Design and Analysis of Algorithms for Solving *n* Coins Problem

A Thesis Submitted to the University of North Bengal

For the Award of

Doctor of Philosophy

in

Computer Science and Application

by

Joydeb Ghosh

Principal Guide

Prof. R. K. Samanta

Co-Guide

Prof. R. K. Pal

Department of Computer Science and Application

University of North Bengal

April 2018

Dedicated

To My Family

DECLARATION

I hereby declare that the thesis entitled "**Design and Analysis of Algorithms for Solving *n* Coins Problem**" has been prepared by me under the joint supervision of **Dr. Ranjit Kumar Samanta (Principal-Guide)**, and **Dr. Rajat Kumar Pal (Co-Guide),** Professor, Department of Computer Science and Engineering of the University of Calcutta, and carried out at the Department of Computer Science and Application of the University of North Bengal.

Joydeb Ghosh.

Joydeb Ghosh
Department of Computer Science and Application
University of North Bengal
Darjeeling – 734013
West Bengal

CERTIFICATE

I certify that Mr. Joydeb Ghosh has prepared the thesis "**Design and Analysis of Algorithms for Solving *n* Coins Problem**" for the award of the Ph.D. degree of the University of North Bengal, under the joint supervision of myself, **Dr. Ranjit Kumar Samanta (Principal-Guide)**, and **Dr. Rajat Kumar Pal (Co-Guide),** Professor, Department of Computer Science and Engineering of the University of Calcutta, and carried out his work at the Department of Computer Science and Application of the University of North Bengal.

Dr. Ranjit Kumar Samanta
Retired Professor
Department of Computer Science and Application
University of North Bengal
Darjeeling – 734013
West Bengal

URKUND

Urkund Analysis Result

Analysed Document:	JOYDEB GHOSH_Final Thesis.pdf (D37861976)
Submitted:	4/23/2018 11:50:00 AM
Submitted By:	nbuplg@gmail.com
Significance:	13 %

Sources included in the report:

https://www.duo.uio.no/handle/10852/10325
https://hal.archives-ouvertes.fr/hal-01294013v2
https://aaltodoc.aalto.fi/handle/123456789/11451
http://www.math.lsa.umich.edu/~hderksen/ProblemSolving/PS10.pdf
https://en.wikipedia.org/wiki/Coin_problem
https://docslide.us/documents/final-solution-of-n-coins-problem.html
http://www.math.kent.edu/~soprunova/64091s15/weight15.pdf
http://coinsguide.reidgold.com/counterfeits.html
http://www.moshe-online.com/tutor/coin_new/coin.html
https://docslide.net/documents/optimal-search-procedure-on-coin-weighing-problem.html
https://docslide.com.br/documents/optimal-detection-of-two-counterfeit-coins-with-two-arms-balance.html
http://www.coinauthentication.co.uk/newsletter10.html

Instances where selected sources appear:

103

R Samanta

joydeb ghosh

Abstract

Counterfeit or anomaly detection problem is a very eminent domain that is of immense importance in the field of Mathematics, Computer Science as well as in security concern. Beyond the theoretical field, computing a solution of the problem has huge significance in commercial sphere as well as to prevent forgery in different fields. Most frequently, we have to deal with a huge collection of information associated with various fields that are defined by some parameters. The parameters hold explicit values depending on the problem instance, and there may subsist some data that deviate from these specific values. Consequently, we consider these data items to be anomalous. Now the problem is to distinguish the counterfeited data by means of some testing mechanism of the items. In some of the variants, we even do not know the actual value of the parameter of the standard items as well as the anomalous items. As this problem considers the number of times the testing method is required as its requisite cost, our objective is to minimize that number to minimize the overall cost.

Counterfeit coins problem is the most important variant of these problems, where among a set of identical coins, the weight of some of them deviate from that of a standard coin, i.e. either heavier or lighter, and the objective is to recognize the forged coins by means of weighing. Moreover, the additional goal is to minimize the number of weighings for which it is sufficient to determine the defective coin(s) using only an equal arm balance when the number of odd coins is precisely known. The problem gets complicated with the increase in the number of counterfeit coins in a set. If P is the number of counterfeit coins in a set of n coins, it is not only sufficient to consider whether the counterfeit coins are heavier or lighter in comparison to a genuine coin individually, but we must also consider their mutual relation like equally heavier or lighter, unequally heavier or lighter, etc. Considering all these allied facts, a set of algorithms have been presented in this thesis for solving all variations of single and two counterfeit coins problem.

Keywords: Algorithm, Anomaly Detection, Counterfeiting, Coins Problem, Computational Complexity, Decision Tree.

.

Preface

Weighings, possibly the oldest and surely most extensively well-known problem concerns in the search for a counterfeit coin. In a set of *n* coins, there is preciously one fake coin with the possibility of heavier or lighter. We want to recognize the counterfeit coin with few weighings with an equal arm balance. The ancient puzzle of counterfeit coin serves as a prime example in discussing various aspects of search problems and, in past few decades, searching has become the special area of interest of computer science and pure mathematics. The rise of high-speed computer and the analysis of algorithms greatly contributed to the rapid development of combinatorial analysis.

The purpose of the thesis is to give an introduction to the basic ideas of single counterfeit coin problem in chapter -1. To find an optimal algorithm for two counterfeit coins problem is very difficult, though we present the collection of most interesting instances of two counterfeit coins problem. In chapter-2 we emphasize the existing counterfeit coin problem-solving techniques. In chapter 3 we presented Generalised Algorithms for solving *n* Coins problem. In chapter-4 we provide Algorithms for Two Counterfeit Coins Problem while both are heavier or both are lighter. In chapter-5 we gave new Algorithms for $\Delta(w(H)) > \Delta(w(L))$ and $\Delta(w(H)) < \Delta(w(L))$.In chapter-6 we established another new Algorithm for Solving Two Coins Counterfeiting with $\Delta w\ (H) = \Delta w\ (L)$. In chapter-7 we have provided the applications and future scopes of our algorithms.

Acknowledgement

I would first like to thank my Principal Guide, **Dr. Ranjit Kumar Samanta** of the Department of Computer Science and Application at the University of North Bengal. The door to Prof. Samanta's office was always open for me whenever I faced any problem relating my research.

I must also express my deepest gratitude to my Co-Guide, **Dr. Rajat Kumar Pal** of the Department of Computer Science and Engineering, University of Calcutta, for providing me with consistent support and encouragement throughout my years of research. This thesis would not have been possible without Dr. Pal.

I am also grateful to **Dr. Debasis Dhal** of Assam University, Silchar, for his valuable comments and cooperation in framing most of the chapters of my thesis. Furthermore, I would like to thank the scholars of Dr. Rajat Kumar Pal, especially, **Ms. Piyali Dutta**, **Mr. Arpan Chakraborty**, and **Mr. Abhinandan Khan**, for their active participation in the thesis writing. They supported my work and helped me to get results of better quality.

I would also like to express my appreciation towards my parents, my wife and our daughter Sohini, for providing support and continuous encouragement throughout my years of research. Last but not least, I would like to thank all the teachers and the non-teaching staff, both at the Department of Computer Science and Application of the University of North Bengal and the Department of Computer Science and Engineering of the University of Calcutta, for their cooperation.

Joydeb Ghosh

(Joydeb Ghosh)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction to Counterfeit Coins Problem

1.1 Introduction

Many real-world problems can be formulated as combinatorial problems. The combinatorics underlying a problem is not always evident, but once captured leads to a clean and elegant mathematical model. For that reason, the first approach too many problems consist of separating the practical aspects of the problem from its combinatorial characteristics. We may be, more or less, conscious of this mental process. A kid who plays chess is most probably unaware of the combinatorics of the game, and therefore, is not able to cleverly evaluate the consequences of his/her moves. A more expert player can predict in advance the possible configurations the game may evolve into and decide his/her move accordingly, or even anticipate who will be the winner. Combinatorics has received an extraordinary impulse during the past few decades. Due to the relevance of its practical applications, combinatorics has evolved from a small mathematical discipline practised by a very restricted number of scholars, into a well-structured theory with a complex hierarchy of notions and results. Computer science is perhaps the field where combinatorial methods have been mostly applied. This is not surprising since computers' world is discrete and classical mathematics must be "discretized" to be implemented on computers. In addition to classical methods, like the pigeonhole principle, the double-counting argument, the Ramsey theorem, etc., recently techniques like the probabilistic method and the linear algebra method have found striking applications in theory of computing. Combinatorial techniques have shown a dramatic power in solving a wide variety of computer science problems, ranging from computational complexity to operation research, computational biology, the design of algorithms, cryptography, zero-knowledge and so on.

A contribution of the present thesis is to provide evidence of the versatility of the combinatorial reasoning. The problems discussed in the thesis attain to two distinct areas of research, Search Theory and Algorithm, which share a deep connection with

combinatorics. It has been this common aspect to solicit our interest and motivate our research in both areas. The Counterfeit Coins Problem is a well-known complex search problem in combinatorics as well as in computer science. It can be related to the Data Structure (such as a binary tree), Algorithm, Graph Theory, therefore researching this problem is meaningful. The counterfeit coins problem can be described as given a set of n look-alike coins containing one counterfeit which is a bit heavier or lighter than the genuine coins. The aim is to find the counterfeit coins, with minimum numbers of weighing w (trials) with a simple arm balance scale. While a balance scale provides information about the counterfeit coins by comparing the weights of the two subsets of coins, we can also detect counterfeit coins by a spring scale which provides information by weighing a subset of coins instead of talking about the balance scale and spring scale, we will use the more general term of the comparison-type device and test-type device. Simple arguments show that, this is achievable if and only if $n \leq 3^w$, for this problem if the quantity of coins is less, it is not difficult to be solved, but if generalizing the quantity of coins to n (such as 500 or more) ask how to determine quickly the minimum number of comparison required and how to weight and needs look for general law and theoretical basis, and then it is not easy to solve. More interesting versions are obtained by varying the amount of information available to us that is whether we have access to some additional coins that are known to be genuine, and the amount of information we seek, that is whether we would like to know if the counterfeit coin is heavier or lighter than the genuine coins. At first, we will consider the versions obtained by changing the answers to the following four questions for the single counterfeit coins, two of which deals with the amount of information we have and the other deals with the amount of information we seek. Four questions are:

Q_1 : Do we know in advance if the counterfeit coin is heavier or lighter than the genuine ones?

Q_2 : Do we know if there is a counterfeit coin?

Q_3 : Do we have access to additional coins that are known to be genuine?

Q_4 : Do we want to know if the counterfeit coin is heavier or lighter than the genuine ones?

Since the answer to $\boldsymbol{Q_4}$ is definitely "yes" when the answer to $\boldsymbol{Q_1}$ is "yes", the combination of different answers to these questions generated the 12 different versions. A solution to one of these problems includes two pieces of information: (*i*) the weighing descriptions, that are the set of coins put on each side of the balance scale in each weighing, and (*ii*) a map from their results in the answer; that is the identity of the counterfeit coins and whether it is lighter or heavier than the genuine ones if such information is desired. In a sequential solution, the description of a trial can be affected by the results of the previous ones. In a non-sequential solution, the trials are independent.

Table 1.1: The optimal relationship between *n* and *w* for which different variants of the single counterfeit coins problem have solutions.

Name	$\boldsymbol{Q_1}$	$\boldsymbol{Q_2}$	$\boldsymbol{Q_3}$	$\boldsymbol{Q_4}$	**Number of Coins**
$\boldsymbol{P_1}$	Yes	Yes	Yes	Yes	3^w
$\boldsymbol{P_2}$	Yes	Yes	No	Yes	3^w
$\boldsymbol{P_3}$	Yes	No	Yes	Yes	$3^w - 1$
$\boldsymbol{P_4}$	Yes	No	No	Yes	$3^w - 1$
$\boldsymbol{P_5}$	No	Yes	Yes	Yes	$(3^w - 1)/2$
$\boldsymbol{P_6}$	No	Yes	Yes	No	$(3^w + 1)/2$
$\boldsymbol{P_7}$	No	Yes	No	Yes	$(3^w - 3)/2$
$\boldsymbol{P_8}$	No	Yes	No	No	$(3^w - 1)/2$
$\boldsymbol{P_9}$	No	No	Yes	Yes	$(3^w - 1)/2$
$\boldsymbol{P_{10}}$	No	No	Yes	No	$(3^w - 1)/2$
$\boldsymbol{P_{11}}$	No	No	No	Yes	$(3^w - 3)/2$
$\boldsymbol{P_{12}}$	No	No	No	No	$(3^w - 3)/2$

The generalized counterfeit coins problem [1, 4, 41] is that we are given *n* coins where we know that a coin may be counterfeit. Along with it may be given one or many true (or genuine) coins. Our problem is a distinctive case of the counterfeit coins problem where we are given eight coins, and we know that one of the coins is false and distinguishable by weight. This algorithm is based on the existing classical solution of a

more formal instance of this problem, known as the Eight Coins Problem, where there eight coins are given among which only one coin is false. The solution is shown in Figure 1, which is a *decision tree*. The tree in this figure represents a set of decisions by which we can get the solution to our problem. We use H or L as a suffix to represent the counterfeit (or false) coins as *heavier* or *lighter*, respectively. In the solution of the eight coins problem in the form of a decision tree in Figure 1, each internal vertex (other than leaf vertices) represents a comparison between a pair of sets of coins using an equal arm balance.

In the decision tree in Figure 1, we consider three coins on one side of the equal-arm balance, which is the root of the tree. If the weight-sums are equal, then surely each of these coins is a true coin, and the false coin is either G or H with their possibilities either heavier or lighter. In this situation, as A is a true coin, which we use in comparing separately with G and H after a comparison between G and H themselves. If the weight-sum containing coins A is greater than the weight-sum containing coins D (i.e. A+B+C > D+E+F), then we can say that the false coins belong to these six coins only, where either A is heavier, or B is heavier, or C is heavier, or D is lighter, or E is lighter, or F is lighter. At the same time in this situation, both G and H are true coins. The next comparison is highly important in order to make the height of the tree as small as possible; we do three things as follows: (*i*) keep a pair of coins A and E on their own sides, (*ii*) another pair of coins B and D interchange their sides, and (*iii*) the remaining pair of coins C and F are removed from the comparison. Therefore, subsequently, the weight-sum of A and D (i.e. A+D) are compared with the weight-sum of B and E (i.e. B+E). Three cases may arise.

Case 1: If weight-sums are equal, then either C or F is a false coin (as these coins are removed from this comparison), where either C is heavier, or F is lighter (following the root of the tree). Therefore, we compare C with A (a true coin). If the weight of C is more than the weight of A, then C is heavier; otherwise, these two coins must have the same weight resulting F as lighter.

Case 2: If A+D > B+E, then certainly either A or E is a false coin, as these coins are kept in their own sides, and the logical relation (following the root of the tree) is unchanged. Here either A is heavier, or E is lighter. Therefore, we compare B (a true coin) with A. If

the weight of A is more than the weight of B, then A is heavier; otherwise, these two coins must have the same weight resulting E as lighter.

Case3: In a similar way, if A+D < B+E, then definitely either B or D is a false coin, as these coins have interchanged their sides and the logical relation (following the root of the tree) has also changed. Here, either D is lighter, or B is heavier. Therefore, we compare B with A (a true coin). If the weight of B is more than the weight of A, then B is heavier; otherwise, these two coins must have the same weight resulting D as lighter. Similarly, we may consider the case of weight-sums following the remaining branch of the root of the tree, where A+B+C < D+E+F. Here either A is lighter, or B is lighter, or C is lighter, or D is heavier, or E is heavier, or F is heavier. The remaining part of the subsequent comparisons is done in a similar way as it is explained above and shown in Figure 1.

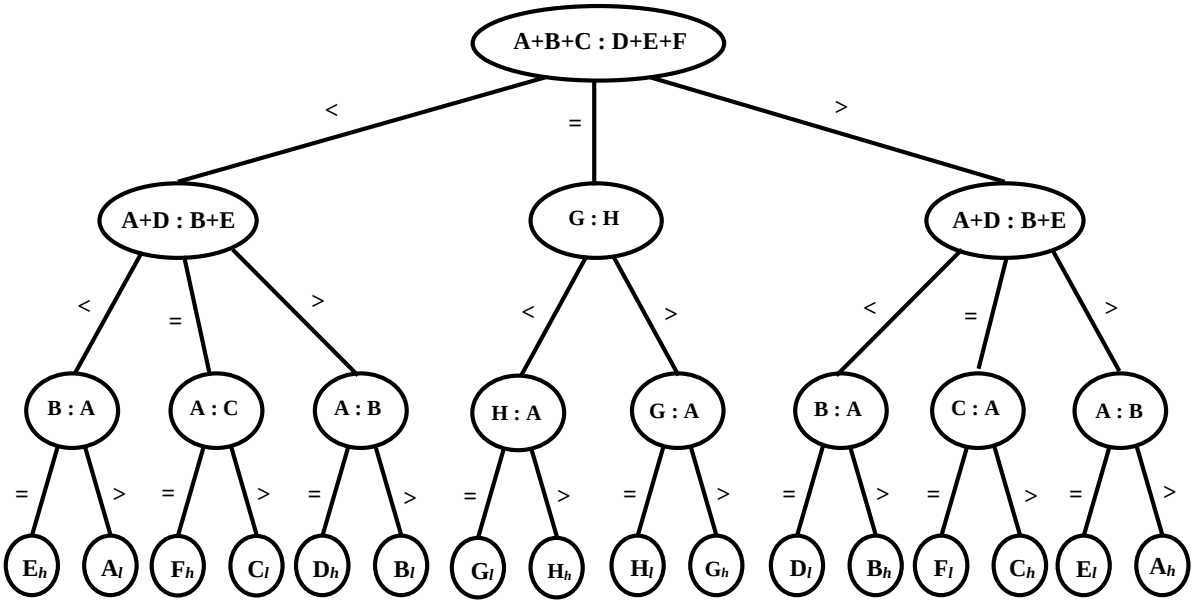


Figure 1: The existing solution of the eight coins problem in the form of a decision tree.

1.2 A Brief Background on Counterfeit Coins problem

In January of 1945, the following problem appeared in the *American Mathematical Monthly*, contributed by E. D. Schell: *You have eight similar coins and a beam balance. At most one coin is counterfeit and hence underweight. How can you detect whether there is*

an underweight coin and if so which one, using the balance only twice? Since such counterfeit coins problem is today as much a part of Mathematics and Computer Science. The problem was popular on both sides of the Atlantic during World War II [1] it has even been suggested that it should be dropped over Germany in an attempt to sabotage their war effort in solving [11, 37, 40]. Kaplansky, Neugebauer, and Panel gave the following general solution to the problem of underweight counterfeit coins: If $3^{n-1} \leq N < 3^n$ then n weighing sufficient to show if there is a counterfeit coin among N. If it is known that the counterfeit coins exist, then n weighing will identify the coins among $3^{n-1} \leq N < 3^n$. Dyson [30] gave an elegant solution using ternary labels when it is not known if the counterfeit coins are heavy or light.

1.2.1 Eight Coins Problem

In this problem, eight coins are given of which only one is false, either heavier or lighter. The aim is to find out the false coins using a minimum number of comparisons amongst the coins and to determine the characteristic feature of the false coins whether it is heavier or lighter in comparison to each of the true coins. Now the question is what the minimum number of comparison is required since each coin has the two possibilities: heavier and lighter, so the total possibility is 16, if we consider w is the total number of comparison then, $w = \log_3 16 \cong 3$.

1.2.2 Twelve Coins Problem

Among twelve similar coins, there is one counterfeit. It is not known whether the coin is lighter or heavier than a genuine one all the genuine coins weigh the same. Using three weighings on a pan balance, how can the counterfeit be identified and, in the process, determined to be lighter or heavier than a genuine coin?

1.2.3 *n* Coins Problem

The N coins problem describes as there is a set of n coins, out of this n coins one coin is counterfeit with the possibility of heavier or lighter remaining $(n-1)$ coins are genuine. We must identify the counterfeit coin with $w = \log_3 2n$ weighings; if one additional genuine coin

is given then a minimum number of weighings is $w = \log_3(2n+1)$, if the counterfeit coin is lighter than the minimum number of weighings is $w = \log_3 n$.

1.3 Difficulty Level of the Counterfeit Coins Problem

A common phenomenon in combinatorial search theory is that while it is often straightforward to find an optimal procedure searching for one object among a small number of objects, it is immensely more difficult to search optimal procedure for any number of objects, because the number may be odd or even finding counterfeit coins among any numbers of given coins suing single arm balance can be represented as a ternary decision tree, since the balance can be top left, top right, or balance on any given measurement. A decision tree can be *adaptive* or *oblivious.* Adaptive algorithms can solve counterfeit coins problem in O(log *n*), but a simple change in the problem solve the problem in O((log $n)^2$) measurements. Bellman and Glass [12] studied the problem of identifying two counterfeit coins problem among *n* coins with a balance scale. They wrote, "A small amount of analysis discloses the enormous difference in complexity between the one coin and two coins problem" solving two counterfeit coins problems is much more difficult because there are seven different versions of the problem such as both are heavier. Two heavier false coins may be equally or unequally heavier. If we denote them as H_1 and H_2 and their weights as $\omega(H_1)$ and $\omega(H_2)$, we can define these cases as $\omega(H_1) = \omega(H_2)$ (equally heavier) and $\omega(H_1) > \omega(H_2)$ (unequally heavier and we assume that H_1 is heavier than H_2). Two lighter false coins may be equally or unequally lighter. If we denote them as L_1 and L_2 and their weights as $\omega(L_1)$ and $\omega(L_2)$, we can define this case as $\omega(L_1) = \omega(L_2)$ (equally lighter) and $\omega(L_1) < \omega(L_2)$ (unequally lighter and we assume that L_1 is lighter than L_2). Let the true coins is denoted as T, and weight of the true coins is $\omega(T)$. Therefore, heavier and lighter coins are denoted as H and L, respectively. Then, $\omega(H) - \omega(T) = \Delta(\omega(H))$ and $\omega(T) - \omega(L) = \Delta(\omega(L))$ the following situations may arise $\Delta(\omega(H)) > \Delta(\omega(L))$, $\Delta(\omega(H)) < \Delta(\omega(L))$, $\Delta(\omega(H)) = \Delta(\omega(L))$.

1.4 Motivations behind the Present Work

The classic puzzle of the counterfeit coin has long served as a stiff test of one's reasoning

power and ingenuity. The problem of the counterfeit coin has circulated in many guises over the years. We have encountered versions involving 8, 10, 12, or 13 coins. In many cases, it is also to be determined whether the counterfeit coin is lighter or heavier than the rest. Finding an algorithm to solve the general problem of determining the minimum number of weighing given *n* coins, one of which is counterfeit, is a popular computer programming exercise. In its standard form, the problem concerns 12 coins identical in size, shape, and appearance. One coin, however, is counterfeit, having a slightly different weight than the other 11 coins. Using only a two-pan balance, what is the smallest number of weightings that would guarantee that we would find the counterfeit coin?

It is said that during the Second World War the English dropped leaflets containing this problem on the German troops with the goal of distracting and thus disorganizing them; supposedly, they wasted 40,000 man-hours on solving it [1]. The classical problem of false coins has recently found applications in the Theory of Coding and Information for detecting errors in code. For most of the twentieth century, Hong Kong had a significant circulation counterfeit coin problem. This can be glimpsed by reading the 1936 Report of the Government Analyst. The introduction of the relatively high value $10 coin towards the end of the century made this problem worse [20, 52]. Every effort was made to incorporate all the most modern anti-counterfeiting features into the coin. This included using a Bimetal Coin and an interrupted fine milled edge. As the problem worsened the material of the coin was changed, but this did not appear to stem the problem. By 2001 the number of counterfeit $10 coins withdrawn had quadrupled and the first quarter figures (of 160,000) for 2002 showed a threefold increase over that previous year [53].

The most frequently seen counterfeit or altered U.S. coins, according to PCGS's 2004 book *Coin Grading and Counterfeit Detection*, include:

- 1856 Flying Eagle Cent
- 1909-S VDB Lincoln Cent
- 1955 double-die Lincoln Cent
- 1916-D Mercury Dime

- Cincinnati commemorative half dollar
- 1804 Bust dollar (a million-dollar rarity)
- 1893-S Morgan dollar
- Saint-Gaudens high-relief double eagle

The above examples show how rampant is the counterfeit coin problem. As such it becomes necessary to find out ways to detect the counterfeit coins. There are various quantitative tests that can be performed to help with counterfeit detection. Often, any one test or several tests are not conclusive, but they can provide important information.

1.5 Objectives of the Present Work

In this thesis, we have exhaustively studied each existing counterfeit coins problem based algorithm that we have obtained along with their difficulty level. We have tried to identify and minimize the drawbacks of the existing algorithm in our present work. The foremost objectives of the proposed generalized algorithm for solving *n coins problems* are highlighted as follows —

(1) To develop an algorithm for counterfeit coins problem that is entirely for any value of *n* and the possibility of the counterfeit is heavier or lighter; it follows some distinct deterministic steps. Whereas the other algorithms, we have studied so far, first they declare the possibility of the counterfeit coin.

(2) All the different counterfeit coin related algorithms have been applied to the problem based on adaptive and non-adaptive search technique. It often does not lead to deterministic steps for finding the solution. As there is no standard definition of difficulty levels these methods are applied to self-addressed problem by changing the original essence of the problem, therefore, to develop a technique which will follow some deterministic steps to find out the valid solution of the problem without depending on various versions of a probable list of numbers and difficulty levels is another important aim of the present work.

(3) In the entire existing two counterfeit coins problems [10, 17, 31], they proposed algorithms for finding only heavier or lighter coins, but they did not give any clearer view about the different possibility of the counterfeit coins; however, we have proposed the crystal clearer view of the various natures of coin counterfeiting and their combinations, which is novel for combinatorial point of view.

(4) To develop a methodology, which can right away decide for solving n coins (where $n = 2^P$ where P is any positive integer ≥ 3) problem using decision tree where there is only one false coin.

(5) To develop an algorithm for counterfeit coins problem, this can straightway be applied for solving many input sizes.

(6) To cultivate an entire decision tree representation of the counterfeit coin problem, this is novel in the sense that nobody has ever tried and/or developed such a depiction earlier.

(7) Identifying some new spheres of application of solving a counterfeit coin problem.

(8) To have a brief study of different counterfeit coins problem generation algorithms.

1.6 Achievements out of the Work Done

In a modest way, the following contributions have been made in this thesis work:

(1) We have studied exhaustively all the different existing counterfeit coins problem related algorithms.

(2) We have modified the classical solution of eight coins problem. In our new solution we reduced the external path length by 4, total number of comparison by 3, maximum number of coin in a comparison by 2, and the average height of the tree by 0.25, which is novel in the context of data structure and analysis of algorithms.

(3) We have exhaustively studied different algorithms based on the single counterfeit coins problem, and we have overcome the problem to solve all the possibility of counterfeit coins heavier and lighter.

(4) We have developed an algorithm for solving *n* coins problem using decision tree, with the computational complexity O(*n*).

(5) We have identified different application domains of the counterfeit coin problem.

(6) We have developed a new algorithm for solving two counterfeit coins problem with the possibility of equally heavier and equally lighter.

(7) We propose a new algorithm for two counterfeit coins with the possibility un-equalled heavier and un-equally lighter.

(8) We have also developed a new algorithm where one coin is heavier and the other is lighter, but the difference between the heavy and the original coin is greater than the difference between the light and the original coin, that is $\Delta(\omega(H)) > \Delta(\omega(L))$. We have also proposed a new algorithm where one coin is heavier and the other is lighter, but the difference between the heavy and the original coin is less than the difference between the light and the original coin, that is $\Delta(\omega(H)) < \Delta(\omega(L))$.

(9) We have also developed a new algorithm in the form of a decision tree for solving two coins counterfeiting with the difference between the heavy and the original coin is equal to the difference between the true and the lighter coin, that is $\Delta\omega(H) = \Delta\omega(L)$.

1.7 Outline of the Thesis

The thesis consists of seven chapters. In Chapter 2, we have discussed the related background and the nature of the problem. We have also made an extensive study on different existing algorithms for solving the counterfeit coins problem, along with their comparative advantages and inadequacies.

In Chapter 3, we have proposed two new solutions of the eight coins problem, and the theoretical lower bound of the single counterfeit coin problem with its associated difficulty levels. In this chapter, we have also considered the general n coins problem, where only one coin is fake out of n identical given coins.

In Chapter 4, we have discussed the necessary and sufficient conditions for the two counterfeit coins problem, and we have developed algorithms for different combinations of heavier and lighter counterfeit coins, and calculated the computational complexity and related experimental results.

Chapter 5 considers the two counterfeit coins problem where the difference between the heavier and the original coin is unequal to the difference between the lighter and the original coin. This chapter also includes all allied experimental results.

In Chapter 6, we have developed a new algorithm for solving two coins counterfeiting, where the difference between the heavier and the original coin is equal to the difference between the true and the lighter coin, and calculated the computational complexity and corresponding experimental results.

The thesis is concluded in Chapter 7, where we have discussed some applications of counterfeit coins problem and the impact of the counterfeit coins problem on the economy. This chapter summarises the works included in this thesis, and also draws attention to some plausible open problems as further research scopes.

Chapter 2

Existing Literature on Counterfeit Coins Problem: A Review

2.1 Overview

In this chapter, we are going to study the different existing theory for solving counterfeit coins problem. This chapter is organized into four sections. In the first section, we have briefly discussed the nature of the problem. In Section 2.2, we have discussed different existing counterfeit coin problem solving techniques. In Section 2.3, we have talked about the total number of different counterfeit coin variants. A summary of the chapter is presented in Section 2.4.

2.2 Nature of the Counterfeit Coins Problem

We are given 80 coins of the same penny; we want to know that one of them is counterfeit and that it is lighter than the others. Establish the fake coins by using four weighings on a pan balance; now suppose that it is identified that there is one counterfeit coin in a set of n alike coins. What is the least number of weighings essential to identify the counterfeit? Among twelve alike coins, there is one counterfeit. It is not known whether the counterfeit coin is heavier or lighter than the genuine one. All the true coins weight is the same. By means of three weighings on a pan balance, how can the fake be identified and, in the process, determined to be lighter or heavier than a true coin? We can extend this problem such as there is one fake among 1000 similar coins. It is not known whether the counterfeit coin is lighter or heavier than a genuine one. Here the balance scale has two pans, the right side and the left side. By weighing, we want to say that two equal-sized subsets of coins are placed in the two pans respectively, and the result is one of the following: (1) left side light, i.e. the total weight of coins on the left pan is smaller than that on the right pane; (2) right side light, i.e. the total weight of coins on the right pan is smaller than that on the left

pane; (3) balanced, i.e. the two sides have the same total coin-weight. A general solution will be presented in this section. Before doing so, let us show some simple results.

Theorem 1: *Let S be a set of coins, one lighter than the rest. The least number of weighings on beam balance in which the lighter can be found is the unique n satisfying* $3^{n-1} < S \leq 3^n$.

Proof: Let $|S| = 3^n$, then for the primary weighing, we divide S into three equal sets s_1, s_2, and s_3 of size 3^{n-1} and place s_1 and s_2 on opposite sides of the beam [11]. If the scale does not balance, the light coin is on the light side of the scale, or else, it is in s_3. In any case, we have reduced our problem to finding the light coin in a set of 3^{n-1} progressing in this way, the light coin be located after n weighings. If $S \leq 3^n$, a similar method can be followed, placing equal sets of s_1 and s_2 coins on the scale, leaving a set s_3 of at most 3^{n-1} coins unweighted. Once more, repetitions will lead us to the light coin in at most $n-1$ additional steps. On the other hand, a single weighing of any sort cannot do better, then cut the size of the set of fake coin by a factor of 3 this is so because three sets are involved in the process, two on the scale and one off, and the outcome merely distinguishes which of the three sets contains the lighter coin. Thus, if $|S| > 3^{n-1}$, $n-1$ cannot sufficient to find the lighter coin in all cases.

Example: In view of the case of 80 coins, where it is given that the fake coin lighter, we must recognize the fake coin in four weighings. We divide the coins into three sets $s_1 = 3^{(4-1)} = 27$, $s_2 = 3^{(4-1)} = 27$, and $s_3 = 80-54 = 26$. A first weigh trial is made by placing s_1, s_2 on each of the two pans. If the pans do not balance, the fake coin is along with those in the light pan. If the scale is balanced, the counterfeit coin is in s_3 which is unweighted. The problem of detecting counterfeit in $s_3 = 26$ can be reduced to 27 by adding one true coin from 54 coins. For a second weighing we divide 27 coins into three groups s_1', s_2', and s_3' where $s_1' = 9$, $s_2' = 9$, and $s_3' = 9$ placing s_1', s_2' on each of the two pans. If the pans do not balance, the counterfeit coin is in the light pan. If the scale is balanced, the counterfeit coin is in s_3' which is unweighted. Now we will split nine coins into three groups s_1'', s_2'', and s_3'' where $s_1'' = s_2'' = 3$ and placing s_1'', s_2'' on each of the two pans. If the pans do not balance, the counterfeit coin is in the light pan. If the scale balances the counterfeit coin is

in s_3'' which is unweighted. Now we will split three coins in three groups s_1''', s_2''', and s_3''' where $s_1''' = s_2''' = 1$ and $s_3''' = 1$. We place s_1''' and s_2''' on each of the two pans. If the pans do not balance, the counterfeit coin is in the light pan. If the scale is balanced, the counterfeit coin is in s_3''' which is unweighted.

Lemma 1: *If in a set S of coins one coin is a different weight than the rest and each coin is labelled possible heavy or possible light. The least number of weighing on a beam balance in which the odd coin can be found is the unique n satisfying* $3^{n-1} < S \leq 3^n$.

Proof: Perceive that this result is very comparable to Theorem 1, in which every coin can be thought of as being categorized possible lighter since the odd coin is known to be light [11]. In fact, the weighing method of Theorem 1 works in this case, with one restriction. Whenever we place coins on the scale, we must be sure to put equal of possible lighter coins on the two sides, if for example $|S| = 3^n$, we divide S into three sets of size 3^{n-1}, say s_1, s_2 and s_3, placing the same number of possible lighter coins in s_1 and s_2 when s_1 and s_2 are compared to the scale, if s_1 is heavier, the counterfeit coin must among the possible heavier, coin in s_1 or the possible lighter, coins in s_2 which together comprise a set of size 3^{n-1} thus in every case we decrease the size by factor of 3, as in Theorem 1. For where |S| is not a power of 3 a similar process is efficient.

Theorem 2: *Assume there are n coins with probably a light counterfeit coin. Then* $2 \leq n \leq 3^k$ *if and only if one can always tell in no more than w weighings (a) whether a counterfeit coin exists and (b) if so which one is.*

Proof: For $n = 1$, the balance is inadequate and therefore one has no way to tell (a). For $n = 1$, note that there are $(n+1)$ sample points. By the information lo*w*er bound [17], $k \geq \log_3(n+1)$ that is $n < (3^k - 1)$ next; presume $2 < n < 3^k - 1$ one proves by induction on k that w weighings are sufficient to tell (a) whether a fake coin exists and (b) if so which one is. For $w = 1$, n must equal to 2. Place them on two sides; one in each. If balanced, then no counterfeit coin exists. If unstable, then the lighter one is counterfeit. Consequently, one weighing is sufficient. Now, consider $k > 2$ if $3^{(k-1)} < n < (3^{(k-1)} - 1)$, then by the induction theory, $(k-1)$ weighings are sufficient to tell (a) whether a counterfeit coin exist and (b) if

so, which one it is? Thus suppose $3^{(k-1)} < n < (3^k - 1)$. Let $h = \lceil (n - 3^{(k-1)} + 1)/2 \rceil$ evidently $1 < h < 3^{(k-1)}$ and $(n-2h) < (3^{(k-1)} - 1)$ put h coins on each side of the balance. If balanced, then $2h$ coins on balance are true. By the induction hypothesis, additional $(k-1)$ weighings are sufficient to tell (a) whether a counterfeit coin exists and (b) if so which one is. For outstanding $(n-2h)$ coins since there are $2h$ genuine coins in hand, one can still tell (a) whether a counterfeit coin exists and (b) if so which one is counterfeit when $(n-2h) = 1$. If unstable, then the h coins on the lighter side contain a counterfeit coin (a) whether a counterfeit coin exists is the answer. Furthermore, if $h < (3^{(k-1)} - 1)$ than by the induction hypothesis, extra $(k-1)$ weighings are adequate. If $h = 3^{(k-1)}$ then one can still use $(k-1)$ weighings to tell (b) if so which one is. By separating unidentified coins into three equal groups in each weighing.

Corollary 1: *Suppose there are n coins with exactly one counterfeit coin which is light. Then* $2 \leq n \leq 3^k$ *if and only if one can always tell no more than k weighings (b) which one is counterfeit.*

This corollary can be generalized as follows —

Lemma 2: *Suppose there are two groups of coins. The first group* s_0 *has* n_0 *coins with possibly a light counterfeit coin. The second group* s_1 *has* n_1 *coins with possibly a heavy counterfeit coin. Assume that* $s_0 \cup s_1$ *contains exactly one counterfeit coin and there exists additional* $min(1, n_0, n_1)$ *genuine coin which can be used for help. Then* $n_0 + n_1 \leq 3^k$ *if and only if one can always tell in no more than k weighings (b) which one is counterfeit and (c) whether the counterfeit coin is heavier or lighter than a genuine coin.*

Proof: One proves it by induction on k. First, note that if $min(n_0, n_1) = 0$ then the lemma reduces to Corollary 1. Consequently, one may suppose $min(1, n_0, n_1) = 1$. For $k = 1$ there are two cases as follows: For Case 1, where $n_0 = n_1 = 1$, the left side and the genuine coin on the right side of the balance scale [17]. If balanced, and then the one in s_1 is a heavy counterfeit coin. If unbalanced, then the one in s_0 is a light counterfeit coin. In Case 2, $n_0 = 1$ and $n_1 = 2$ (or $n_0 = 2$ and $n_1 = 1$) put the two unidentified coins in s_1 on balance, one on each side. If balanced, then the one in s_0 is a light counterfeit coin. If unbalanced, then the

one on the heavy side of the weighing scales is a heavy counterfeit coin. Next, consider $k > 2$ one may suppose $3^{(k-1)} < n_0 + n_1 \leq 3^k$. Let $h = \lceil (n_0 + n_1 - 3^{(k-1)})/2 \rceil$ then $1 < h \leq 3^{(k-1)}$ and $(3^{(k-1)}-1) \leq (n_0 + n_1 - 2h) \leq 3^{(k-1)}$ since $k \geq 2$, $(3^{(k-1)} - 1) \geq 2$. This enables one to take $2h$ coins from $s_0 \cup s_1$ with even numbers of coins from s_0 and s_1, in that order. Now, put the $2h$ chosen coins on balance such that the two sides contain the same number of coins from s_0 and the same number of coins from s_1. If balanced, then the counterfeit coin is among the $(n_0 + n_1 - 2h)$ cions not on balance. By the induction hypothesis, $k-1$ more weighings are sufficient to tell (b) and (c). If unbalanced, then the counterfeit coin is among coins from s_0 on the lighter side and coins from s_1 in the heavier sides. Thus, the total number of unknown coins is h. By the induction hypothesis, $k-1$ more weighings are enough to solve the problem.

Theorem 3: *Suppose there are n coins with the possibility of a counterfeit coin and there exists one additional genuine coin. Then* $n \leq (3^k-1)/2$ *if and only if one can always tell in no more than k weighings (a) whether a counterfeit coin exists, (b) if so which one is and (c) whether the counterfeit coin is heavier or lighter than a genuine coin.*

Proof: There are $(2n+1)$ sample points. By the information, lower bound, $k \geq \lceil log_3(2n+1) \rceil$ [17, 27]. Thus, we have $n \leq (3^k-1)/2$. Next, assume $n \leq (3^k-1)/2$ one show by induction on k that k weighings are sufficient to tell (a), (b), and (c). For $k = 1$, one must have $n = 1$. Thus, one weighing is adequate with helping of the supplementary genuine coin. By taking into consideration $k > 2$, let $h' = (n-3^{(k-1)}/2)/2$ and $h = \lfloor h'/2 \rfloor$ then $h' \leq 3^{(k-1)}$ and $n-h' = (3^k-1)/2$ plus h unknown coins on the left side and $h' - h$ unknown coins on the right side of the balance. When $h' - h > h$, put one genuine coin on the left side, too. If unbalanced, then the h' unknown coins on balance contain exactly one counterfeit coin which is light when it is on the light side and is heavy when it is on the heavy side. By Lemma 2, $k-1$ more weighings are enough to solve the problem. If balanced, then all coins on balance are genuine, and one needs to deal with only $n-h'$ coins not on balance. By the induction hypothesis, $k-1$ more weighings are enough to solve the problem.

Theorem 4: *Suppose there are n coins with the possibility a counterfeit coin. Then* $3 < n < (3^k-3)/2$ *if and only if one can always tell in no more than k weighings (a) whether a counterfeit coin exists, (b) if so which one is, and (c) whether the counterfeit coin is heavier or lighter than a genuine coin.*

Proof: Presume that k weighings are sufficient and that in the first weighing, each side of the weighing scale has x coins. If the result of the first weighing is balanced [17, 22], then there are $(2(n-2x) + 1)$ likely sample points, $\log_3(2(n-2x) + 1) \leq k-1$, i.e. $(n-2x) \leq (3^{k-1}-1)/2$ if the outcome of the first weighing is unbalanced, then there are $2x$ likely sample points, by the information lower bounds $\log_3(2x) \leq k-1$, i.e. $2x \leq 3^{k-1}$. Note that $2x$ is an even number thus $2x \leq 3^{k-1}-1$ therefore $n \leq (3^{k-1}-1)/2 + (3^{k-1}-1) = (3^k-3)/2$. Furthermore, if $n = 1$, then one has no way to do any weighing; if $n = 2$, then one has simply one way to do weighing which can tell (a) but not (b) and (c). Next, let us presume $3 \leq n \leq (3^k-3)/2$ one can prove by induction on k that k weighings are sufficient to tell (a), (b), and (c). For $k = 2$, one must have $n = 3$. Put two coins in the balance, one on each side. If balanced, then the two coins on weighing scale are genuine, and only one coin is still unidentified. Hence, by Theorem 3 one more weighing can solve the problem. If unstable, then the one not on balance is authentic. Now, consider $k > 3$. If $n \leq (3^k-3)/2$, then by the induction hypothesis, $k-1$ weighings are adequate. Thus, one may guess that $(3^{k-1}-1)/2 \leq n \leq (3^k-3)/2$. Let $h = max(1, \lceil (n - \frac{1}{2}(3^{k-1}-1)/2 \rceil)$ then $1 \leq h \leq (3^{k-1}-1)/2$ and $1 \leq n-2h \leq (3^{k-1}-1)/2$. Put $2h$ coins on the balance, h coins on both side. If balanced, then there remain $n-2h$ coins unidentified; they are not on balance. By Lemma 2, $k-1$ more weighing is enough to solve the problem. If unequal, then there remain $2h$ coins unidentified; they are on balance. By Lemma 2, $k-1$ more weighings are sufficient.

Theorem 5: *Presume there are n coins with the possibility of a fake coin. Then* $3 \leq n \leq (3^k-3)/2$ *if and only if one can always tell in no more than k non-adaptive weighings (a) whether a counterfeit coin exists, (b) if so which one is and (c) whether the counterfeit coin heavier or lighter than a authentic coin.*

Proof: The proof can be finished by implementing *k* weighings, in the proof of Theorem 5, non-adaptive. To give explanation this, consider an instance that there are twelve coins with possibly one counterfeit coin. A sequential algorithm for it is shown in Figure 2.2, where 1, 2, 3, 4 : 5, 6, 7, 8 represents a weighing with coins 1, 2, 3, 4 on the left side and coins 5, 6, 7, 8 on the right side. The weighing scale has three weighing outcomes, balanced, left side light and right-side light, respectively. 12_L and 12_H denote respectively the outcomes that coin 12 is a light counterfeit coin and that coin 12 is a heavy counterfeit coin, in the decision tree in Figure 2.2; we consider four coins on the side of the equal arm balance in the first weighing and three coins in second weighings [17], one coin in third weighings, respectively.

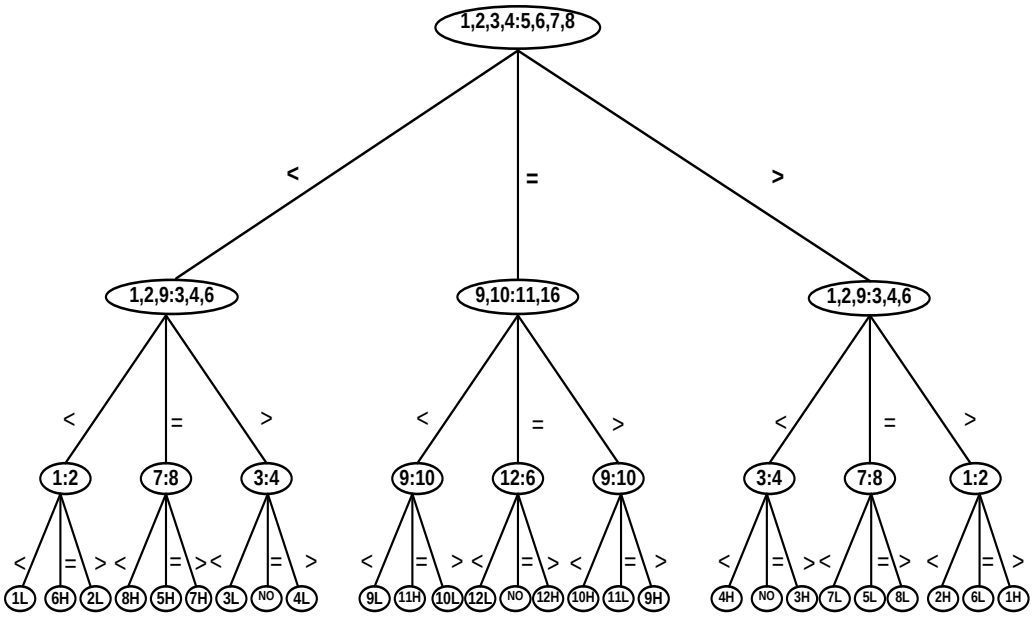


Figure 2.1: The solution of the twelve coins problem in the form of a decision tree.

2.3 A Study on Different Existing Counterfeit Coins Problem Solving Techniques

In this section, we have made a comprehensive study on different counterfeit coins problem solving techniques. This work is published in [1, 2, 3, 4] a natural generalization of our counterfeit coins problem straight away comes to mind. Suppose in a set of *n* coins there

are d defective (heavier) coins and $n-d$ good coins. The weight of the good coins is same as is the weight of all defective coins. Note that, by interchanging the roles of good and defective we may suppose $d \leq n/2$. Let λ be the weight of a good coin and μ the weight of a defective coin, if $d\mu < (d+1)\lambda$, i.e. $\mu < (d+1)\lambda/d$, then it is without difficulty seen that larger of two numerically not the same sets will always be the heavier. Therefore if we assume $\mu < (d+1)\lambda/d$, as we shall do from now on, the information can only be gained when we weigh equal-sized sets alongside each other. This universal defective-coin problem is apprehensive. Even for $d = 2$, the precise answer for all n is not known. Let us see where the extra complexity arises. In single coin problem we recognize after a weighing $A : B$ in which three sets A, B, $S - (A \cup B)$, the defective coin lies. Now suppose $d \geq 2$ and the scale balance weighing A against B. Then we only know that A and B hold the same number of defective coins but, in general, not how many they contain. in the same way, if $A < B$ then we can simply be certain that B contain additional defective than A. Still as the subsequent result essentially Tošić [10] suggest, the worst-case cost will most likely be very close to the information-theoretic bound for all n and d.

Proposition: *let $L^2(n)$ denote the worst-case cost of our weighing problem when precisely two of $n \geq 2$ coins are recognized to be faulty. Then $\lceil \log_3 {}^nC_2 \rceil \leq L^{(2)}(n) \leq \lceil \log_3 {}^nC_2 \rceil + 1$, and equality is attained on the left-hand side for infinitely many n's.*

Proof: Our exploration domain S consist of all likely pairs [21, 23] $\{i, j\} \subseteq \{1, 2, ..., n\}$; consequently, $|S| = {}^nC_2$. It is straightforward to ensure that $n > 3^L$, let $L = 4$. then $n > 81$ implies ${}^nC_2 > 3^{2L-1}$, ${}^nC_2 = 3240$, $3^{2L-1} = 3^7 = 2187$, and $n > 2.3^L$ imply ${}^nC_2 > 3^{2L}$. The information-theoretic bound that yields for $L \geq 0$:

i) $L^2(n) \geq 2L$ for $n > 3^L$

ii) $L^2(n) \geq 2L + 1$ for $n > 2.3^L$

Now we show, on the contrary, that for $L \geq 0$,

i') $L^2(n) \leq 2L + 1$ for $n \leq 2$

ii') $L^2(n) \geq 2L + 2$ for $n > 3^{L+1}$

We prove (i′) and (ii′) at the same time by induction. For $L = 0$ the assertions are trivial, so suppose $L = 1$. To authenticate (i′) we weigh in our first test two sets A, B of cardinality $\lfloor n/2 \rfloor$. If there is balance, in symbols $A = B$, subsequently A and B must each contain a defective coin (while $|S - (A \cup B)| \leq 1$) which can then be determined independently with $2\lceil \log_3 \lfloor n/2 \rfloor \rceil \leq 2L$ additional weighings. If on the other hand, one of the sets is heavier, say A, in symbols $A > B$, then the two defectives are contained in $S–B$. In view of the fact that $|S–B| = \lceil n/2 \rceil \leq 3^L$, we finish off by induction on (ii′) that yet again $2L$ more weighings will do. To authenticate (ii′) we divide S into three sets A, B, C of cardinality $\lfloor n/3 \rfloor$ each. Then $|S - (A \cup B \cup C)| \leq 2$, if $(S - (A \cup B \cup C))$ contains two u, v, then we obtain $B' = B \cup \{u\}$, $C' = C \cup \{v\}$, or else we set $B' = B$, $C' = C$. Note that all sets (A, B, C, B', C') have size $\leq 3^L$ and also that $|S - (A \cup B')| = |S - (A \cup C')| \leq 3^L$. In our first test we weigh A against B, and in the second test we weigh B' against C'.

Case 1: $A = B$ if $B' = C'$, then $B \notin B'$ and u should be a member of one of the defectives, the other one being in $\overline{C}$ which can be determined with L more tests. If $B' > C'$, then each of A and B contains a heavier coin, so we are finished with $2L$ additional weighings. If $B' < C'$, then both defective are $S - (A \cup B')$ since $|S - (A \cup C')| \leq 3^L$, we apply induction [10].

Case 2: $A > B$ if $B' = C'$, then both defective is in A, and we are finished by induction. If $B' > C'$ then $B \notin B'$ and u must be a member of one of the defectives, while the other one is in A which can be determined with L more weighings. If $B' < C'$, then each of A and C' contains exactly one defective coin and we may find them by $2L$ more weighings.

Case 3: $A < B$ if $B' = C'$, then both of the B' and C', contains a defective coin, if $B' > C'$ then the defective is in $S - (A \cup C')$, and we are from side to side by induction. The case $B' < C'$, will not occur.

To prove our final statement, we sharpen the implications (i) and (ii). Solving for n we see that

i″) ${}^{n}C_2 > 3^{2L}$ implies $n(n-1)/2 > 3^{2L}$, implies $(n^2-n-2.3^{2L}) > 0$, implies $n > ((1 + \sqrt{(1^2+8.3^{2L})}/2)$, implies $n > (½ + \sqrt{(¼+2.3^{2L})}$, for $n > (½ + \sqrt{(¼+2.3^{2L})}$, and therefore for $n \geq \lceil 3^L\sqrt{2} \rceil + 1$ and likewise.

ii″) ${}^{n}C_2 > 3^{2L+1}$ for $n > (½ + \sqrt{(¼+2.3^{2L}}$) and hence for $n \geq \lceil 3^L\sqrt{6} \rceil + 1$.

From (i′), (i″) and (ii′), (ii″), in that order, we infer that the information-theoretical bound is attained by $L^2(n)$ for all n lying in intervals of the form $[\lceil 3^L\sqrt{2} \rceil + 1, 3^L\sqrt{2}]$ and $[\lceil 3^L \sqrt{2} \rceil + 1, 3^{L+1}]$, $L \geq 1$.

Example 2.1: Let $S = \{1, 2, ..., 13\}$, in our first examination we weigh $A = \{1, 2, 3, 4\}$ in opposition to $B = \{5, 6, 7, 8\}$. If we do not have equilibrium, then we may presume $A > B$. In our second examination we weigh $\{1, 9, 10\}$ in opposition to $\{2, 11, 12\}$. In the Figure 2.2, we draw a line between elements $\{i, j\}$ if and only if $\{i, j\}$ is a candidate for the defective pair subsequent to these weighings.

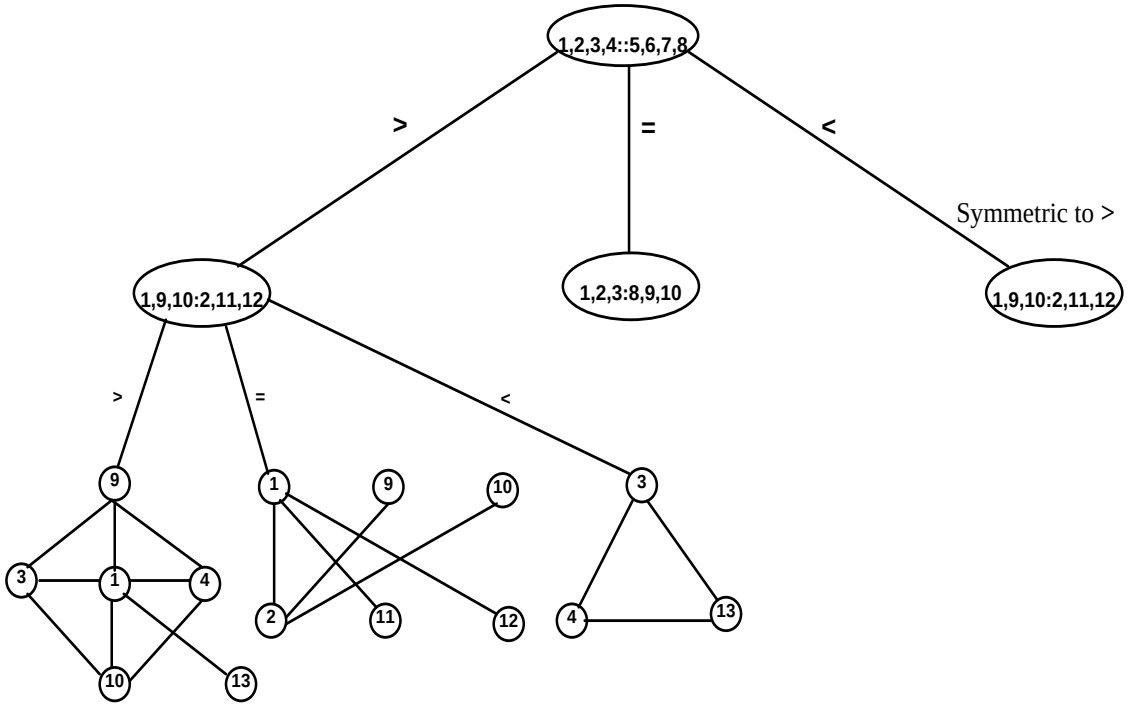


Figure 2.2: The solution of the two counterfeit coins problems among thirteen coins.

If we obtain the answer “>” in the second test, then by weighings 3 in opposition to 4 in the third test we divide the lines Figure 2.4 into three “stars”: By our definition of the lines, the middle of a star must be defective whence the other defective coin is determined with one additional test. The answer “<” in the second test is symmetric to this case, so let us presume we receive as an answer “=” in this case, we test 1 against 3 in the third examination, whence our point-line Figure 2.4 splits into the three configurations of Figure 2.5.

Evidently, these three possibilities can once more be dealt with using one additional test. The investigation of the case where we have equality $A = B$ in our first weighing can without difficulty be done using the same approach. If “=” result after the first test, weigh (1, 2, 3 : 8, 9, 10). The consequential configurations for gather than, balance, and less than $> , = , <$ are, for greater than the subsequent comparison will be considered.

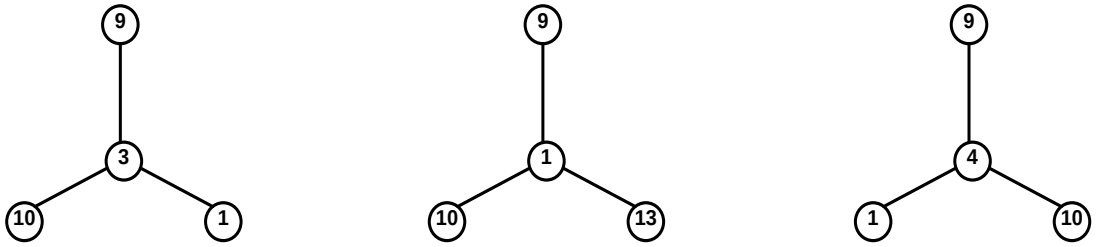


Figure 2.3: Explanation of Figure 2.2 when we get *greater than* answer.

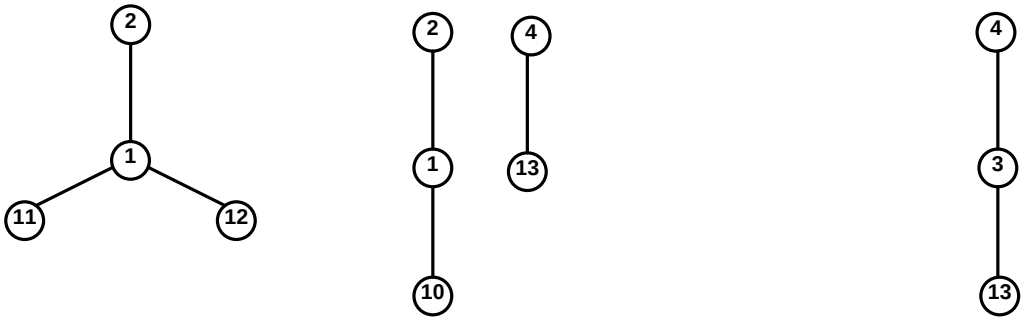


Figure 2.4: Explanation of Figure 2.2 when we get *less than* answer in the second symmetric.

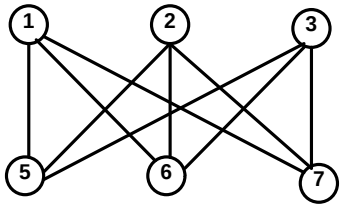


Figure 2.5: Explanation of Figure 2.2 when we get three answers *balance*, *less than*, and *greater than*.

In each case two additional weighings adequate. *Tošić* has considered the *d*-defective coin problem for $d \leq 5$. His outcome show that the best possible $L^d(n)$ differs once more only to some extent from the information theoretic bound $\lceil log_3{}^nC_d \rceil$ at least for certain series of *n*. Part of the difficulties lies in the number-theoretic problem of identifying the right exponent *L* for which $3^{L-1} < {}^nC_d \leq 3^L$. In addition, it is not even clear whether $L^d(n)$ is essentially an growing function of *n*, as for $d = 1$or $d = 2$. Pyber [48] has shown that $L^d(n) \leq \lceil \log_3{}^nC_d \rceil + 15d$, and it might well be true that $L^d(n) \leq \lceil \log_3{}^nC_d \rceil$ for nearly everyone or all pairs (*n*, *d*).

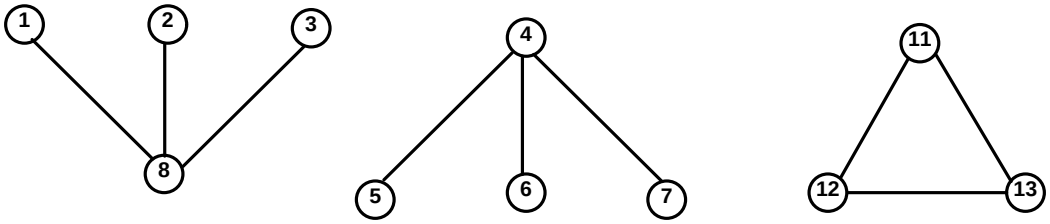


Figure 2.6: Explanation of Figure 2.2 when we get *direct equal* answer following comparison will be considered.

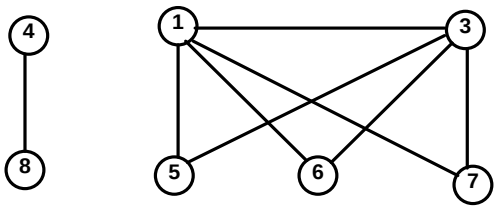


Figure 2.7: Explanation of Figure 2.2 when we get *less than* answer following comparison will be considered.

2.3.1 The Model

Let us start with a small number of examples, thereby gaining some first insights into common description and attribute differences of a variety of search problems.

Example 2.2: Weighings maybe the oldest and positively most extensively known problem concerns the search for a fake coin. In a set of n coins there is exactly one fake coin, say, heavier than the rest. We want to distinguish the counterfeit coin with as few weighings as likely using an equal arms balance. A variation is theoretical there are some good coins whose common weight is recognized and some heavier coins whose common weight is also known. Determine the counterfeit coins by using a spring scale.

Example 2.3: Group tests it is feared that a infectious decease has infected parts of a given population. To recognize the sick, a blood sample is drawn from every single person. Instead of evaluating all samples (which may be too costly), some samples are poured together, and the mixture is then analysed. In this way, we study whether the group tested contains at least one sick member or is overall healthy. By sensibly choosing the groups, the sick subpopulation is isolated.

Example 2.4: Sorting we are given a well-organized first of items (say records or words, ordered alphabetically), and an additional item. The task is to bring the new item into its proper place using as few comparisons as possible.

Example 2.5: Twenty questions is a game known in variants all over the world [23]. Somebody leaves the room. The other players agree on a certain subject (or a person or almost anything). Upon returning the player is required to settle on this subject by asking only yes-no questions, winning if he learns the answer with at most 20 queries.

In all these examples we are given a ground-set containing the unidentified element or elements, and we are faced with the task to recognize these elements by performing certain tests.

Definitions: Let S be a non-empty set, called the search domain, $x^* \in S$ and let F be a family of functions on S, called the test family, we decide a function $f_1 \in F$ and receive as answer

the value $f_1(x^*)$. With this information, we decide once more a function $f_2 \in F$ and get back the significance $f_2(x^*)$, and so on. A successful search algorithm A consists of in the selection of functions $f_1, f_2, f_3, \ldots \in F$ such that the values $f_1(x^*), f_2(x^*), f_3(x^*), \ldots$ decide x^* uniquely. We tacitly suppose that at least one such series always exists. The pair (S, F) is called a search process.

Let us highlight again that the option of the k-th test $f_k \in F$ function will, in general, depend on the principles $f_1(x^*), f_2(x^*), f_3(x^*), \ldots, f_k(x^*)$ previously obtained. Even though we will often use the sequential notation $A = (f_1, f_2, f_3, \ldots)$ search algorithm, we mean the "dynamic" understanding of A in the above sense.

Search processes (S, F) can be classified according to a range of different features. Let us in a few words discuss some of the most important types.

i) The cardinality of S. If S is finite or denumerable, then we speak of discrete processes, or else of continuous processes (e.g., determination of a certain real number in [0, 1]).

ii) An a priori probability allocation p may be given on S, i.e. $x^* \in S$ the unidentified element with probability $p(x^*)$. In Example 2.2, one can visualize that the probability of a certain person or subpopulation being contaminated depends on the location or social status or the like. The case when no probabilities are given may also be treated in this general setting, by assuming standardized distribution (as in Example 2.5).

iii) We may categorize the processes according to the family F of allowable test functions. In Example 2.5, we are only permitted functions which achieve two values and even among these we may use only a small number of, depending on the linear structure of S.

iv) A very important characteristic concerns the nature of the algorithms A. An algorithm A is called sequential if the selection of f_k depends on the values $f_1(x^*)$, $f_2(x^*), f_3(x^*), \ldots, f_k(x^*)$ obtained until then (as in Example 2.4). If the functions f_1, f_2,

f_3, ... are fixed in advance, then A is called predetermined. In some books, [13, 23, 32,] one finds the terms dynamic and static, or adaptive and non-adaptive. Since predetermined algorithms can clearly be regarded as special cases of sequential algorithms, they will, in general, take longer than the best sequential algorithms. On the other hand, if the data necessary for sequential algorithms exceed the available space, predetermined algorithms may well be called for.

v) The final two distinctions anxiety the global nature of our search processes. So far, we have been necessary to identify the unknown element x^* with certainty. For very large problems, however, we may be content with determining x^* "approximately" certainly, say with a probability $\geq 1-\varepsilon$ we then speak of probabilistic algorithms. A different variant is that we stop the algorithm once we recognize that x^* is in a "tiny" subset. In this case, we speak of fairly accurate algorithms. For example, in the real number search we may be fulfilled to decide the x^* up to an error $\delta > 0$, i.e. we stop when we know that x^* is in an interval of length $\leq \delta$.

vi) Finally, we categorize search processes according to their in general aim. The problem at hand may call for minimizing the length of successful algorithms in view of all possibilities for x^* this is called a worst-case problem we may be fascinated in minimizing the average length (given a certain a priori distribution). We then talk of an average- case problem. Once more, the tests may have different costs and necessary to look for algorithms of minimal cost or a combination of time and cost (so-called trade-off problems).

After this general outline, it is time to spell out the exact range of combinatorial search treated in this thesis. With no further mention, we will always make the subsequent assumptions:

Assumption 2.1: The search domain S is finite [23], the cardinality of S will predominantly be denoted by $|S| = n$.

Assumption 2.2: The trial family T is finite. Since each function $f \in F$ attains only finitely numerous values, we may suppose without loss of generality that $f(S) \in \{0, 1, ..., q-1\}$ for all $f \in F$ where $q \geq 2$.

Assumption 2.3: We are only paying attention in algorithms that determine the unidentified element with certainty. The most significant and, at any rate, the best-understood case arises for $q = 2$, i.e. when there are always at most two answers to a test enquiry, as in the game of 20 questions. We then call (S, F) a binary search process. For $q = 2$ a test function $f \in F$ is determined by the set $A = \{x \in S : f(x) = 1\}$ with $f(x^*) = 1$, or 0 according to whether $x^* \in A$ or $x^* \notin A$. Therefore F may be recognized with a family, $\mathfrak{R} \subseteq 2^S$ and every algorithm A with a sequence of sets $A_1, A_2, A_3, ...$ in $\mathfrak{R}$ we will often make use of this correspondence.

2.3.1.1 Search Processes and Tree

Search problems occur in almost all field of human activity [23]. We can consider of an abstract method of searching, like searching for the explanation of a mathematical problem, the causes of a medical disease, the significance of a cryptical text, the motivations of a historical event, or, more ambitiously, an explanation to the complex mysteries of the universe. In end, philosophers, physicians, archaeologists, historians, in brief, all people vigorous in some knowledge field do nothing but searching. According to a more provisional acceptation of the word, "search" indicates the search for a physical object which has been missing in the shoe of a huge number of other objects, like the search for a needle in the straw, an article in a large data file, a website on the Internet, the faulty part in a mechanical tool.

Search problems like those occur very frequently in daily life. For that reason, we can assert that even a person who is not predominantly active in science spends most of his or her time searching for something. Now that we are certain that searching is a essential activity of the human being, Aligner [21, 23], the search domain S is finite; the cardinality of S will mainly be denoted by $|S| = n$ test family F is finite. In view of the fact that each function $f \in F$ attains only finitely many values, we may assume without loss of generality that $f(s) \subseteq \{0, 1, q-1\}$ where $q \geq 2$. We are only paying attention in algorithms that decide

the unidentified element certainty. The most significant and, at any rate, the best-understood case arises for $q \geq 2$, i.e. when there are always at most two answers to a test enquiry, we then call (S, F) a binary search process. For $q = 2$ a test function $f \in F$ is determined by the set $A = \{x \in S: f(x) = 1\}$ with $f(x^*) = 1$ or 0 according to whether $x^* \in A$ or $x^* \notin A$. Consequently, F may be recognized with a family $\mathfrak{R} \subseteq 2^S$, and each algorithm A with sequence of sets $A_1, A_2, \ldots$ in $\mathfrak{R}$. We will regularly make use of this correspondence.

Definition: *(S, F) is called an (n, q) – process if $|S| = n \geq 1$ and $f(s) \subseteq \{0, 1, \ldots, q-1\}$, $q \geq 2$ for all $f \in F$. Assume we are given a successful algorithm $A = \{f_1, f_2, \ldots\}$ the values $f_1(x^*), f_2(x^*), \ldots, f_L(x^*)^{(x)}$ settle on $x^* \subset S$ exclusively where $l(x^*)$ will, in general, depend on x^*.*

Definition: *The number $l(x^*)$ is called the (search) length for x^* in A and $L(A) = max_{x^* \in S}$ $l(x^*)$ the length of A. If we have a probability distribution $p = (p(x^*): x^* \in S$ on (i.e. $(p(x^*) \geq 0)$ for all $x \in S$ and $\sum_{x^* \in S}, p(x^*) \geq 1)$ then $L(A;P) = \sum_{x^* \in S} p(x^*)l(x^*)$ is called the average length of A. If a distribution is not unambiguously given, then the uniform distribution on S is supposed, whence in this case $L(A) = 1/n \sum_{x^* \in S} l(x^*)$.*

Let us see what the function is played by computers with respect to search problems, thus reaching closer to the topic of this thesis. While computers are of little or no help in puzzling out the mysteries of the universe, they have a vital role in solving many realistic problems having the subsequent common goal: Searching for an unidentified object in as short time as possible or with as little cost as possible [23]. Computers have a great responsibility for the birth of a theory completely devoted to search problems, although initially, search played only a trivial role with respect to the problem of sorting which furnished the concept for studying combinatorial search problems. Search theory originated in the sixties because of the introduction of high-speed computers. The improvement of algorithm analysis very much contributed to the growth of this area of study. Indeed, frequently the implementation time of an algorithm is highly an acted by the time exhausted is searching. Experimental data demonstrate how the replacement of a well-organized search scheme for a bad one greatly improves the running time of the algorithm.

2.3.1.2 Search Process and Codes

There is one more extremely useful way to signify algorithms A of an (n, q) search process (S, F). Let $A=\{0, 1, ..., q-1\}$ for every $x^* \in S$, we consider the sequence $f_1(x^*)$, $f_2(x^*)$, ..., $f_L(x^*)^{(x)}$ which exclusively determine x^*. This sequence is a vector $w(x^*) \in (A)^{l(x*)}$, which we name the codeword of x^* (with respect to). The code corresponding to A is then the group of all codeword's (x^*), $x^* \in S$. Let us give a common definition of what we denote by a code [23].

Definition: Let A be a set. We set $A^* = \cup^{\infty}_{i=0} A^i$ and call the elements of A^* words over the alphabet A. By convention, A^0 consist of the empty word. A code C over the alphabet A is just subset of A^*. C is called an (n, q) – code if $|A| = q$ and $|C| = n$. If $w \in C$ and $\in A^l$, then $l = l(w)$ is the length of the codeword w, and $L(C) = max_{x \in c}\ l(w)$ is the length of C. Therefore, to some algorithm A of an (n, q) - process (S, F) there corresponds a exclusive (n, q)-code $C = \{w(x^*): x^* \in S\} \subseteq A^*$, called the search code of A [23]. The mapping $x^* \rightarrow w(x^*)$ is a bisection between S and C with $l(x^*) = l(w(x^*))$ for all $x^* \in S$.

2.3.1.3 Search Processes: Worst Case

Once an (n, q) search process (S, F) admits every feasible test function $F =\{0, 1, q-1\}^S$ then we may confine our notice to prefix codes or trees. It is the easiest to regard as trees denoted by $T(n, q)$ the class of (n, q) trees.

Theorem 6: *Let* $n \geq 1$, $q \geq 2$ *then* $L(n, q) = \lceil log_q{}^n \rceil$ *where* $log_q{}^n$ *the logarithm to the basis* q.

Proof: Let $T \in f(n, q)$ with $L(T) = L$. In view of the fact that there are at most q successors to each inner node, we see by induction that for the k-level $S_k(T)$, $|S_k(T)| = q^k$ $(k = 0, 1, ..., L)$ if v is an end node of T with $l(v) < L$, then we change v by an inner node v_0' and put together to v_0' a string of descendent nodes v_1', v_2', ..., $v_{L-i'}$ by means of $l(v_i') = l(v) + i$ where $v_{L-i'}$ is an end-node once more. In this way, we get hold of a new tree $T' \in f(n, q)$ with $E(T') \subseteq S_L(T')$. By our inequality above, we conclude $n = |E(T)| = |E(T')| \leq q^L$ and those $L = \lceil \log_q{}^n \rceil$, since L is an integer [23].

Example 2.5: Consider the weighing problem in Example 2.1. We have 80 coins one is heavier. We require to find the counterfeit coin x^* by weighings with equal arms balance. The search domain S consist of the 80 coins and the check function f has three possible outcomes as the false coin may be on the left-hand or right-hand side or it may be in the outstanding set not measured by f. The theorem tells us that $\lceil \log_3 n \rceil$ weighings are required. Since not all tests are permitted, it is not right away clear whether $\lceil \log_3 n \rceil$ weighing will be adequate.

2.3.1.4 Search Processes: Average Case

We turn to the (L') problem for (n, q) processes (S, F) when $F = \{0, 1, q-1\}^S$. Once more, we consider the class of $T(n, q)$ of (n, q) trees. [13, 23] The subsequent inequality is of central significance.

Proposition

i) *let* $T \in T(n, q)$ *and let* $l_1, l_2, ..., l_n$ *be the length of the leaves* of T, *then* $\sum_{i=1}^{n} q^{-l_1}$ *with equality iff T is regular.*

ii) *suppose* $l_1, l_2, ..., l_n \in N_0$ *satisfy* $\sum_{i=1}^{n} q^{-l_1} \leq 1$ *then there exist an* (n, q) *- tree whose leaves have exactly the lemgth* $l_1, l_2, ..., l_n$.

2.3.1.5 Alphabetic Search Processes

A search process (S, F) with $F \subseteq F_{mon}$ is called alphabetic with respect to the linear order on S. In view of the fact that in alphabetic search processes the function values are significant, it is best to use the code demonstration of algorithms. Let A be an algorithm and let C be the corresponding search code. Assume $i < j$, and let w_i, $w_j \in C$ be the codeword's of x_i, x_j from the monotony of the check function we infer that for the first letters w_i, w_j which are distinct, the one in w_i must be smaller than in the one in w_j this propose the definition [13, 23].

Definition: *Let C be an* (n, q) *prefix code over* $\{0, 1, q-1\}$. *Let* $v = v_1, v_2, ..., v_s$ *and* $w = w_1, w_2, ..., w_t \in C$. *We classify* $v < w \Leftrightarrow v_i < w_i$ *where* i *is the smallest index with* $v_i \neq w_i$. *The*

relation < is obviously a linear order on C, called the lexicographic order. If $S = \{x_1 < x_2 < ... < x_n\}$*, then the search code* $C = \{w_1, w_2, ..., w_n\}$ *is said to be alphabetic with reverence to* $x_1 < x_2 < ... < x_n$ *if* $w_1 < w_2 < ... < w_n$ *holds for the corresponding codeword.*

2.3.1.6 Binary Search Tree

Consider a search process (S, F_{mon}) with $q = 2$. With an example, we are given an ordered list of objects say records or words [13, 23], ordered alphabetically, and an extra item. The task is to bring the new item into its proper place using as small comparison as possible. Of this kind with the unique task being a new element z into its proper place among $y_1 < y_2 < ... < y_n$. Moments though show that this sorting situation prevails for all n.

Given among $y_1 < y_2 < ... < y_n$ and new elements z then the tests $z : y_j$ correspond exactly to the functions $f_j \in F_{mon}$ where $S = \{0, 1, ..., n\}$ is the set of positions with $i \in S$ significance $y_j < z < y_{j+1}$. Thus, the problem of sorting z into its proper place among $y_1 < y_2 < ... < y_n$ corresponds exactly to the search process (S, F_{mon}) with $s = n+1$. This interpretation, as a "sorting in" of z, suggests the subsequent generalization, usually called the data location problem.

Presume, we are given the list $Y = \{y_1 < y_2 < ... < y_n\}$ and a new element z, by comparing z to a range of y_j's we want to find out z appears in the list Y, or if not to determine the correct position where it be supposed to be sorted. In our search domain consist therefore of a pair $X \cup Y$ where $X = \{x_0, x_1, ..., x_n\}$ and $Y = \{y_1 < y_2 < ... < y_n\}$, where y_j means $z = y_j$, and x_j is interpreted as $y_j < z < y_{j+1}$. In view of the fact that we now have three possible outcomes to a test $z < y_j$, $z = y_j$, $z > y_j$ this is a ternary problem.

Proposition: *The worst-case cost for the data location problem is* $DL(n) = \lceil log_2^{(n+1)} \rceil$.

Let us now turn to the more satisfying average case. We are given probabilities $p_0, p_1, ..., p_n$ and $q_1, q_2, ..., q_n$ with $\sum p_i + \sum q_i = 1$ for the outcomes x_i and y_j, respectively [23]. The average-case cost is denoted by $\bar{D}L(n)$. There are two extreme cases when all $q_j = 0$ or when $p_i = 0$. The previous case corresponds exactly to the situation of the final section with.

2.3.1.7 Predetermined Algorithms

Let us get an obvious picture of what predetermined algorithm looks like. Consider the (n, q) process (S, F). A predetermined algorithm A of length, say, L must put down functions $f_1, f_2, f_3, \ldots \in F$. Once and for all, A may consequently be represented in the subsequent matrix form. Connected columns of a matrix M_A with the elements $x_1, x_2, \ldots, x_n$ of S and rows with the functions $f_1, f_2, \ldots, f_L$ writing, $f_i(x_j)$ in the box indexed by (i, j) M_A are those an LX_n matrix over $\{0, 1, \ldots, q-1\}$. That A is a successful algorithm is in fact reflected by the fact that all columns of M_A are distinct.

The significant fact is that the contrary is also true. If M is an n-columned matrix whose rows are permissible and whose columns are pair-wise distinct, then M corresponds to a successful predetermined algorithm for the search process (S, F) without a doubt, if we let the functions $f_1, f_2, \ldots, f_L$ corresponds to the row, then no matter what the answer to the test f_i is, at the end the unidentified element x^* will be exclusively determined by the pair wise distinctness of the columns of M. Let us say that the n- columned matrix for the process (S, F) if the columns of M are pair-wise distinct and if all rows are sequence $f(x_1)$, $f(x_2)$, ..., $f(x_n)$ for some $f \in F$, $S = \{x_0, x_1, \ldots, x_n\}$. Our ($L$) problem for predetermined algorithm reduces thus to finding search matrices for (S, F) with a minimal number L of rows.

Example 4: When $F = \{0, 1, q-1\}^S$, then the only form on the search matrices M is that the columns be different. In view of the fact that there are q^L distinct vectors of length $\lceil \log_q n \rceil$ over $\{0, 1, \ldots, q-1\}$ it follows that any such matrix must satisfy $n \leq q^L$, i.e. $L \geq \lceil \log_q n \rceil$. On the other hand, by taking any $n \leq q \lceil \log_q n \rceil$ distinct columns of length $\lceil \log_q n \rceil$ we get a suitable search matrix, and thus that the (L)-problem has the same answer for sequential and predetermined algorithms when all tests are admitted [23]. As an example, consider $n = 12$, $q = 2$. Any 0, 1-matrix with 12 distinct columns and $4 = \lceil \log_2 12 \rceil$ rows will characterize a predetermined algorithm, e.g., the following matrix:

0	0	0	0	1	1	1	1	1	1	1	0
0	0	0	1	0	1	0	0	1	1	0	1
0	0	1	0	0	0	1	0	1	0	1	1
0	1	0	0	0	0	0	1	0	1	1	1

Let us indicate by $L_{pre}(S, F)$ the worst-case cost of the search process (S, F) when just predetermined algorithms are accessible. Let $L_{pre}(n, q)$ be the worst-case cost of an (n, q)-process with no restrictions on the functions.

2.3.1.8 Binary Search Processes

The most significant and best-studied case of search processes arises for $q = 2$. The condition $(q-1|n-1)$ is irrelevantly satisfied and the concept of a process being normal or irreducible match, may identify a test function $f: S \rightarrow \{0, 1\}$ with the set $\Re = \{x \in S: f(x) = 1\}$. We will therefore frequently write $(S, \Re)$ to indicate binary processes. Every algorithm A corresponds to a sequence $= \{A_1, A_2, ..., A_n\}$, where each elements $x^* \in S$ is exclusively determined by containment. The resultant search codes consist of 0, 1 words and will consequently be called binary codes. Analogously, the decision trees will be called binary tree [23].

2.3.2 General Sequential Algorithms

Group testing takes benefit of that by identifying groups containing no defective, thus identifying all objects in such a group in one stroke. However, the determination, of how big a group should be, is a delicate question [17]. On the one hand, one would like to identify a large pure group such that many items are recognized in one test; this argues for testing a large group. Nonetheless, on the other hand, if the outcome is positive, then a smaller group contains more information; this argues for testing a small group. Keeping a balance between these two contradictory goals is what most algorithms strive for.

2.3.2.1 Binary Tree Representation of a Sequential Algorithm

A binary tree can be inductively defined as a node, called the root, with its two disjoint binary trees [17], called the left and right subtree from the root, either both empty and both non-empty. Nodes taking place in the two subtrees are called children of the root, and all the nodes that have a given node as a child are ancestors (the immediate ancestor is called

a parent*).* Two children of the same parent are siblings. Nodes which have no children are called leaves, and all other nodes are called internal nodes. The path length of a node is the number of that node's ancestors. A node is also said at level 1 if its path length is (l–1). The depth of a binary tree is the maximal level over all levels. Let S denote the sample space. Then a group testing algorithm T for S can be represented by a binary tree, also denoted by T, by the following rules:

(i) Each internal node u is related with a test $t(u)$; its two links linked with the two outcomes of $t(u)$ (we will always designate the negative outcome by the left link). The test history $H(u)$ of a node u is the set of tests and outcomes connected with the nodes and links on the path of u.

(ii) Each node u is also connected with an event $S(u)$ which consists of all the members of S consistent with $H(u)$. $| v(u) | \leq 1$ for each leaf v.

2.3.2.2 The Structure of Group Testing

The information lower bound is frequently not attainable. For example, consider a set of six items containing precisely two defects. Then $\lceil \log |S| \rceil = \lceil \log {}^6C_2 \rceil = 4$. If a subset of one article is tested, the split is 10 (negative) and 5 (positive); it is 6 and 9 for a subset of two, 3 and 12 for a subset of three, and 1 and 14 for a subset of four. At least four more tests are required. The reason that information lower bound cannot be achieved in general for group testing is that the split of $S(u)$ at an internal node u is not random but must be attainable by a group test. Consequently, it is of significance to study which type of splitting are allowed in group testing.

The rest of this section reports work done by Hwang, Lin and Mallows [55, 56]. While a group testing algorithm surely performs the tests in the order initial from the root of the tree proceeding to the leaves, the analysis is often more suitable. If started from the leaves (that is the way the Huffman tree, a minimum weighted binary tree, is constructed). Thus, instead of asking what splits are permitted, the question becomes: For two child nodes x, y of u, what types of $S(x)$ and $S(y)$ are permitted to combine into $S(u)$. Let N indicate a set of n items, D the defective set and $S_0 = \{D_1, D_2, ..., D_k\}$ the preliminary

sample space. Without loss of generality, presume for any item $\cup_{i=1}^{k} D_i = N$, for any item not in $\cup_{i=1}^{k}$, D_i can right away be recognized as good and deleted from N. A subset S_i of S_0 is said to be possible if there exists a group testing tree $T(A)$ for S_0 and a node u of T such that $S(u) = S_i$. Let it $\pi = \{ S_1, S_2, ..., S_m\}$ be a partition of S_0, i.e. $S_i \cap S_j = \phi$ for all $i \neq j$ and $\cup_{i=1}^{k} S_i = S_0$.

The separation π is said to be realizable if there exists a group testing tree T for S_0 and a set of m nodes $u_1, u_2, ..., u_m$ of T such that $S(u_i) = S_i$. For $1 \leq i \leq m$, characterize $||S|| = \cup_{Di \in S} D_i$. $||S'|| = N / ||S||$. The complement of $||S||$ and $S'' = \{A: A \subseteq ||S||, A \supseteq$ some $D_i\}$ the closure of S in addition, in a separation $\pi = \{S_1, S_2, ..., S_m\}$ of S_0, S_i, and S_j are said to be partition if there exist and $I \subseteq N$, $I \subseteq N$ such that $I \cap D = \phi$ for all $D \in S_i$ and $I \bigcap D = \emptyset$ for all $D \in S_j$ given π define a directed graph G_π by taking S_i as a node, and the directed edge from S_i to S_j ($S_i \rightarrow S_j$) $i \neq j$, if and only if there exist $A_i \in S''$, $A_j \in S''$ such that $A_i \in A_j$ if there exist $D \in S_i$ such that $D \subseteq ||S_j||$.

2.3.2.3 Li's *s*-Stage Algorithms

Li [61] extended a 2-stage algorithm of Dorfman [62] (for Probabilistic Group Testing (PGT)) to s stages**.** At stage 1 the n items are randomly divided into g_1 groups of k_1, (some possibly k_1–1) items. Each of these groups is tested and items in pure groups are identified as good and separated. Items in contaminated groups are pooled together and randomly re-divided into g_2 groups of k_2 (some possibly k_2–1) items, thus, entering stage 2. In general, at stage i, $2 < i < s$, items from the contaminated groups of stage i–1 are pooled and arbitrarily divided into g_i groups of k_i (some possibly k_i–1) items, and a test is performed on each such group k_s, is set to be 1, thus, every item is identified at stages.

Let t_s, denote the number of tests required by Li's s-stage algorithm. Note that s = 1 corresponds to the individual testing algorithm, i.e. testing the items one by one. Thus t_1 = next consider $s = 2$. For easier examination, assume that n is separable by k_1. Then $t_2 = g_1 + g_2 \leq n/k_1 + dk_1$ ignoring the constraint that k_1 is an integer, the upper bound is minimized by setting $k_1 = (n/d)^{½}$ (using straightforward calculus). This gives $g_1 = (nd)^{½}$ and now consider the general s case, where $t_2 \leq 2(nd)^{½}$.

$$t_s = \sum_{i=1}^{s} g_i \leq n/k_1 + dk_1/k_2 + \ldots + dk_{s-2}/k_{s-1}$$

Again, ignoring the integer constraints, then the upper bound is minimized by

$$k_1 = (n/d)^{s-1/s},\ 1 \leq i \leq s-1$$

This gives

$$g_i \leq n(n/d)^{1/s}$$

and

$$t_s \leq sd(n/d)^{1/s}$$

The first derivatives of the upper bound with respect to a continuous s is

$$d(n/d)^{1/s}(1-s\ \ln(n/d)/s^2)$$

which is a unique root $s = \ln(n/d)$. It is simple to verify that $s = \ln(n/d)$ is the unique maximum of the upper bound. Therefore,

$$t_s \leq sd(n/d)^{1/s} \leq ed\ \ln(n/d) = e/\log_e(d\ \log(n/d)),$$

where $e \cong 2.718$, since $sd(n/d)^{1/s}$ is not concave in s, one cannot conclude that the integer s, which maximizes the function is either $\lfloor \ln\ n/d \rfloor$ or $\lceil \ln\ n/d \rceil$. Li gave numerical solutions for such s for given values of n/d.

To perform the algorithm, one needs to work out the optimal s and k_i, for $i = 1, s$. Each k_i can be computed in constant time. Approximating the optimal s by the ceiling or floor function of $\log(n/d)$, then Li's s-stage algorithm runs in time $O(\log(n/d))$.

Li's s-stage algorithm can be simply adapted to be a parallel algorithm. Define $n' = \lceil n/d \rceil$. Apply Li's algorithm to the (d, n') problem except that the g_i; groups at stage i, where $i = 1, s$ are partitioned into $\lceil g_i/p \rceil$ classes and groups in the same class are tested in the same round. Then the number of rounds with p processors is about the same as $M_{Li}(d, n')$.

We now show the astonishing result that, Li's *s*-stage algorithm can be implemented as a 3-bin algorithm. The three bins are labelled "queue", "good item" and "new queue". At the beginning of stage *i*, items which have been recognized as good or in the good-item bin, and all other items are in the queue bin. Items in the queue bin are tested in groups of size k_i (some possibly $k_i - 1$) according to Li's *s*-stage algorithm. Items in groups tested negative are thrown into the good-item bin, and items in groups tested positive are thrown into the new-queue bin. At the end of stage *i*, the queue bin is emptied and change labels with the new-queue bin to start the next stage. Of course, at stage *s*, each group is of size one and the items thrown into the new-queue bin are all defective.

2.3.2.4 Hwang's Generalized Binary Splitting Algorithm

It is well-known that one can recognize a defective from a contaminated group of *n* items in $\lceil log\ n \rceil$ tests through binary splitting. Namely, partition the *n* items into two disjoint groups such that neither group has a size exceeding $2^{\lceil \log n \rceil - 1}$. Test one such group, the outcome indicates either the tested group or the other one is contaminated. Apply binary splitting of the new contaminated group. A recursive argument shows that in $\lceil \log n \rceil$ tests a contaminated group of size 1 can be obtained, i.e. a defective is recognized. A special binary splitting technique is the halving method which partitions the two groups as evenly as likely. By applying binary splitting *d* times, one can identify the *d* defects in the (*d*, *n*) problem in at most $d \lceil \log n \rceil$ tests. Hwang [60] suggested a way to synchronize the *d* applications of binary splitting such that the total number of tests can be reduced. The idea is, approximately, that there exists on average a defective in each *n*/*d* item. Instead of catching a contaminated group of size about half of the unique group, which is the spirit of binary splitting, one could expect to catch a much smaller contaminated group and thus to identify a defective therein in a fewer number of tests. The following is his generalized binary splitting algorithm *G*:

Algorithm *G*:

Step 1: If $n \leq 2d - 2$ test the *n* items separately. If $n \geq 2d - 1$, set $l = n - d + 1$, define $\alpha = \lfloor \ln n/d \rfloor$.

Step 2: If $n > 2d - 2$, test a group of size 2^{α}. If the result is negative, the 2^{α} items in the group are recognized as good. Set $n = n - 2^{\alpha}$ and go to Step 1. If the result is positive, use binary splitting to recognize one defective and an unspecified number, say x, of good items. Set $n = n - 1 - x$ and $d = d - 1$, go to Step 1.

2.3.2.5 The Nested Class

Sobel and Groll [59] introduced a class of simple and efficient algorithms for Probabilistic Group Testing (PGT), called the nested class. A nested algorithm can be described by the following rules:

1. There is no ceiling on the test group in anticipation of a group is tested to be infected. Mark this group the current infected group and denote it by C.

2. The next test should be in a group; say G which is a proper subset of C if G is infected and then G replace C as the current infected group. Or else, items in G are classified as good and C / G replaces C as the current infected group.

3. If the current infected group is of size one, identify the item in the group as defective. Test any group of unidentified items, if any.

Note that the generalized binary splitting algorithm is in the nested class. A simple set of recursive equations can now explain the number of tests necessary by a minima nested algorithm. Let $H(d, n)$ indicate that number and let $F(m; d, n)$ indicate the same except for the existence of a current infected group of size m.

$$H(d, n) = min_{1<m<n}\ max\{H(d, n-m), F(m; d, n)\}$$

$$F(m; d, n)\} = min_{1<m<n}\ max\{F(m-k; n-k), F(k; d, n)\}$$

With boundary conditions,

$$H(d, d) = H(0, n) = 0$$

$$F(1; d, n) = H(d - 1, n - 1)$$

Since the recursive equations have three parameters (d; n, m), and each equation compares O(m) values, where the range of m is n, a brute force solution, requires O(n^3d) time. However, a careful analysis can significantly cut down the time complexity.

Define a line algorithm as one which orders the unclassified items linearly and always tests a group at the top of the order. It is easily verified that a line algorithm identifies the items in order except —

1. A good item may be identified together with a sequence of items up to the first defective after it.

2. When only one unidentified defective is left, then the order of recognition is from both ends towards the defective (this is because once an infected group is recognized, all other items can be deduced to be good).

2.3.2.6 (d, n)-Algorithm and Merging Algorithm

A problem apparently unrelated to the group testing problem is the merging problem which has been studied comprehensively in the computer science literature [13], for example). Consider two linearly ordered sets

$$A_d = \{a_1 < a_2 < ... < a_d\},$$

$$B_g = \{b_1 < b_2 < ... < b_g\}.$$

Assuming a_i and b_j are all distinct, the problem is to merge A_d with B_g into a single linearly ordered set $U_{d+g} = \{u_1 < u_2 < ... < u_{d+g}\}$, by means of a sequence of pair-wise comparisons between elements of A_d and elements of B_g. Hwang [63] compared the two problems and recognized some relationships between them, whereby algorithms for solving one problem may be transformed to similar algorithms for solving the other problem. He showed that a class of merging algorithms well studied in the merging literature could be transformed to a class of corresponding (d, n)-algorithms. The problem of merging A_d with B_g can also be viewed as the problem of determining which elements in U_{d+g} are elements of A_d. Interpreting elements of A_d as defectives and elements of B_g as

high-quality items, then the sample space of A_d in U_{d+g} is exactly the same as that of the d defectives in $I = \{I_1, I_2, ..., I_n\}$, where $n = d + g$.

Furthermore, both the merging problem and the (d, n)-problem are to determine the one sample point from the sample space $S(d, n)$; clearly, a merging algorithm can also be represented by a binary tree:

(i) A series of comparisons is represented by a directed path from the root to a node. Each internal node is linked with a moderately while the two outgoing links on the node denote the two possible outcomes.

(ii) Each node is also linked with the event whose sample points are consistent with the outcomes of the series of comparisons made along the path preceding the node.

A merging algorithm and a (d, n)-algorithm are said to be jointly convertible if they can be represented by the same rooted binary tree. Though a comparison and a test serve the same purpose of partitioning the sample space into two smaller subspaces, the sets of possible partitions induced by each of them are quite different. In general, a comparison of a_i versus b_j answers the question. Are there at least i of the $i + j - 1$ smallest elements of U_{d+g} elements of A_d? On the other hand, a group test on $X \in I$ answers the question: Is there at least one defective in X? However, if $i = 1$ or d, then the comparison a_i versus b_j can be seen to correspond to a group test on $I = \{I_1, I_2, ..., I_j\}$, or $X = (I_{j+d}, I_{j+1}, ..., I_{g+d})$ respectively in the sense that there is a one-to-one correspondence between each of the two possible outcomes in the two problems such that the resulting situations again have isomorphic sample spaces.

2.3.2.7 Number and Group Testing Algorithm

One attractive problem is to count the number of group testing algorithms for S. This is the total number of binary trees with $|S|$ leaves (labelled by members of S) which satisfy the group testing arrangement. While this problem remains open, Moon and Sobel [57] counted for a class of algorithms when the sample space S is the power set of n items. Call a group pure if it contains no defects, and infected, otherwise. A group testing algorithm is

nested if whenever an infected group is known, the next group to be tested must be a proper subset of the infected group.

Lemma 3: *let U be the set of unclassified items and supposes that $C \in U$ is tested to be contaminated. Furthermore, suppose $C' \in C$ is then tested to be contaminated. Then items in C / C' can be mixed with items in U / C without losing any information.*

Proof: In view of the fact that C' being infected implies C being infected, the sample space, given both C and C' being infected is the same as only C', is being infected. But under the latter case, items in C / C' and U / C are not noteworthy.

Thus, under a nested algorithm, at any stage, the set of unspecified items is categorized by two parameters m and n, where $m > 0$ is the number of items in an infected group and n is the total number of unspecified items. Let $f(m, n)$ denotes the number of nested algorithms when the sample space is categorized by such m and n. By using the "nested" property, Moon and Sobel [57] obtained:

$$f(0, 0) = 1,$$

$$f(0, 0) = \sum_{k=1}^{n} f(0, n-k)\, f(k, n) \text{ for } n \geq 1,$$

$$f(1, 0) = f(0, n-1) \text{ for } n \geq 1,$$

$$f(m, n) = \sum_{k=1}^{m-1} f(m-k, n-k)\, f(k, n) \text{ for } n \geq m \geq 1,$$

where k is the size of the group to be tested. Recall that the Catalan numbers $C_k = 1/k\,({}^{2k-2}C_{k-1})$ satisfy the recurrence relation $C_k = \sum_{i=1}^{k-1} C_i C_{k-i}$.

2.3.2.8 Two Disjoint Sets Each Containing Exactly One Defective

Chang and Hwang [54] considered the Combinatorial Group Testing (CGT) problem of identifying two defectives in $A = \{A_1, A_2, ..., A_m\}$ and $B = \{B_1, B_2, \ldots, B_n\}$ where A and B are disjoint, and each contains precisely one defective. At first, it seems that one cannot do better than working on the two disjoint sets independently. The following example shows that perception is not always dependable for this problem.

Example 2.6: Let $A = \{A_1, A_2, A_3\}$ and $B = \{B_1, B_2, B_3, B_4, B_5\}$. If one identifies the defects in A and B independently, subsequently it takes $\lceil \log 3 \rceil + \lceil \log 5 \rceil = 2 + 3 = 5$ tests. On the other hand, the following algorithm shows that the two defects can be recognized in 4 tests.

Step 1: Test $\{A_1, A_2,\}$, if the result is negative, then A has two items and B have four items left. Binary splitting will recognize the two defectives in log 2 + log 4 = 3 more tests. Therefore, it suffices to consider the optimistic result.

Step 2: Test B_1 If the outcome is negative, and then A_1 must be defective. The defective in the four remaining items of B can be identified in 2 more tests. If the outcome is positive, then the defective in the three items of A can be identified in 2 more tests.

Note that there are $3\times5 = 15$ samples $\{A_i, B_j\}$ since $\lceil \log 15 \rceil = 4$ one certainly cannot do better than 4 tests. In general, the sample space is $A\times B$ which will also be denoted by $m\times n$ if $| A | = m$ and $| B | = n$. Does there always exist an algorithm to identify the two defectives in $A\times B$ in $\lceil \log mn \rceil$ tests? Chang and Hwang [55] answered in the affirmative. A Sample space is said to be A-distinct if no two samples in it share the same A-item A_j. Suppose S is a sample space with $| S | = 2^r + 2^{r-1} + \ldots + 2^{r-p} + q$, where $2^{r-p-1} \geq q >$ an algorithm T for S is called A-sharp if it satisfies the following conditions:

(i) T solves S, in r+1 tests.

(ii) Let v_i be the ith node on the all-positive path of T_i the path where every outcome is positive. Let $v'(i)$ be the child-node of v_i with the negative outcome. Then $|Sv'(i)| = 2^{r-i}$ for $i = 0, 1, p$.

(iii) $|S(v(p+1)| = q$ and $S(v(p+1)$ is A-distinct $|S| = 2^r$, then the above conditions are replaced by the single condition. (i') T solves S, in r tests.

Lemma 4: *There exists an A-sharp algorithm for any A-distinct sample space.*

Proof: Ignore the B-items in the A-distinct sample space. Since the A-items are all distinct, there is no restriction on the partitions. It is easily verified that there exists a binary splitting

algorithm which is *A*-sharp. For m fixed, define n_k to be the largest integer such that $mn_k \leq 2^k$. Clearly, there exists a k for which $n_k = 1$.

2.3.2.9 The Two Defective Cases

Let $n_t(d)$ denote the largest n such that the (d, n)-problem can be solved in t tests. Since $M(d, n)$ is no decreasing in d for $d < n$ a total solution of $n_t(d)$ is equivalent to a total solution of $M(d, n)$. While $n_t(1) = 2^t$ is easily obtained by binary splitting, the solution of $n_t(2)$ is unexpectedly hard and remains open. In this section bounds on $n_t(2)$ are studied. For $t \geq 1$ let i_t denote the integer such that

$$({}^{it}C_2) < 2^t < ({}^{it+1}C_2)$$

Since no integer i is a solution of ${}^{i}C_2 = 2^t$ for $t \geq 1$, no uncertainty arises from the definition of it. By the information lower bound it is evidently an upper bound of $n_t(2)$ Chang, Hwang and Lin [56] showed that $i_t - 1$ is also an upper bound of $n_t(2)$.

Lemma 5: $i_t = \lfloor 2^{(t+1)/2} - 1/2 \rfloor + 1$.

Proof: It suffices to prove:

$$\binom{\left\lfloor 2^{\frac{t+1}{2}} - \frac{1}{2} \right\rfloor + 1}{2} < 2^t < \binom{\left\lfloor 2^{\frac{t+1}{2}} - \frac{1}{2} \right\rfloor + 2}{2}$$

Since

$$\left\lfloor 2^{\frac{t+1}{2}} - \frac{1}{2} \right\rfloor + 1 < 2^{\frac{t+1}{2}} + \frac{1}{2} < \left\lfloor 2^{\frac{t+1}{2}} - \frac{1}{2} \right\rfloor + 2,$$

$$\binom{\left\lfloor 2^{\frac{t+1}{2}} - \frac{1}{2} \right\rfloor + 1}{2} < \frac{\left(2^{\frac{t+1}{2}} + \frac{1}{2}\right)\left(2^{\frac{t+1}{2}} - \frac{1}{2}\right)}{2} = 2^t - \frac{1}{8} < \binom{\left\lfloor 2^{\frac{t+1}{2}} - \frac{1}{2} \right\rfloor + 2}{2}$$

By noting that no integer i is a solution of $\binom{i}{2} = 2^t$ and it follows that:

$$\binom{\left\lfloor 2^{\frac{t+1}{2}} - \frac{1}{2} \right\rfloor + 1}{2} < 2^t < \binom{\left\lfloor 2^{\frac{t+1}{2}} - \frac{1}{2} \right\rfloor + 2}{2}$$

2.4 Summary

In this chapter of the thesis, we have discussed different techniques for solving counterfeit coins problem. The most fundamental method for solving counterfeit coins problem is search processes and a decision tree and different elimination based techniques used for solving a given counterfeit coins problem. The basic search processes and decision tree method first assigns some definite number of coins in each of the pans, and then it goes on checking whether the pan is balanced, lighter or heavier. If the pans are balanced, then our search domain reduced by two-thirds for single counterfeit coins problem, where the counterfeit coin has the only possibility lighter or heavier. If the counterfeit coin has the possibility both heavier and lighter then problem became little harder because the sample space is increased by twice, where we suspect all the coins has the two possibilities, in this thesis we have discussed all the theoretical aspect of counterfeit coins problem, when we can solve the problem or when it is not possible to reach the lower bound of the counterfeit coins problem, for two counterfeit coins problem our main purpose is to reduce the search domain since two counterfeit coins problem is complex, we use predetermined algorithm since this algorithm is as good as sequential algorithm particularly when test function is permitted. We have discussed many properties of the predetermined algorithm. If the pans are balanced for two counterfeit coins problems our search domain reduced is only, we have discussed the algorithm given by Tošić [10] with example. We have discussed the property of the binary tree illustration of a sequential algorithm which can be used in counterfeit finding.

In this thesis, we have disused the Li's *s*-stage algorithms, how we can divide the set of objects to attain the lower bound. In Hawng's generalized binary splitting algorithm, we bring to a close that one can identify a defective from an infected group of *n* items in $\lceil \log n \rceil$ tests through binary splitting. Namely, partition the *n* items into two disjoint groups such that neither group has a size more than $2^{\lceil \log n \rceil - 1}$. In the situation of (*d*, *n*)-algorithm and

merging algorithm *A* problem apparently unrelated to the group testing problem is the merging problem which has been studied at length between elements of A_d and elements of B_g compared the two problems and established some relationships between them whereby algorithms for solving one problem may be converted to similar algorithms for solving the other problem. The problem of merging A_d with B_g can also be viewed as the problem of determining which elements in U_{d+g} are elements of A_d. Interpreting elements of A_d as defectives and elements of B_g as good items, then the sample space of A_d in U_{d+g} is the same as that of the *d* defectives.

Chapter 3

A Generalised Algorithm for Solving *n* Coins Problem

3.1 Overview

In this chapter, we are going to study the new solutions for eight coins problem and algorithms for solving *n* coins problem. This chapter is organized into eight sections. In Section 3.2, we have briefly discussed the eight coins problem and its two new solutions. In Section 3.3, we have discussed algorithms for solving *n* coins problem. In Section 3.4, we have discussed the extension of the algorithm to solve $n \geq 6$ (even) coins problem. In Section 3.5, we have discussed the extension of the algorithm to solve $n \geq 7$ (odd) coins problem. In Section 3.6, we have discussed the applications of the problem, and in Section 3.7, we have shown the experimental results.

3.2 Introduction to Eight Coins Problem

Eight coins problem [1, 3, 25] is a well-known problem in Mathematics as well as in Computer Science. In this problem, eight coins are given. Say A, B, C, D, E, F, G, and H, and we are told that only one is counterfeit (or false), as it has a different weight than each of the others. We want to determine which coin it is, making use of an equal arm balance. At the same time, we want to know using a minimum number of comparisons and determine whether the false coin is heavier or lighter than each of the remaining. The tree in this figure represents a set of decisions by which we can get the solution(s) of our problem. Therefore, it is called a decision tree. We use lower-case *h* or *l* as a suffix to represent the counterfeit (or false) coin as *heavier* or *lighter*, respectively. In the solution of the eight coins problem in the form of a decision tree in Figure 3.1, each internal vertex (other than leaf vertices) represents a comparison between a pair of sets of coins using an equal arm balance. Needless to mention that in this comparison, both the sets contain an equal number of coins. The tree starts with a vertex, where it considers three coins be kept on either side of the equal arm balance. Surely, if we consider all the eight coins at a time

to distribute them into two sets of four coins each to be compared, then it is a useless comparison as one coin out of eight coins is given as counterfeit (or false). Thus, we cannot start with all the coins at a time to be compared to finding out the false coin; rather it leads a redundant comparison. In the decision tree in Figure 3.1, we consider three coins on one side of the equal-arm balance, which is the root of the tree. If the weight-sums are equal, then surely each of these coins is a true coin, and the false coin is either G or H with their possibilities either heavier or lighter. In this situation, as A is a true coin, which we use in comparing separately with G and H after a comparison between G and H themselves. If the weight-sum containing coin A is less than the weight-sum containing coin D (i.e. A+B+C < D+E+F), then we can say that the false coin is either of these six coins only, where either A is lighter, or B is lighter, or C is lighter, or D is heavier, or E is heavier, or F is heavier. At the same time in this situation, both G and H are true coins. The next comparison is highly important in order to make the height of the tree as small as possible; we do three things as follows: (*i*) keep a pair of coins A and E on their own sides, (*ii*) another pair of coins B and D interchange their sides, and (*iii*) the remaining pair of coins C and F are removed from the comparison. Therefore, subsequently, the weight-sum of A and D (i.e. A+D) compare with the weight-sum of B and E (i.e. B+E). Three cases may arise.

Case 1: If weight-sums are equal, then either C or F is a false coin (as these coins are removed from this comparison), where either C is lighter, or F is heavier (following the root of the tree). Therefore, we compare C with A (a true coin). If the weight of A is more than the weight of C, then C is lighter; otherwise, these two coins must have the same weight resulting F as heavier.

Case 2: If A+D < B+E, then certainly either A or E is a false coin, as these coins are kept in their own sides, and the logical relation (following the root of the tree) is unchanged. Here either A is lighter, or E is heavier. Therefore, we compare B (a true coin) with A. If the weight of B is more than the weight of A, then A is lighter; otherwise, these two coins must have the same weight resulting E as heavier.

Case 3: In a similar way, if A+D > B+E, then definitely either B or D is a false coin, as these coins have interchanged their sides and the logical relation (following the root of the

tree) has also changed. Here, either D is heavier, or B is lighter. Therefore, we compare B with A (a true coin). If the weight of A is more than the weight of B, then B is lighter; otherwise, these two coins must have the same weight resulting D as heavier.

Similarly, we may consider the case of weight-sums following the remaining branch of the root of the tree, where A+B+C > D+E+F. Here either A is heavier, or B is heavier, or C is heavier, or D is lighter, or E is lighter, or F is lighter. The remaining part of the subsequent comparisons is done in a similar way as it is explained above and shown in Figure 3.1.

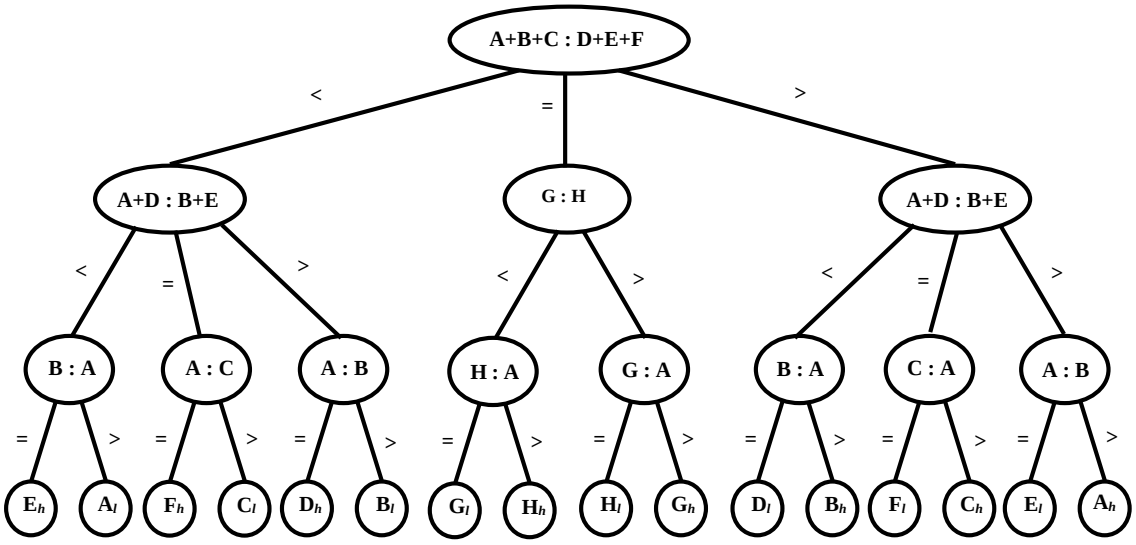


Figure 3.1: The existing solution of the eight coins problem in the form of a decision tree [71].

3.2.1 Two New Solutions of the Eight Coins Problem

In this section, we introduce two new solutions of the eight coins problem and discuss how they are evolving. In the next section, we compare all these solutions based on their relative merits and demerits.

Before developing the newer solutions of the problem under consideration in two subsections of this section, let us show how a false coin, which is either heavier or lighter, can be differentiated in a naive way; this solution is shown in Figure 3.2. In this solution, we compare only a pair of coins to find out the false one. Anyway, this is also a valid

solution with sixteen terminal vertices as eight lighter and eight heavier possibilities of eight different coins of the problem. In contrast to the existing solution in Figure 3.1, this solution takes five (levels of) comparisons in the worst-case, where the solution in Figure 3.1 requires at most three (levels of) comparisons. Therefore, Figure 3.2 is not a desired decision tree in finding out all the possibilities of a coin, either heavier or lighter, of the eight coins problem. In a desired solution of the eight coins problem, we can involve at most three comparisons only (as this is the best-known result and the lower bound on the number of comparisons is also three) [24, 25]. Two such solutions are developed and introduced in the following two subsections.

3.2.1.1 A Solution to the Eight Coins Problem

In the eight coins problem, we are having at most eight coins out of which only one coin in counterfeit (or false), though we do not know whether the false coin is heavier or lighter. Now we like to state two basic observations as follows.

Observation 1: It is meaningless to involve all the coins simultaneously in comparison to finding out a false coin. In that case four of them forms a group and the remaining four forms another group. Finally, one side is always heavier (or lighter) than the other, which is known a priority. This is a redundant comparison and results in giving a very few information; coins are subsequently compared further that increases the height of the tree.

Observation 2: On the other hand, if we consider only pairs of coins to be compared, as done in the case of Figure 3.2, the tree height also increases. It might be a naive way of finding out the false coin, either heavier or lighter, but as the height of the tree increases, it is not the desired solution as it is already mentioned earlier.

Based on the above observations what we conclude that the root of the decision tree must start with comparing coins, keeping either two or three of them in a group. In our first (new) solution we start with consisting only four coins, keeping two of them in a group and comparisons. Suppose A, B, C, and D are these coins where A and B are in a group and C and D in the other, as shown in Figure 3.3. Two cases may arise, either equal or not equal. The case of equality tells that all the four coins A through D are true coins; otherwise one of them must be a false coin, with possibility heavier or lighter. In the next step following

the case of equality, we introduce three coins E, F, and G in a group and compare them with A, B, and C in another group, where each of the coins A, B, and C is a true coin. For A, B, and C (and D) are true coins, so from this comparison, we may find the false coin among E, F, and G (and H) with possibility heavier or lighter. Three cases may arise.

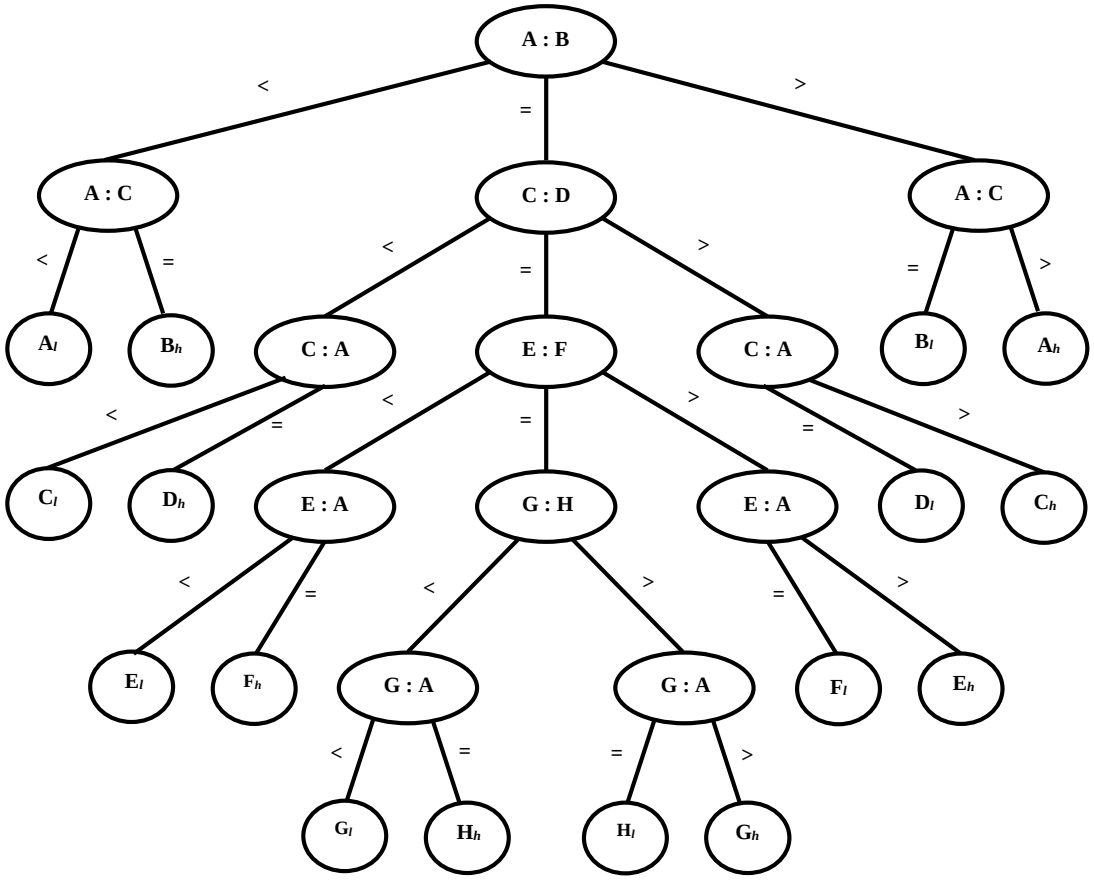


Figure 3.2: A naive solution of the eight coins problem that has been solved with the help of a decision tree.

Case 1: The weight-sum of coins E, F, and G is less than the weight-sum of coins A, B, and C (i.e. E+F+G < A+B+C). This case tells that only either E or F or G is a lighter coin, as A, B, and C are true coins.

Case 2: If the weight-sum of coins E, F, and G is same as the weight-sum of coins A, B, and C (i.e. E+F+G = A+B+C), then surely each of all these coins is a true coin, resulting only H as the false coin, either heavier or lighter.

Case 3: The weight-sum of coins E, F, and G is greater than that of coins A, B, and C (i.e. E+F+G > A+B+C). This case tells that only either E or F or G is a heavier coin, as A, B, and C are all true coins.

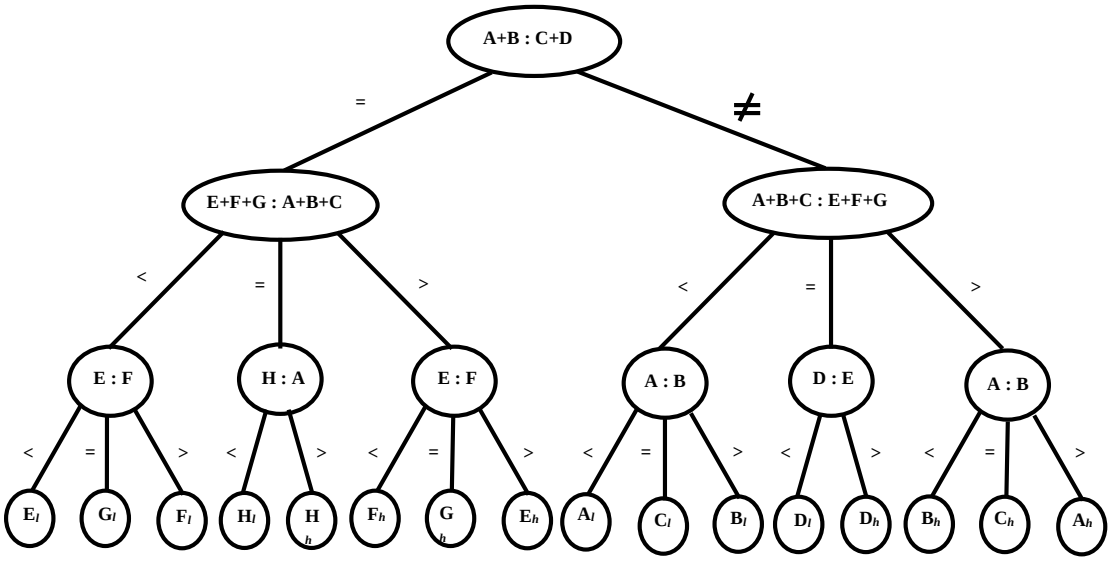


Figure 3.3: The first new solution of the eight coins problem in the form of a decision tree.

Following Case 1 above, to know which of the coins E, F, and G is lighter, we compare any two of them, as shown in Figure 3.3. We compare E and F and conclude the following. If the weight of E is less than the weight of F, then coin E is lighter (as none of them is heavier). If the weight of the E is greater than that of F, then coin F is lighter. Otherwise, if both the coins are having the same weight, then coin G is the false coin with possibility lighter.

Following Case 2 above, we naturally conclude that the coin H is the false coin with possibility either heavier or lighter. This can easily be determined by comparing the coin H with a true coin A, as shown in Figure 3.3. If the weight of H is less than that of A, then coin H is lighter; otherwise, H is heavier.

Following Case 3 above, to identify the false coin among the coins E, F, and G as heavier, we compare any two of them, as shown in Figure 3.3. We compare E and F and conclude the following. If the weight of E is less than the weight of F, then coin F is heavier (as none of these coins is a lighter coin). If the weight of the E is greater than that of F,

then coin E is a heavier coin. Otherwise, if both the coins are of the same weight, then coin G is the false coin with possibility heavier.

Now we consider the other part of the root vertex of the decision tree. Here the weight-sum of A and B is not equal to the weight-sum of C and D (i.e. $A+B \neq C+D$); then we conclude that the false coin belongs to the set of these four coins only. Following this case of inequality, we introduce three coins E, F, and G in a group and compare them with the group of coins A, B, and C (as we did in the case of equality above). Note that in this case E, F, and G (and H) are all true coins.

Since the false coin belongs to the set of coins A, B, C, and D, we want to find out them with their possibilities of either heavier or lighter through the following cases; three cases may arise.

Case 1: The weight-sum of coins A, B, and C is less than the weight-sum of coins E, F, and G (i.e. $A+B+C < E+F+G$). This case tells that only either A or B or C is a lighter coin, as E, F, and G are all true coins.

Case 2: If the weight-sum of coins A, B, and C is same as the weight-sum of coins E, F, and G (i.e. $A+B+C = E+F+G$), then certainly each of all these coins is a true coin, resulting only D as the false coin, either heavier or lighter.

Case 3: The weight-sum of coins A, B, and C is greater than that of coins E, F, and G (i.e. $A+B+C > E+F+G$). This case tells that only either A or B or C is a heavier coin, as E, F, and G are all true coins.

Following Case 1 above (in the case of inequality), to know which of the coins A, B, and C is lighter, we compare any two of them, as shown in Figure 3.3. We compare A and B and conclude the following. If the weight of A is less than the weight of B, then coin A is lighter (as none of them is a heavier coin). If the weight of A is greater than that of B, then coin B is lighter. Otherwise, if both the coins are of the same weight, then surely coin C is the false coin with possibility lighter.

Following Case 2 above, we eventually conclude that the coin D is the false coin with possibility either heavier or lighter. This can easily be determined by comparing the weight of coin D with that of a true coin E, as shown in Figure 3.3. If the weight of D is

less than that of E, then coin D is lighter; otherwise, D is heavier.

Following Case 3 above, to know which of the coins A, B, and C is heavier, we compare any two of them, as shown in Figure 3.3. We compare A and B and conclude the following. If the weight of A is less than that of B, then coin B is heavier (as none of these coins is a lighter coin). If the weight of A is greater than that of B, then coin A is a heavier coin. Otherwise, if both the coins are of the same weight, then coin C is the false coin with its only possibility heavier.

The above development towards a new decision tree for the eight coins problem is shown in Figure 3.3, as the first new solution of the problem in this thesis. The second new solution of the eight coins problem is developed in the next subsection. We critically compare all these solutions in Section 3.7.

3.2.1.2 One Step ahead with the Eight Coins Problem

With this newly developed solution too we start with consisting only four coins, keeping two in a group. Suppose A, B, C, and D are these coins where A and B are in a group, and C and D in the other, as it is in the case of developing a new solution of the eight coins problem in the previous subsection (see Figures 3.3 and 3.4). Here we reach into three possibilities, splitting the case of inequality in either *less* and *greater*, other than the case of equality as it is (in the previous subsection). This method is unlike to the previous method in the way that here we always consider only two coins in a group while making comparisons.

Following the case of equality, we may conclude that all the four coins A, B, C, and D are true coins, and the false coin belongs to the set of remaining four coins E, F, G, and H only, with their possibilities either heavier or lighter. Therefore, this branch of equality follows eight possible leaves with both the possibilities of heavier and lighter for coins E, F, G and H. The other two branches (of *less* and *greater*, following the root of the tree) share the remaining eight possibilities, with four each as stated below.

If the weight-sum of coins A and B is less than the weight-sum of coins C and D (i.e. $A+B < C+D$), then ultimately this branch is supposed to conclude with possibilities either A is lighter, or B is lighter, or C is heavier, or D is heavier. On the contrary, if the

weight-sum of coins A and B is greater than the weight-sum of coins C and D (i.e. A+B > C+D), then we conclude with either of the following possibilities: either A is heavier, or B is heavier, or C is lighter, or D is lighter.

Certainly, following the case of equality of the root of the tree, we introduce coins from the remaining set. Instead of introducing all of them, we apply a trick by introducing any three of them along with a true coin. Say, at this step coins E and F form a group, and coins G and A (a true coin) from the other group. Three possible cases may arise.

Case 1: If E+F = G+A, then surely each of all these coins is a true coin, resulting only H as the false coin, either heavier or lighter. Then we compare H with A and conclude accordingly. If the weight of H is less than that of A, then H is lighter; otherwise, H is a heavier coin.

Case 2: If E+F < G+A, then there are three possibilities: Either E is lighter, or F is lighter, or G is heavier (as A is a true coin). Therefore, subsequently, we compare E and F to conclude the following. If E < F, then E is lighter (as F cannot be heavier). On the other hand, if E > F, then F is lighter (as E cannot be heavier). Otherwise, if the weight of the E is same as the weight of the F (i.e. E = F), then G is heavier.

Case 3: If E+F > G+A, then there are three possibilities: Either E is heavier, or F is heavier, or G is lighter (as A is a true coin). Therefore, subsequently, we compare E and F (same as before) to conclude the following. If E < F, then F is heavier (as E cannot be lighter). On the other hand, if E > F, then E is heavier (as F cannot be lighter). Otherwise, if the weight of the E is same as that of F (i.e. E = F), then G is lighter (as A is identified as a true coin in the earlier step).

Now, we consider the case of A+B < C+D, following the root of the tree. As a result, we may conclude that (*i*) either A is lighter, or B is lighter, or C is heavier, or D is heavier, and (*ii*) each of the remaining four coins, that are E, F, G, and H, is a true coin. Therefore, in a similar way, we apply a trick, where we consider three of the four probable false coins along with a true coin from the remaining (true coins). In addition, we interchange the sides of a pair of coins (say B and C) of the root vertex in making the new groups as follows. Here we keep A and C in a group and B and E (a true coin) in the other.

Three cases may arise:

Case 1: If A+C < B+E, then definitely A is the false coin with possibility lighter. This is because (*i*) E is a true coin, (*ii*) D is no longer there in this comparison, and (*iii*) coins B and C have changed their sides, although the inequality relation is same as before.

Case 2: If A+C = B+E, then surely D is the false coin with possibility heavier. This is because in this case (*i*) all the coins A, B, C, and E are true coins, and (*ii*) D is the only dropped coins in this comparison following the earlier case of inequality.

Case 3: If A+C > B+E, then certainly either C is heavier, or B is lighter, as only this pair of coins have changed their sides and inequality relation has also changed. Therefore, one more step is required to find out the false coin that we achieve by comparing B with E, which is a true coin. Only two cases may arise as follows. If B < E, then B is a lighter coin; otherwise, they must have the same weight to conclude that C is heavier.

The remaining branch of the root of the case of A+B > C+D is treated in a similar way. Here we may conclude that either coin A is heavier, or B is heavier, or C is lighter, or D is lighter. Therefore, similar to the previous step of inequality following the root of the tree, we consider the four coins A, B, C, and E, and group them identically in the same way. Here, coin **E** is a true coin. Three probable cases may arise.

Case 1: If A+C > B+E, then definitely A is the false coin with possibility heavier. This is because (*i*) E is a true coin, (*ii*) D is no longer there in this comparison, and (*iii*) coins B and C have changed their sides, although the inequality relation is same as before.

Case 2: If A+C = B+E, then surely D is the false coin with possibility lighter. This is because in this case (*i*) all the coins A, B, C, and E are true coins, and (*ii*) D is the only dropped coins in this comparison following the earlier case of inequality.

Case 3: If A+C < B+E, then certainly either C is lighter, or B is heavier, as only this pair of coins have changed their sides and inequality relation has also changed. Therefore, one more step is required to find out the false coin that we achieve by comparing B with E, which is a true coin. Only two cases may arise as follows. If B > E, then B is a heavier

coin; otherwise, they must have the same weight to conclude that C is lighter.

This completes the development of the second new solution of the eight coins problem. The solution is shown in the form of a decision tree in Figure 3.4. In the next section, we consider all these solutions, existing, naive, or newly developed in this thesis for their relative comparisons on several parameters.

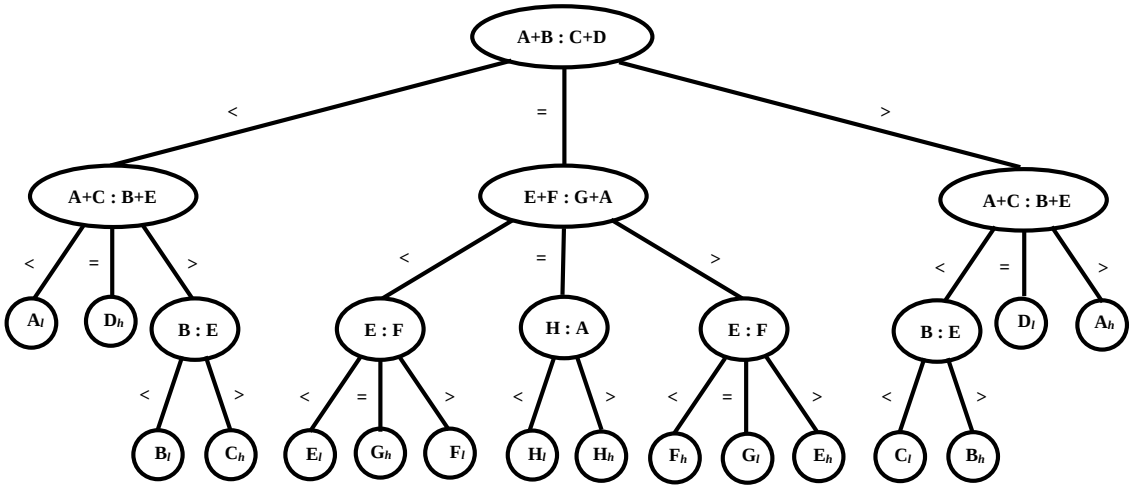


Figure 3.4: The second new solution of the eight coins problem in the form of a decision tree.

3.2.2 Relative Comparisons among the Solutions

In this thesis, we have considered the eight coins problem from its interest from a combinatorial optimization point of view. An existing solution to this problem is already there in the literature [25, 71] (see Figure 3.1). A naive solution of the same is shown in Figure 3.2, and two new solutions are developed in this thesis, as shown in Figures 3.3 and 3.4. In this section, we compare all these solutions based on their maximum number of comparisons towards a solution, their external path length, the total number of comparisons required, consideration of a maximum number of coins involved in comparison, the average height of the decision tree, and so and so forth. All these results are shown in Table 3.1. Now, we analyze the parameters considered in comparing the solutions included in this thesis. The height of the best existing solution of the eight coins problem is four where the desired decisions, either a coin is heavier or lighter, are all available at the fourth level

in the form of leaf vertices of the decision tree. Therefore, at most three comparisons are required starting from the root of the tree. This result of a maximum number of comparisons is equally good for the cases of two newly developed solutions introduced in this thesis, in the worst-case; naturally, the naive solution does not follow it.

The *external path length* is a measure of a tree, which is the sum of the number of branches (or edges) traversed in going from the root *once* to every leaf in the tree [24]. In this respect, the existing solution and the first new solution developed in this thesis are equally good. Needless to mention that in this respect, the naive solution is worse though the second new solution developed in this thesis is better, as four leaf vertices (as decisions) are achieved at the third level.

In terms of total number of comparisons, i.e. the number of internal vertices (other than leaf vertices only), both the existing solution and the naive solution are equally good, but in developing each of the new solutions introduced in this thesis, we have reduced by 25% of the number of comparisons in comparing to that of the earlier two.

Someone may think that the number of coins present in comparison may not be an important issue, though the notion tells that the maximum number of comparisons (or even the height of the tree) may reduce if we involve more coins in a comparison. If only pairs of coins are considered in comparisons, as done in computing the naive solution, the height of the tree increases, though we do know that we should not involve all the coins in comparison, as discussed earlier. Incidentally, our first new solution involves a maximum of six coins in comparison, whereas the second new solution involves only four.

We know that the height of the tree is one more than the maximum number of comparisons involved, starting from the root of the tree, in computing all the desired levels. On the other hand, the average height of the tree is a parameter obtained by computing the average external path length over the desired leaves in a tree. Each solution of the eight coins problem present in this thesis results in obtaining 16 leaf vertices, with eight heavier possibilities and eight lighter possibilities of the given eight coins of the problem. Thus, the average external path length (or average height of the tree) is the average distance from the root of each of the leaf vertices of the tree. In this respect, our first new solution is as good as the existing one, though the second new solution is certainly much better.

Table 3.1: The relative merits and demerits of different solutions of the eight coins problem based on different parameters of making comparisons.

Parameters of making comparisons	Existing solution (in Figure 3.1)	A naive solution (in Figure 3.2)	The first new solution (in Figure 3.3)	The second new solution (in Figure 3.4)
Maximum number of comparisons	3	5	3	3
External path length	48	56	48	44
Total number of comparisons	12	12	9	9
Maximum number of coins in a comparison	6	2	6	4
Height of the tree	4	6	4	4
Average height of the tree	3	3.5	3	2.75

At the end of this discussion, we may conclude that our first new solution is very close to the existing one, though here the total number of comparisons is just nine, whereas that of the existing solution is twelve. Our second new solution outperforms all these; its average height is just 2.75, though the maximum number of coins involved in a comparison case has been just four. Therefore, up to this point of time, we may come to an end (or infer) that the second new solution (that is in Figure 3.4) is the best solution of the eight coins problem.

3.3 Algorithms for Solving *n* Coins Problem

In this section in developing our algorithm, we minimally modify the classical solution, shown in Figure 3.1, to compute a better solution in terms of comparisons as shown in Figure 3.4. Now to minimize the number of comparisons, for the equality case at the root

we do not compare the coins G and H. This is because one out of G and H must be a counterfeit coin, and so they are of unequal weight. We remove this redundant comparison and compare G with A (which is a known correct coin). If they are of unequal weight, it means that G is the faulty coin, and we find out whether it is heavier or lighter from this comparison only. Otherwise, it is understood that H is faulty. In which case we compare H with the known correct coin A and find out whether it is heavier or lighter.

3.3.1 Extension of the Algorithm to Solve $n > 8$ (Powers of 2) Coins Problem

The *n* coins problem is solved using the procedure *n-coin_problem* as described here in this section. This procedure takes, as inputs, the starting index of then coins, and the number of coins. This procedure, then calls the *less_than* function or the *greater_than* function depending upon the weight of the pans. The function *less_than* is called when in the first comparison (i.e. done at the root of the decision tree) the weight of the left pan is less than that of the right pan. Similarly, the function *greater_than* is called when the left pan is heavier than the other. These functions are recursively called within each of the functions until only two suspected coins (of whom one is certainly a counterfeit coin) are left.

As the tree structure goes down, each of these procedures eliminates some coins at each stage, until the number of suspected coins is two [4]. At this stage, these functions take the help of the *check* function. This *check* function helps to identify the counterfeit coin between those two suspected coins, with the help of a known correct coin. When the procedure *n-coin_problem* is being called recursively, and the number of remaining coins is only two or only four, then the procedure calls the *2-coin_problem* or the *4-coin_problem*, respectively (instead of calling the *n-coin_problem* again). Basically, these two procedures act as the base case when we are calling the *n-coin_problem* recursively. Now, let us look at the functioning of the generalized algorithm for solving *n* coins problem, where *n* is a power of 2, which is greater than eight.

Input: An integer number *n* of coins (out of which only one coin is counterfeit, either heavier or lighter)**.**

Output: The index *z* of the counterfeit coin, where *z* is an integer (location of the false coin) and declaring whether it is heavier or lighter**.**

In developing the algorithm that solves the eight coins problem, as shown in Figure 3.4, we now consider each of the procedures as outlined above in isolation and explain how they work. We start with the main procedure, i.e. the *n-coin_problem*.

3.3.2 Procedure *n-coin_problem*

We have been given *n* coins, out of which only one coin is counterfeit (or false). To find the counterfeit coin (and to know whether it is heavier or lighter), we use the decision tree structure. We call the *n-coin_problem* with *start* = 1 and the number of coins = *n*. Here, *start* indicates the starting index of the coins to be considered (or compared), and *n* is the size of the problem. The total number of possible outcomes in cases of *n* coins problem is equal to 2*n*. We first determine the number of coins to be kept on each of the pans of the equal arm balance [4]. This is given by 3*x*, where $x = n/8$. Thus, 6*x* coins are compared initially. We can have three possible cases as follows:

1. Left pan is lighter than the right pan. This case is handled by the *less_than* function. Since the left pan is lighter, it means that any one of the 3*x* coins kept on the left pan is lighter, or any one of the 3*x* coins kept on the right pan is heavier. From this condition, we can reach $2\times3x = 6x$ number of possible outcomes as leaf nodes. The counterfeit coin exists among the coins indexed by *start* and 6*x*. The left pan contains the coins indexed from the *start* through 3*x*, and the right pan contains the coins indexed from 3*x*+1 through 6*x*. The first coin on the left pan is indexed by *S_left* (i.e. at the beginning it is same as a *start*). Similarly, the first coin of the right pan is indexed by *S_right*, (i.e. at the beginning it is the coin same as 3*x*+1).

2. Left pan is heavier than the right pan. This case is handled by the *greater_than* function. Since the right pan is lighter, it means that any one of the 3*x* coins kept on the left pan is heavier, or any one of the 3*x* coins kept on the right pan is lighter. Likewise, from this condition, we can reach $2\times3x = 6x$ number of possible outcomes as leaf nodes. The indexing is the same as in the previous case.

3. Both the pans are equal. Since the pans are equal, it means that the $6x$ coins compared at the root are correct. The counterfeit coin then lies in the $2x$ number of coins yet to be considered. In this case, i.e. in the case of equality, the n coins problem gets reduced to $2x$ coins problem. To solve this $2x$ coins problem, we call the *n-coin_problem* recursively, with the number of coins now equal to $2x$. Since the first $6x$ coins are correct, now $start = 6x+1$ and $n = 2x$. From this case of equality, we can reach $2 \times 2x = 4x$ number of possible outcomes as leaf nodes.

3.3.3 Procedure *less_than* Function

We retain the first x coins in the left pan (starting from *S_left*) and interchange the next x number of coins with the first x coins in the right pan (starting from *S_right*) and retain the next x coins on the right pan as it is. Thus, in this comparison, the last x coins of both the pans are not considered. Rather, these coins are removed from the comparison at this stage. Again, in this case, there are three possible outcomes:

(i) The left pan is lighter: The counterfeit coin exists among the first x number of coins in the left pan or the middle x number of coins in the right pan. Thus, we must update the value of *S_right*. It now points to the first coin of the middle x number of coins of the right pan. The value of *S_left* remains as it is.

(ii) The left pan is heavier: The counterfeit coin exists among the middle x number of coins in the left pan or the first x number of coins in the right pan. Thus, we must update the value of *S_left*. It now points to the first coin of the middle x number of coins of the left pan. The value of *S_right* remains as it is.

(iii) Both pans are equal: The counterfeit coin exists among the last x number of coins of both the left and the right pan. Thus, the value of both *S_left* and *S_right* needs to be updated. *S_left* now points to the first coin of the last x coins of the left pan, and *S_right* now points to the first coin of the last x coins of the right pan.

Now, the value of x is halved as we move down one level in the decision tree structure. If the value of x is equal to ½, only two coins are left undecided. At this point, we call the *check* function with these two undecided coins and a known correct coin, as it

has been considered for the three coins G, H, and A, where A is a correct coin (see the case of equality in Figure 3.4 following the root of the tree).If the value of *x* is more than ½, the *less_than* function is called recursively with the updated values of *S_left* and *S_right*. Case (iii) does not occur more than once. This is because all the undecided coins (among which the counterfeit coin exists) are compared with the next subsequent levels.

3.3.4 Procedure *greater_than* Function

This procedure is very much the mirror image of the earlier procedure. Here we retain the first *x* coins in the left pan (starting from *S_left*) and interchange the next *x* number of coins with the first *x* coins in the right pan (starting from *S_right*) and retain the next *x* coins of the right pan as it is. Thus, in this comparison, the last *x* coins of both the pans are not considered. Again, in this case, there are three possible outcomes:

(i) The left pan is heavier: The counterfeit coin exists among the first *x* number of coins in the left pan or the middle *x* number of coins in the right pan. Thus, we must update the value of *S_right*. It now points to the first coin of the middle *x* number of coins of the right pan. The value of *S_left* remains as it is.

(ii) The left pan is lighter: The counterfeit coin exists among the middle *x* number of coins in the left pan or the first *x* number of coins in the right pan. Thus, we must update the value of *S_left*. It now points to the first coin of the middle *x* number of coins of the left pan. The value of *S_right* remains as it is.

(iii) Both pans are equal: The counterfeit coin exists among the last *x* number of coins of both the left and the right pan. Thus, the value of both *S_left* and *S_right* needs to be updated. *S_left* now points to the first coin of the last *x* coins of the left pan, and *S_right* now points to the first coin of the last *x* coins of the right pan.

Now, the value of *x* is halved as we move down one level in the decision tree structure. If the value *of x is* equal to ½, only two coins are left undecided. At this point, we call the *check* function with these two undecided coins and a known correct coin. The *check* function takes the three coins as parameters, in the following order– lighter or correct coin (for less than the condition it would be the coin indexed by *S_right*, and for greater

than condition it would be the coin indexed by *S_right*+1), heavier or correct coin (for less than the condition it would be the coin indexed by *S_left*+1, and for greater than condition it would be the coin indexed by *S_left*), and known correct coin. This *check* function then finds out the counterfeit coin by comparing these three coins. If the value of *x* is more than ½, the *greater_than* function is called recursively with the updated values of *S_left* and *S_right*. In a similar way, as in the case of a *less_than* function, here also the third case (i.e. Case (iii) above) does not occur more than once. This is because all the undecided coins (among which the counterfeit coin exists) are compared with the next subsequent levels [5].

3.3.5 Procedure *4-coin_problem*

This procedure solves the *n-coin_problem* when the problem is reduced to $n = 4$, i.e. there are only four undecided coins out of which one is counterfeit. The parameters passed to this function are a *start* (i.e. the index of the coin from which the four undecided coins occur) and the index of a correct coin. It compares the coins indexed by *start* and *start*+1. If they are equal, then it means that these two coins are correct, and the problem reduces to two coins problem. Then the *2-coin_problem* is called with a *start* = *start*+2 and the index of a correct coin. Otherwise, if the weight of the initial two coins is unequal, then the *check* function is called with these two coins to find out the counterfeit coin.

3.3.6 Procedure *2-coin_problem*

This solves the *n-coin_problem* when the problem is reduced to $n = 2$, i.e. there are only two undecided or unchecked coins. The parameters passed to this function are a *start* (i.e. the index of the coin from which two undecided coins occur) and the index of a correct coin. This gives the index of the counterfeit coin and mentions whether it is heavier or lighter.

3.3.7 Procedure *check* Function

The *check* function takes three coins as parameters in the following order– lighter or corrects coin, heavier or correct coin, and a known correct coin. This procedure tells whether the first coin passed is lighter (which may be the lighter coin or a correct coin), or the second coin passed is heavier (which may be the heavier coin or a correct coin), or the

reverse. This is done by comparing the weights of the first and second coin passed with the known correct coin passed.

3.4 Extension of the Algorithm to Solve $n \geq 6$ (Even) Coins Problem

In this version of the algorithm, we first find the smallest possible value y depending upon the given value of n such that $y \geq n$, where y is a power of 2. If $y = n$, this version of the algorithm exactly matches to the steps of the algorithm stated in Section 3.1. Needless to mention that if $n = 8$, then $x = 1$ and the number of coins compared at the root of the decision tree is 6 keeping the first 3 coins on the left pan and the next 3 coins on the right pan. Similarly, for $n = 16$, $x = 2$ and the number of coins compared is equal to 12 keeping the first 6 coins on left pan and the next 6 coins on right pan, for $n = 32$, $x = 4$ and the number of coins compared is equal to 24 keeping the first 12 coins on the left pan and the next 12 coins on the right pan, and so on [4].

On the other hand, if n is not equal to y, rather $n < y$ (where $2^{p-1} < n < 2^p = y$, for some integer value of $p \geq 3$), then the number of coins compared in each pan is $2x$, where $x = y/8$. As, for example, if $n = 10$, then $y = 16$ and $x = 2$, and the total number of coins compared at the root of the decision tree is 8 keeping the first 4 coins on the left pan and the next 4 coins on the right pan, if $n = 22$, then $y = 32$ and $x = 4$, and the total number of coins compared at the root of the tree is 16 keeping the first 8 coins on left pan and the next 8 coins on right pan, and so on.

If the left and the right pans are equal in weight at the first comparison made at the root of the tree, then the counterfeit coin lies among the remaining $n-2^{p-1}$ coins of the n coin problem. For example, if the decision follows the equality branch of the tree for $n = 10$, then the counterfeit coin belongs to the remaining 2 coins, for $n = 22$, the counterfeit coin belongs to the remaining 6 coins, and so on. Then following this equality branch of the decision tree, we either recursively call procedure *n-coin_problem*, or straightway call procedure *4-coin_problem*, or *2-coin_problem*, as the situation arises.

On the other hand, if the left pan is lighter than the right pan, then the counterfeit coin lies among the first $4x$ coins, of which one of the first $2x$ coins can be lighter, or one

of the next 2*x* coins can be heavier. Then it calls the *less_than* procedure to deal with the left sub-tree of the tree.

If the left pan is heavier than the right pan, then the counterfeit coin lies among the first 4*x* coins, of which one of the first 2*x* coins can be heavier, or one of the next 2*x* coins can be lighter. Then it calls the *greater_than* procedure to deal with the right sub-tree of the tree.

In this way, each time we move down towards the leaf of the decision tree the value of *x* is divided by 2. When $x = ½$, there are only 2 coins undecided. One of these 2 coins is the counterfeit one. This time the *check* function finds the desired counterfeit coin and declares whether it is heavier or lighter.

3.5 Extension of the Algorithm to Solve $n \geq 7$ (Odd) Coins Problem

In our algorithm, this is an obvious checking whether the given number *n* of coins is even or odd. If it is even, then it is checked whether it is a power of two. If it is a power of two, as it has been developed in Section 3.2. On the other hand, if the value of *n* is odd, we do some prior computations. In this case, just one additional checking is made as it has been detailed below [4, 5].

We know that if *n* is odd, then we can write $n = 2m+1$, where 2*m* is obviously an even number. Then we divide these 2*m* coins into two groups, each having *m* number of coins; each of the groups of *m* coins is kept on each of the pans of the equal arm balance. If they are equal, then the counterfeit coin must be the last coin, i.e. the *n*th indexed coin. But still, we do not know whether it is heavier or lighter than each of the remaining coins. For that, we compare the last coin (i.e. the *n*th indexed coin) with the first coin, which is a known correct coin. If the last coin is lighter than the correct coin, we conclude that the *n*th coin is lighter; otherwise, the last coin must be heavier than the correct coin, and we conclude that the *n*th coin is heavier.

On the other hand, the equal arm balance must show the case of inequality, where *m* coins in a group are kept on either of the pans, and the counterfeit coin belongs to amongst the first $n-1$ ($= 2m$) coins under consideration, which is a case of finding the

counterfeit coin for a given set of even number of coins. Incidentally, this algorithm has been developed in section 3.2.

3.6 Applications of the Problem

The application of the problem under consideration is not only limited to coins, but to any kind of valued article that is frequently counterfeited, like ornaments, watches, metal (like gold), jewelry, etc. A *counterfeit* is an imitation that is made usually with the intent to deceptively represent its content or true origin. The word *counterfeit* most frequently describes the forgeries of currency or documents, but can also describe software, pharmaceuticals, clothing, and more recently, motorcycles and cars, especially when these results in patent or trademark infringement.

Coin counterfeiting occurs regularly in the antique coin market, but there are various modern forgeries that also make it into general circulation [19, 53]. Counterfeit antique coins are generally made to a very high standard so that they often fool collectors; this is not easy, and many coins still stand out. The importance of solving this problem is more in the theoretical field of computer science and/or mathematics. The very fact that the decision tree structure can be used to solve such problems of large size, by eliminating a part of the solution domain after each step of decision making, is of utmost importance, especially because as our algorithm works for any value of n, it does not matter if the value of n is not known a priori. This is, in fact, a new innovative work, and important in solving problems using the decision tree structure.

3.7 Experimental Results of the Algorithm

The output of our algorithm for $n = 64$, the counterfeit coin is at 62 is shown in Figure 3.5.

3.8 Summary

In this chapter, we have considered the eight coins problem, one that is well-known in the literature. Only one out of eight coins is a false coin, which is either heavier or lighter. The usual objective of this problem is to find the false coin using a minimum number of comparisons, and in the form of a decision tree.

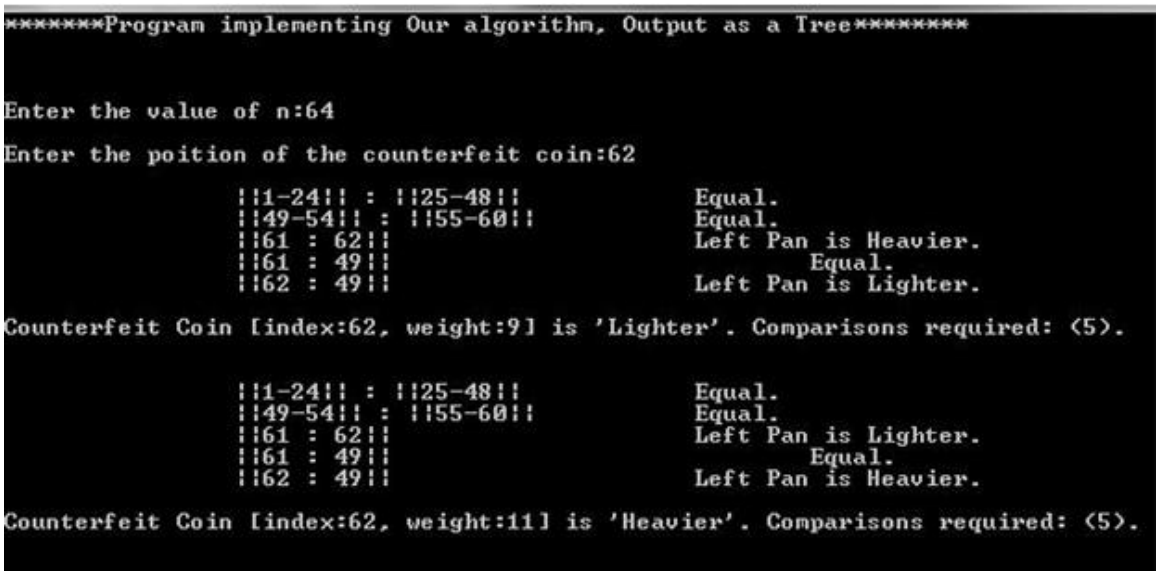


Figure 3.5: Results of our algorithm, which finds the counterfeit coin at 62.

In this chapter, we have developed two new solutions for the eight coins problem, and they are as good as or better than the existing classical solution. Our next target is to find some other equally good or better solutions. Moreover, we would like to consider the twelve coins problem, the sixteen coins problem, and so on, for their probable solutions, and in each only one coin is counterfeit (or false), either heavier or lighter.

At the very beginning, we have made it clear that our objective is to develop an algorithm for the counterfeit coins problem, where the number of coins can be anything, from two to any larger value. We first observe the existing solution for the eight coins problem, and then we modify it slightly to suit our needs. After that, we extract the algorithm behind it and generalize it for all values of *n*, where *n* is an even number. We identify two types of even numbers, ones which are powers of two, and the other is the rest of the numbers, six or more. The algorithm works slightly in different ways for the two types of cases, but the solution is reached in both cases. After developing the algorithm for solving the counterfeit coin problem where the number of coins is even, we widen our algorithm for handling the situation of an odd number of coins problem. This shows that the algorithm developed herein is truly generalized and works for all values of *n* (where *n* is the number of coins). Furthermore, the algorithm terminates after executing for a finite (and predictable) number of steps.

Chapter 4

Algorithms for Two Counterfeit Coins Problem while Both are Heavier or Both are Lighter

4.1 Overview

In this chapter, we are going to study about the two different algorithms for solving two counterfeit coin problems. This chapter is organized into seven sections. In the Section 4.2, we have briefly discussed about the two counterfeit coin problems. In Section 4.3, we have discussed general algorithm for differentiating two counterfeit coin problems. In Section 4.4, we have talked about the algorithms for both the counterfeit coins are equally heavier and equally lighter. In Section 4.5, we have presented algorithms for both the counterfeit coins are equally heavier and equally lighter. In Section 4.6, we have discussed the computational complexity of the algorithms. In Section 4.7, we have presented experimental results. A summary of the chapter is presented in Section 4.8.

4.2 A Brief Study on Two Counterfeit Coins Problem

We have a set of n identical coins and it is known that there are two or more defective coins present. An equal arm balance is provided. When it is a one coin problem, i.e. we know that there is only one counterfeit coin among n cons. We are given that only one coin is defective out of n. It can weigh an equal number of coins on the two arms of the balance. Let us consider that we take k coins in each pan for each weighing, $k < n$ every time [31, 33, 39], where k depends on n for each of the iterations, so we take $2k$ coins from n. If the two groups' balance, the defective coin must be in the remaining $n– 2k$ set of coins. If the two groups do not balance, we know that the false coin is in one of the k groups.

After each weighing, the number of coins to be examined further reduces, but the problem is of the same type. This allows us to apply dynamic programming to this problem. When it is a two coin problem the scenario changes. We know that there are two defective

coins in a set of n coins and it is given that both are heavier (lighter) than the others in the set. Each weighing has k coins on each pan. The following two cases can arise:

1. If the two sets balance "=" either they both contain one false coin, or both the false coins are in the remaining $n-2k$ set. In this case, examining the coins in one pan, we can take the conclusion that whether there is the counterfeit coin or not. Where lays the characteristic of dynamic programming, that the result of a case depends on the previous case.

2. If the two sets do not balance "=" either one of them contain one defective coin and the other false coin is in the remaining $n-2k$ set or both the false coins lie in one of the k sets with none in the remaining $n-2k$ set.

4.2.1 Problem Formulation

The problem under consideration conveys that in the search space, there are n coins all of which are identical in exterior. By a standard or true coin, we indicate that its weight is precise to a value, say x unit, and a forged (or false or counterfeit) coin is a coin which only differs from a standard one in terms of its weight. If the weight of a fake coin is y unit, then one of the following situations may arise: $x > y$; $x < y$, i.e. the anomalous coin is either lighter or heavier than a true coin. Now, it is restricted through the problem statement that we do not have the information about the original weights of the coins. We are only provided with a single arm balance using which the relative weights of the coins can be found out. Now, depending on the number of fake coins in the search space the problem shows variation in the outcomes of the weighing process [18, 29].

To explain this, let us take one example. Say, only one coin is false among 10 coins. We put five coins in the left pan and five coins in the right pan, and say, the left pan goes upside and right pan downside. From this weighing we may assume two things: either the fake coin is heavier residing on the right pan while the coins on the left pan are all true, or the fake coin being lighter than a standard one, resides on the left pan. Thus, from this single weighing we cannot conclude anything else. Whatever be the case, we can conclude that at least one false coin is there on the coin set. Hence, to detect the counterfeit coin

categorically we must proceed through further weighing and our objective is to focus on meaningful weighing instead of redundant comparisons, i.e. each comparison can reduce the search space and finally lead us to the conclusion.

To formulate the problem, we must define the cases emphatically. Let T, H, and L denote a true coin, a heavier false coin, and a lighter false coin, respectively, whereas their weights are denoted as $\omega(T)$, $\omega(H)$, and $\omega(L)$, respectively. For one counterfeit coin problem as only single coin is anomalous, there are two possibilities: $\omega(T) > \omega(L)$ and $\omega(T) < \omega(H)$. The scenario is not as straightforward in case of two coins problem as similar outcome of weighing may claim different false coin combination and thus introduces conflicts [9]. For an example, if the left pan is heavier than the right, we cannot conclude immediately that both the false coins (heavier) are in the left pan or both (lighter) are in the right pan as there are several false coin combinations as discussed below.

4.2.2 Variation of the Problem

As our objective is to detect counterfeit coins from a set of coins, we are given the number of counterfeit coins and the cardinality of the set of coins. We are developing an algorithm to find 2 (two) false coins among *n* coins. Here we may observe two issues. What is the minimum value of *n*? Also, what kind of false coins is there, heavier or lighter than the original coin?

As a false coin means that it is either heavier or lighter than a true coin, finding out two false coins introduces several cases:

- **Both are heavier:** Two heavier false coins may be equally or unequally heavier. If we denote them as H_1 and H_2 and their weights as $\omega(H_1)$ and $\omega(H_2)$, we can define this case as:
  - $\omega(H_1) = \omega(H_2)$ [Equally heavier]
  - $\omega(H_1) > \omega(H_2)$ [Unequally heavier and we assume H_1 to be heavier than H_2]

- **Both are lighter:** Two lighter false coins may be equally lighter. If we denote them as L_1 and L_2 and their weights as $\omega(L_1)$ and $\omega(L_2)$, we can define this case as:

  - $\omega(L_1) = \omega(L_2)$ [Equally lighter]

  - $\omega(L_1) < \omega(L_2)$ [Unequally lighter and we assume that L_1 is lighter than L_2]

- **One is heavier and the other is lighter:** Let the true coin is denoted as T, and weight of the true coin is $\omega(T)$. Also, heavier and lighter coins are denoted as H and L respectively. Then,

  - $\omega(H) - \omega(T) = \Delta H$

  - $\omega(T) - \omega(L) = \Delta L$

The following situations may arise: $\Delta H > \Delta L$, $\Delta H < \Delta L$, $\Delta H = \Delta L$. Thus, there are seven different cases for finding out 2 counterfeit coins.

Lemma: *To find p counterfeit coins among n coins,* $n \geq 2p + 1$

Proof: If there is Single false coin among two coins, a standard coin must be provided to detect the false coin unless the weight of the correct coin is given. Thus, if $p = 1$ then min $(n) = 3 = 2p + 1$. Now if there are two false coins (we know the type of them) we can identify them from four coins if and only if two false coins are of different weights. If they are of same weights we cannot conclude which set of coins are true as there are equal number false and true coins. Thus, we need a standard coin and the total number of coins is five, i.e. $2p + 1$. To satisfy all the cases $min(n)$ for p false coins is $2p + 1$.

In general, if there are p counterfeit coins we can detect them from $p + 2$ (coins if all the p coins are of distinct weights, but if at least two false coins are of same weights we cannot distinguish between the set of original coins and the false coins. So, the satisfy the above cases, especially the case where all the false coins are of same weight we need at least one more true coin than the false coin, i.e. the minimum number coins are required is $p + p + 1 = 2p + 1$ [8, 9].

4.3 A General Algorithm for Differentiating Two Counterfeit Coins Problem

Let us consider that there are n coins. Our objective is to find two false coins using a single arm balance. Here we assume that we always put an equal number of coins on the both arms of the balance for weighing. Observing the relative status of the arms, i.e. left pan is heavier or lighter than the right pan, we decide on which pan the false coin resides.

Our algorithm proceeds by dividing the coins into three sets K_1, K_2, and K_3 such that the cardinality of the set K_1 and K_2 are equal. At first, K_1 and K_2 is put on the arms for weighing. Depending on the result of this weighing and the specification of the counterfeit coins, i.e. both are equally heavier or equally lighter or one is heavier than the other etc., which is provided, we decide that which set contains the false coin(s).

Now, as said above, we must divide into K_1, K_2, and K_3. As n is an integer there are three cases: (a) $n/3$, (b) $(n + 1)/3$, (c) $(n - 1)/3$. For the first case $|K_1| = |K_2| = |K_3| = n/3$. For the second case, $|K_1| = |K_2| = (n + 1)/3$ and $|K_3| = n - 2(n + 1)/3 = (n - 2)/3$ so there is difference of single coin between K_1 or K_2 and K_3. For the third case, $|K_1| = |K_2| = (n - 1)/3 + 1 = (n + 2)/3$ and $|K_3| = n - 2(n + 2)/3 = (n - 4)/3$. There is a difference of two coins between $|K_1|$ or $|K_2|$ and $|K_3|$.

After creating sets K_1, K_2, and K_3 they are put into weighing, depending on the result of weighing at any node, information flows downward to the leaves. At the leaf nodes we are performing two operations [4, 6]:

- **OCP (One Counterfeit Coin Problem)** (K_i) denotes the operation to find one counterfeit coin in K_i set. We call it whenever we are sure that there is only one false coin in K_i.

- **TCP (Two Counterfeit Coin Problem)** (K_i) is performed when we are sure that we are sure that there are two false coins in K_i.

4.4 Algorithms for Both the Counterfeit Coins are Equally Heavier or Equally Lighter

Case 1: $\omega(H_1) = \omega(H_2)$ [Equally Heavier]

***n* is divisible by 3, $\lvert K_1\rvert = \lvert K_2\rvert = \lvert K_3\rvert = n/3$**

The decision tree, in Figure 4.1 shows flow of algorithm for this version. At the root node, K_1 and K_2 are compared. At the internal node either K_1 or K_2 is compared with K_3. And at the leaf node OCP is applied to the respective seats or TCP is further applied taking its input either K_1 or K_2 or K_3, depending on the information it has got from the previous level of comparisons.

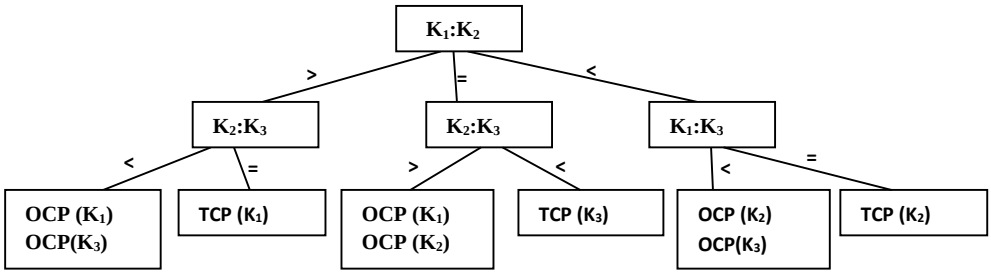


Figure 4.1: Decision tree for finding two equally heavier coins among *n* coins where *n*/3.

($n - 1$) is divisible by 3, $\lvert K_1\rvert = \lvert K_2\rvert = ((n-1)/3) + 1 = (n+2)/3$, $\lvert K_3\rvert = n - 2(n+2)/3 = (n-4)/3$

If *n* is such that ($n - 1$) is divisible by 3, and then the set K_3 has two coins less than the set K_1 and K_2. So, after comparing K_1 and K_2 at the root node, we like to perform the comparison between ($K_2 - 2$) or ($K_1 - 2$) with K_3 and depending on the information gained, we perform the OCP or TCP on the respective sets at the leaf node.

($n + 1$) is divisible by 3, $\lvert K_1\rvert = \lvert K_2\rvert = (n+1)/3$, $\lvert K_3\rvert = n - 2(n+1)/3$

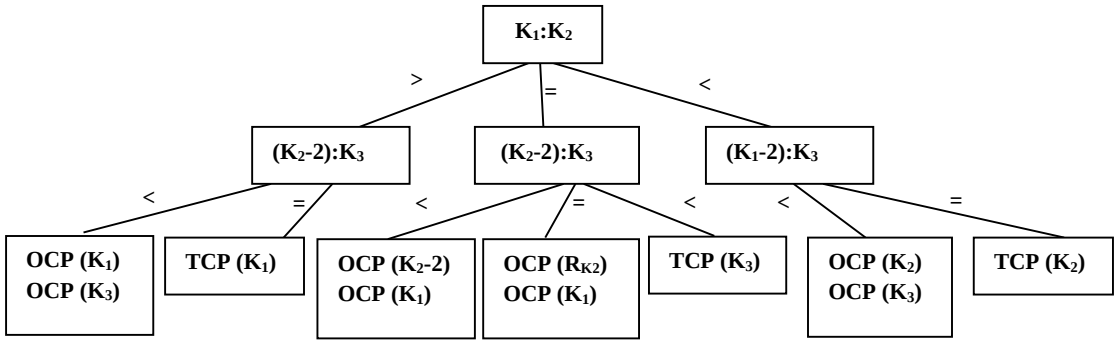


Figure 4.2: Decision tree for finding two equally heavier coins among *n* coins where (*n* – 1)/3.

If *n* is such that (*n* + 1) is divisible by 3, of course the sets K_1 and K_2 have one more coin than the set K_3. So, at the second level of comparison K_3 is compared to either with (K_1 – 1) or with (K_2 – 1).

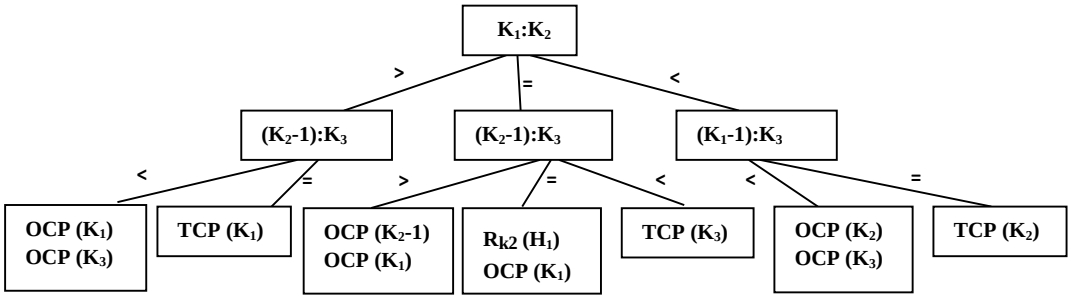


Figure 4.3: Decision tree for finding two equally heavier coins among *n* coins where (*n*+1)/3.

So far, we have seen that to find two false coins among *n* coins, the algorithm proceeds by comparing K_1 and K_2 at the root node and then at the second level performs the comparison between ($K_1 - i$) or ($K_2 - i$) with K_3, where *i* is 0 or 1 or 2 depending on the fact *n*/3 or (*n* – 1)/3 or (*n* – 2)/3 respectively, and at the leaf we simply call one coin problem for the set (K_j) or two coin problem for the set (K_j). OCP finds a single counterfeit coin from the respective set [4]. To illustrate, from Figure 4.3, we may observe that if $K_1 > K_2$ and there are two equally heavier coins in the set of *n* coins, there are several possibilities:

- Two heavier coins may reside in the set K_1 resulting $K_1 > K_2$.
- One heavier coin resides in K_1 and the other is in K_3.
- But K_1 and K_2 at the same time cannot contain one heavier coin each, as they are equally heavier, it would equal the pan.
- K_2 contains only true coins.

Thus, we perform $K_2 : K_3$, as we like to know whether set K_3 contains the false coin or not. If we find $K_2 < K_3$, we are then sure that the set K_3 has a false coin (as the set K_2 is of true coins). So now we have the information that set K_1 and K_2 contains a false coin each. But we do not know which one it is. So, we perform one counterfeit coin problem for K_1 and K_3 [4] individually; it takes only O(log *n*) time to find the odd ones.

If we find $K_2 = K_3$, we are sure that the set K_1 contains both the heavier coins (as K_2 is true set K_3 is also true). So, the algorithm (TCP) is recursively applied taking its input set K_1 instead of *n*, and proceeds at first dividing the set K_1 into three sets depending on the division by 3 and so on.

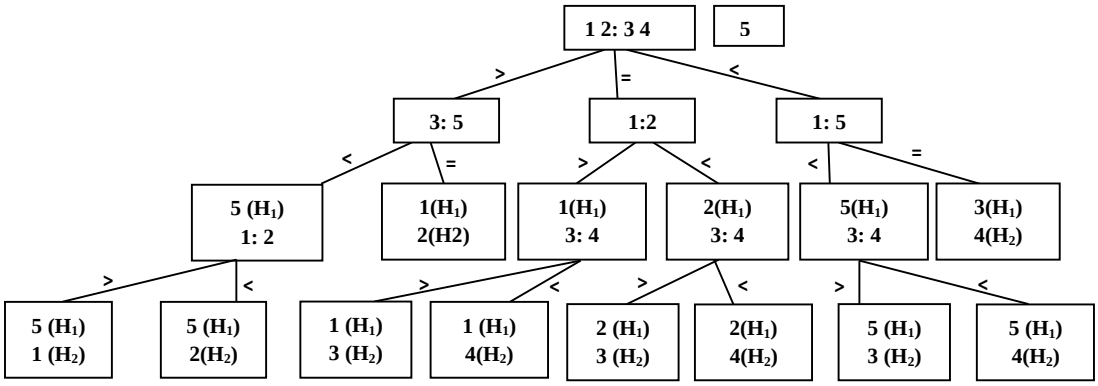


Figure 4.4: Decision tree for finding two equally heavier coins, where $n = 5$ coins.

We may observe the division of the set and the recursion of the algorithm reduces the cardinality of the set for the next iteration. Thus, this reduction may lead the algorithm to start with the set having only four coins or five coins. We call this the base for the TCP.

We will see later that only four comparisons are needed to identify the false coins in the base.

Let us take $n = 15$. So, $K_1 = K_2 = K_3 = 5$. From Figure 4.4, we may observe that OCP or TCP is applied to only 5 coins in this case. So, if we solve the problem for the base, i.e. for the set $n = 5$, for any number of coins greater than 5 (any large integer greater than 5) we can find the false coins.

Case 2: $\omega(L_1) = \omega(L_2)$ [Equally Lighter]

***n* is divisible by 3, $\|K_1\| = \|K_2\| = \|K_3\| = n/3$**

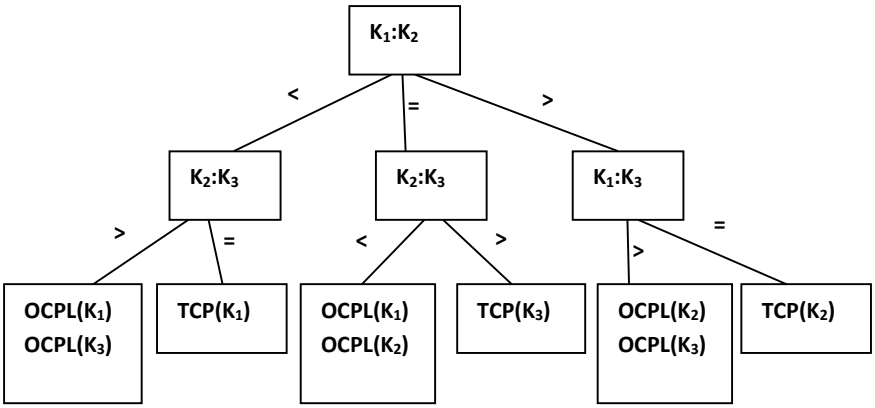


Figure 4.5: Decision tree for finding two false coins given $\omega(L_1) = \omega(L_2)$ among n coins where $n/3$.

As discussed in the Case 1 for $\omega(H_1) = \omega(H_2)$, Case 2 can be solved in the same way. Only the difference is that when a set is heavier than the other set it must contain only the true coins. The decision tree, in Figure 4.5 shows the flow of the algorithm for this version. At the root node, K_1 and K_2 are compared. At the internal node either K_1 or K_2 is compared with K_3, and at the leaf node OCP is applied to the respective sets or TCP is further applied taking its input either K_1 or K_2 or K_3, depending on the information it has got from the previous level of comparisons.

(n – 1) is divisible by 3, $|K_1| = |K_2| = ((n - 1)/3) + 1 = (n + 2)/3$, $|K_3| = n - 2(n + 2)/3 = (n - 4)/3$

If n is such that $(n - 1)$ is divisible by 3, and then the set K_3 has two coins less than the set K_1 and K_2. So, after comparing K_1 and K_2 at the root node, we like to perform the comparison between $(K_2 - 2)$ or $(K_1 - 2)$ with K_3 and depending on the information gained, we perform the OCP or TCP on the respective sets at the leaf node, shown in Figure 4.6.

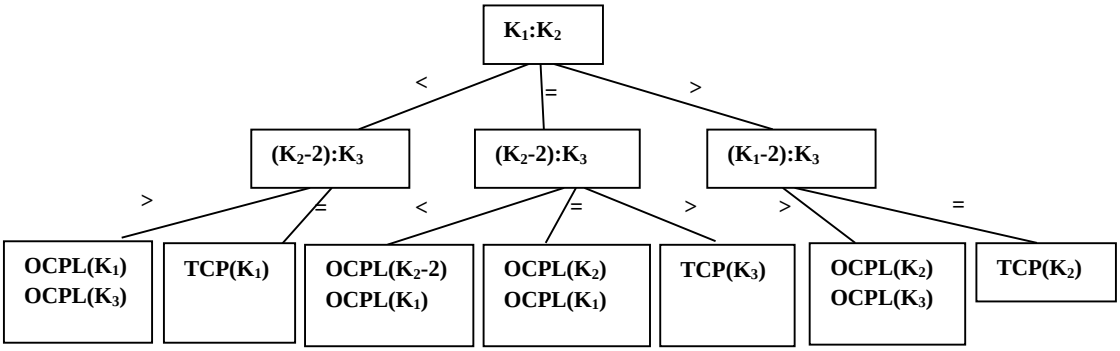


Figure 4.6: Decision tree for finding two false coins given $\omega(L_1) = \omega(L_2)$ among n coins where $(n - 1)/3$.

(n + 1) is divisible by 3, $|K_1| = |K_2| = (n + 1)/3$, $|K_3| = n - 2(n + 1)/3$

If n is such that $(n + 1)$ is divisible by 3, of course the sets K_1 and K_2 have one more coin than the set K_3. Subsequently, at the second level of comparison K_3 is compared to either with $(K_1 - 1)$ or with $(K_2 - 1)$. Again, at the leaf node, as we can see, OCP or TCP are called.

4.5 Algorithms for Both the Counterfeit Coins are Unequally Heavier or Unequally Lighter

Case 1: $\omega(H_1) > \omega(H_2)$ [Both are Unequally Heavier]:

n is divisible by 3, $|K_1| = |K_2| = |K_3| = n/3$

The decision tree in Figure 4.5 shows the flow of the algorithm for this version. At the root node, K_1 and K_2 are compared. If $K_1 = K_2$, then we are sure that they contain only true coins and K_3 has two false coins. Therefore, we perform TCP on the set K_3. If $K_1 > K_2$, there are some possibilities:

- K_1 and K_2 contains a false coin each and weight of the heavier coin in the set K_1 is greater than that in the set K_2.
- Both the heavier coins are in K_1.
- K_1 contains a false coin and K_3 contains a false coin.

Thus, in the second level of comparison, we check $K_2 : K_3$. If we find they are equal, then K_2 and K_3 have true coins and we perform TCP on K_1 as now K_1 contains both the false coins. If we find $K_2 < K_3$ then one false coin is in K_1 and other is in K_3 and K_2 is the set of true coins. So, we perform OCP on K_1 and K_3 separately. Again, as there are only two heavier coins, if $K_2 > K_3$ then the set K_1 and K_2 contains a false coin each and K_3 has true coins only. The false coin in K_1 is heavier than the false coin in K_2. Similarly, if we find $K_1 < K_3$ taking $K_1 : K_3$ we perform the operations as shown by the decision tree in Figure 4.7.

(n+1) is divisible by 3, $|K_1| = |K_2| = (n + 1)/3$, $|K_3| = n - 2(n + 1)/3$

If n is such that $n + 1$ is divisible by 3, then the algorithm does the same thing as for the version $n/3$ except for the second level of comparisons, it takes $(K_2 - 1)$ instead of K_2 and $(K_1 - 1)$ instead of K_3. At the third level of comparisons we see that if $(K_2 - 1) = K_3$ then $(K_2 - 1)$ and K_3 are both the set of true coins and the remaining coin of the set K_2 (i.e. the coin is K_{2R}) may be a false coin. Therefore, we take a true coin (here from the set K_3, as we already have known that this is true. Moreover, the first coin of the set K_3 is chosen) and compares it with the K_{2R}. If we find K_{2R} = a true coin from K_3 then K_1 contains both the false coins. Thus, we perform TCP on the set K_1. If K_{2R} is greater than the true coin, it is obvious that it is a heavier coin and to find the other heavy coin we perform OCP on K_1.

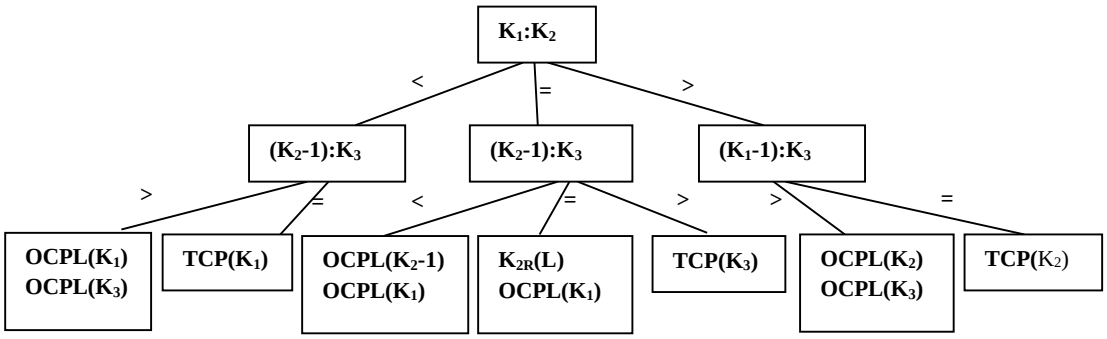


Figure 4.7: Decision tree for finding two false coins given $\omega(L_1) = \omega(L_2)$ among *n* coins where $(n + 1)/3$.

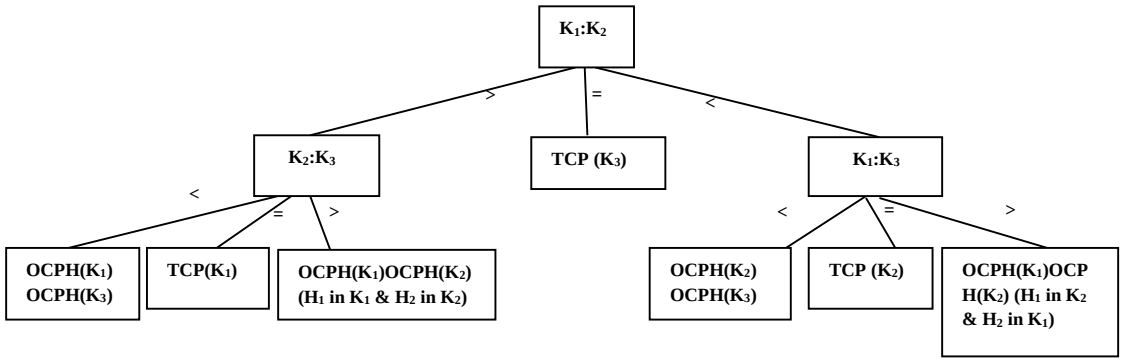


Figure 4.8: Decision tree for finding two unequally heavier coins among *n* coins where $n/3$.

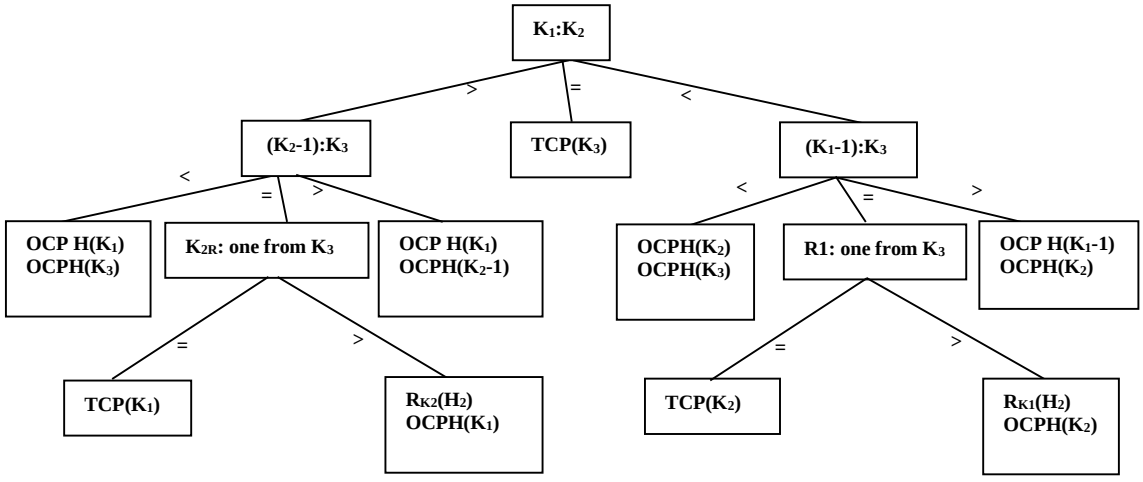


Figure 4.9: Decision tree for finding two unequally heavier coins among *n* coins where $(n+1)/3$.

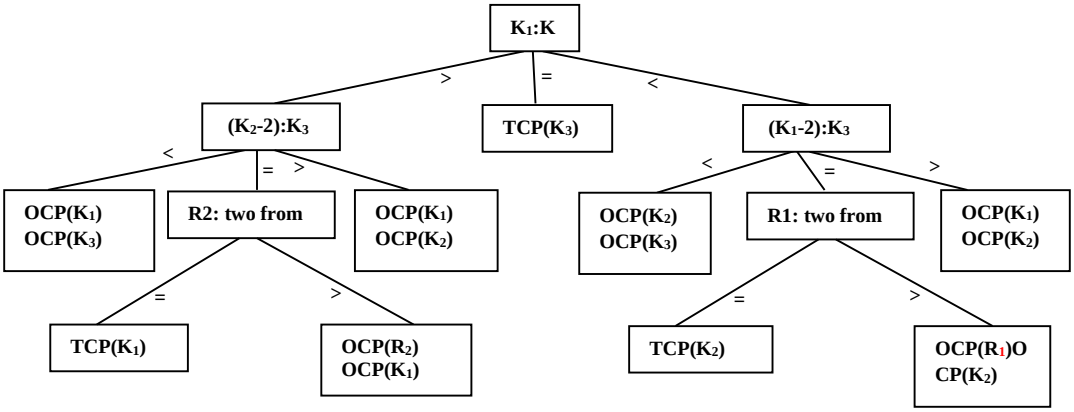


Figure 4.10: Decision tree for finding two unequally heavier coins among *n* coins where (*n*–1)/3.

(*n* – 1) is divisible by 3, $|K_1| = |K_2| = ((n-1)/3) + 1 = (n+2)/3$, $|K_3| = n - 2(n+2)/3 = (n-4)/3$

Similarly, if *n*–1 is divisible by 3 at the second level of comparison the algorithm takes K_2–2 instead of K_2 or K_1–1 instead of K_1 to compare with K_3. The algorithm performs as defined in the decision tree in Figure 4.8. To check the type of the remaining coins in the set K_1 or K_2, i.e. the set K_{1R} or K_{2R}, two true coins are now needed.

The decision tree as shown in Figure 4.10 shows the base problem for the case $\omega(H_1) > \omega(H_2)$. The base ($n = 5$) is solved using only four comparisons.

Case 2: $\omega(L_1) < \omega(L_2)$ [Both are Unequally Lighter]

***n* is divisible by 3, $|K_1| = |K_2| = |K_3| = n/3$**

The decision tree in Figure 4.7 shows the flow of the algorithm for this version. At the root node, K_1 and K_2 are compared. If $K_1 = K_2$, then we are sure that they contain only true coins and K_3 has two false coins. So, we perform TCP on the set K_3. If $K_1 < K_2$, there is some possibilities.

$\omega(H_1) > \omega(H_2)$

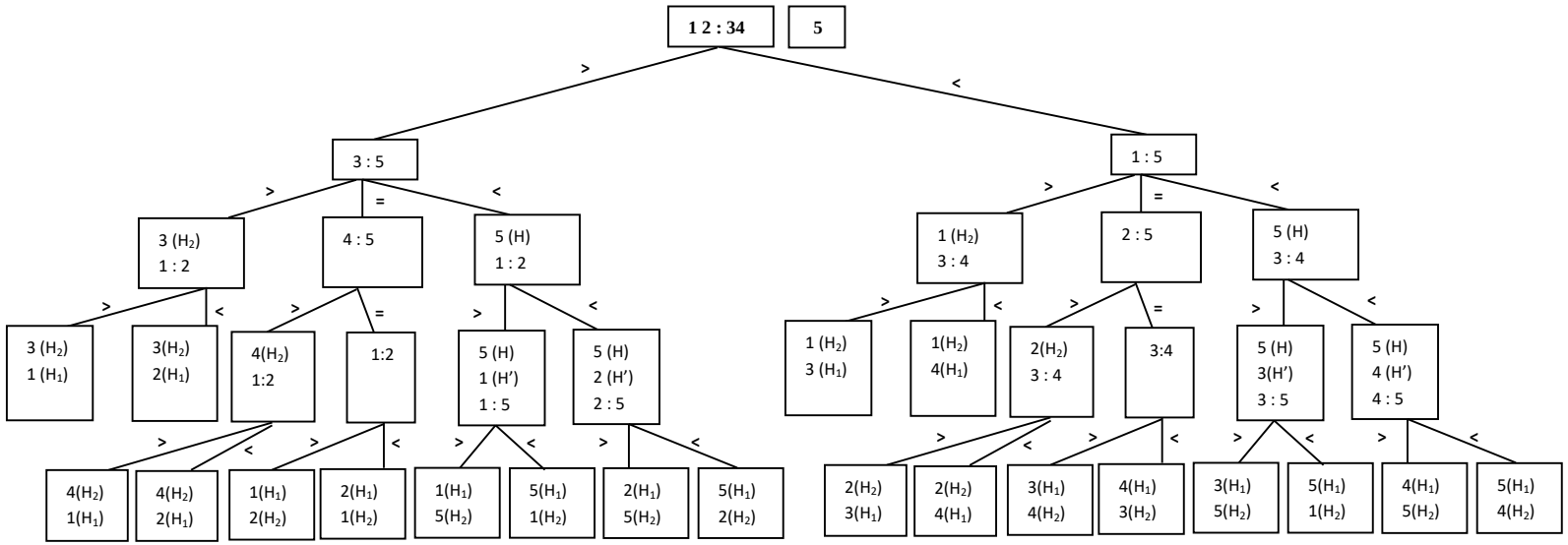


Figure 4.11: Decision tree for finding two unequally heavier coins among five coins.

- K_1 and K_2 contains a false coin each and weight of the lighter coin in the set K_1 is less than that in the set K_2.

- Both the lighter coins are in K_1.

K_1 contains a false coin and K_3 contains a false coin.

Thus, in the second level of comparison, we check $K_2 : K_3$. If we find they are equal, then K_2 and K_3 have true coins and we perform TCP on K_1 as now K_1 contains both the false coins. If we find $K_2 < K_3$ then one false coin is in K_1 and other is in K_2 and K_3 is the set of true coins. Therefore, we perform OCP on K_1 and K_2, separately [4, 5]. If $K_2 > K_3$ then the set K_1 and K_3 contains a false coin each and K_2 has true coins only. The false coin in K_1 is lighter than the false coin in K_2. Similarly, if we find $K_1 < K_3$ taking $K_1 : K_3$ we perform the operations as shown by the decision tree in Figure 4.12.

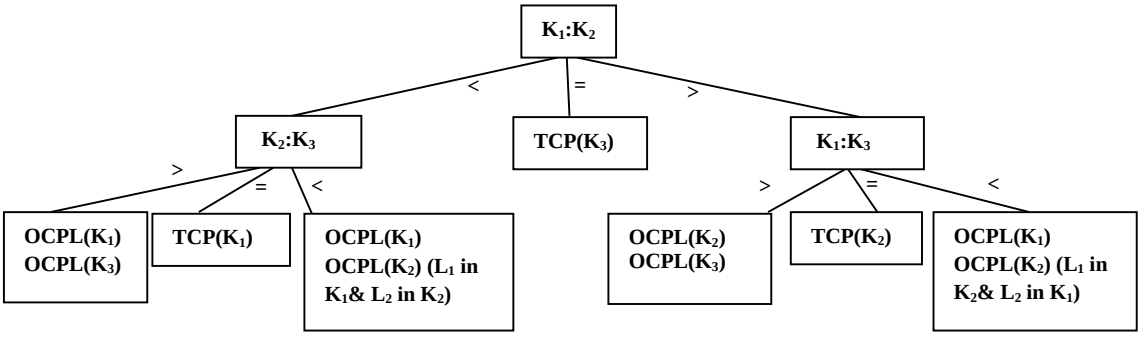


Figure 4.12: Decision tree for finding two false coins given $\omega(L_1) < \omega(L_2)$ among n coins where $n/3$.

$(n + 1)$ is divisible by 3, $|K_1| = |K_2| = (n + 1)/3$, $|K_3| = n – 2(n + 1)/3$

If n is such that $n+1$ is divisible by 3, then the algorithm does the same thing as for the version $n/3$ except for the second level of comparisons, it takes $(K_2–1)$ instead of K_2 and $(K_1–1)$ instead of K_3. At the third level of comparisons we see that if $(K_2–1) = K_3$ then $(K_2–1)$ and K_3 are both the set of true coins and the remaining coin of the set K_2 (i.e. the coin is K_{2R}) may be a false coin. Therefore, we take a true coin (here from the set K_3, as

we already have known that this is true. Moreover, the first coin of the set K_3 is chosen) and compares it with the K_{2R}. If we find K_{2R} = True coin from K_3 then K_1 contains both the false coins. Therefore, we perform TCP on the set K_1 [8]. If K_{2R} is less than the true coin, it is obvious that it is a lighter coin and to find the other light coin we perform OCP on K_1. The algorithm performs as defined in the decision tree in Figure 4.13.

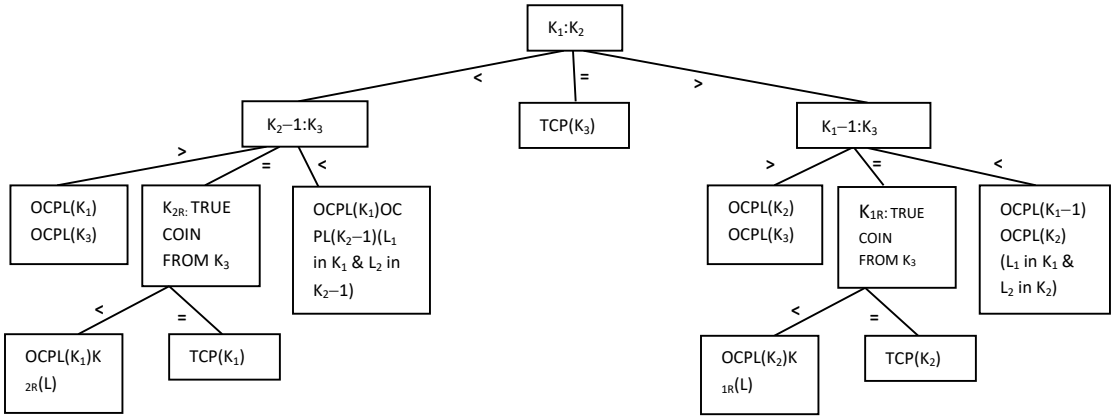


Figure 4.13: Decision tree for finding two false coins given $\omega(L_1) < \omega(L_2)$ among *n* coins where $(n + 1)/3$.

($n - 1$) is divisible by 3, $|K_1| = |K_2| = ((n - 1)/3) + 1 = (n + 2)/3$, $|K_3| = n - 2(n + 2)/3 = (n - 4)/3$

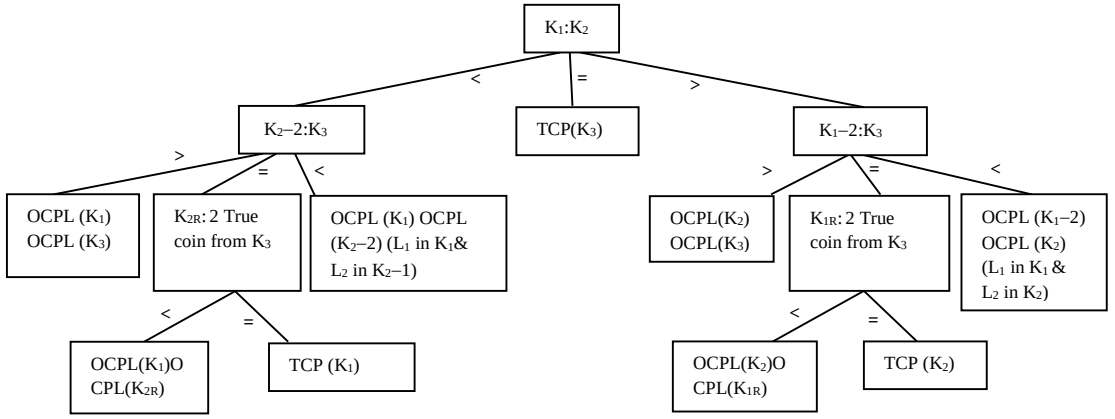


Figure 4.14: Decision tree for finding two false coins given $\omega(L_1) < \omega(L_2)$ among *n* coins where $(n–1)/3$.

Similarly, if $n-1$ is divisible by 3 at the second level of comparison the algorithm takes K_2-2 instead of K_2 or K_1-1 instead of K_1 to compare with K_3. The algorithm performs as defined in the decision tree in Figure 4.14. To check the type of the remaining coins in the set K_1 or K_2, i.e. the set K_{1R} or K_{2R}, two true coins are now needed.

4.6 Computational Complexity of the Algorithms

Each iteration n is divided into nearly three equal parts, and the cardinality of the set of coins on which the operations are performed, always reduces by a factor of three. Let us consider the case $n/3$. As we see at the ith level each set is of the cardinality $n/3^i$. Now if we reach to a set with five coins, then we can solve it using only four comparisons. So, let at ith level of comparison the cardinality of the set reduces to five. Thus, $n/3^i = 5$, i.e. $3^i = n/5$. Hence, $i = \log_3(n/5)$. Again, if TCP(K_i) is applied at each iteration before reaching a set with five coins, then $2\times i$ comparisons are required resulting in a total of $2\times i + 4$ comparisons.

If OCPH(K_i), i.e. the method to identify one counterfeit coin (heavier than a true coin), or OCPL(K_i), i.e. the method to identify single counterfeit coin (lighter than a true coin) is applied at the kth level of comparison, it is sure that before this iteration TCP() is applied for $k-1$ times. We know that OCPH() or OCPL() requires $\lceil \log_2 n \rceil$ comparisons and at the kth level it is to be applied on $n/3^k$ coins. The tree shown in Figure 4.15 depicts how the cardinality of the set decreases in each level of comparisons. Hence, it would require total number of $2|K|+2\log_3(n/3^k)$ comparisons. Therefore, in the worst case it would take $O(2|K|+2\log_3(n/3^k)) + O(2\times\log_3(n/5)+4)$, i.e. $O(\log n)$ comparisons as a whole [13].

4.7 Experimental Results

In this section, we discuss our algorithm in the analysis of the methods used, i.e. TCP() and OCP() with the purpose of showing the performance of the algorithm. As we observe in the decision tree structures in the previous section, at the leaf nodes we have applied OCP() or TCP() on some specific set of coins. Here, it is worth mentioning that to attain the leaves

starting from its root, the number of comparisons required is constant. Hence, the computational complexity of the algorithm depends on the complexity of these functions.

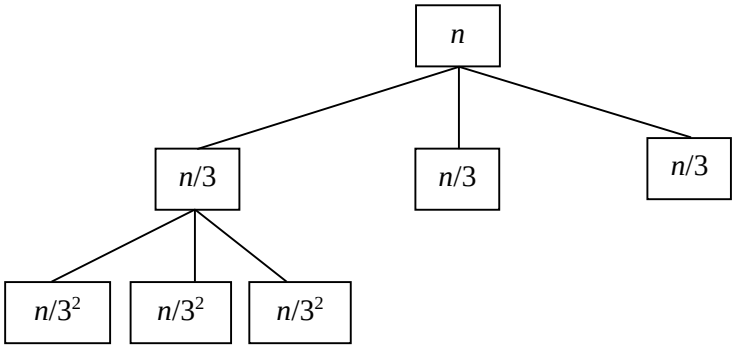


Figure 4.15: This tree shows how the cardinality of the set decreases in each level of comparisons.

Table 4.1: Average number of comparisons for different values of *n* considering Case A (equally heavier or lighter).

Number of coins (*n*)	Total number of comparisons (*S*)	Possible number of false coin combination *C* ($^nC_2$)	Average number of comparison (AVG = *S*/*C*)
10	211	45	4
18	1053	153	6
29	2939	406	7
46	9201	1035	8
54	13365	1431	9
72	27396	2556	10
82	34267	3321	10
100	54926	4950	11
108	65232	5778	11
144	130842	10296	12
198	251883	19503	12
200	254788	19900	12

Table 4.2: Average number of comparisons for different values of *n* considering Case B (unequally heavier or lighter).

Number of coins (*n*)	Total number of comparisons (*S*)	Possible number of false coin combinations *C* ($2\times{}^{n}C_2$)	Average number of comparisons (AVG = *S/C*)
10	500	90	5
18	1998	306	6
26	4928	650	7
70	50368	4830	10
72	53712	5112	10
80	65554	6320	10
100	112370	9900	11
107	127122	11342	11
127	177694	16002	11
146	265592	21170	12
161	315772	25760	12
162	306666	26082	11

To incorporate generalization, we choose some values of *n* so that it covers all the three categories for the subdivision of *n* and calculate the average number of comparisons requisite in the case of TCP(). As we have developed four such algorithms to cover all possible false coin combinations among the search space, we individually show the performance of the algorithms in terms of average number of comparisons required for different values of *n*.

Table 4.1 shows the variation of required number of weighing with the number of coins under consideration if the false coins are equally heavier (or equally lighter) [5]. In Figure 4.16(a), the horizontal axis denotes the total number of coins while the vertical axis refers to the average number of comparisons required to find two counterfeit coins among a set of identical looking coins. In this case we need to consider all possible combinations like, the counterfeit coin could be any coin at any position of the indexed location of the

given sequence and it could be either heavier (in case of equally heavier) or lighter (in case of equally lighter). To compute the average case complexity, we must consider all the possible false coin pairs. Hence, for a given value of n, there are ${}^{n}C_2$ numbers of combinations as the combination of ith heavier and jth heavier is same as the combination of jth heavier and ith heavier. Hence there are ${}^{n}C_2$ leaves in the decision tree. As a result, on an average number of comparisons required is O(log n) [5, 6, 8]. Taking this fact into account, there are ${}^{n}C_2$ various combinations of the false coins' positions. Thus, for a given value of n, there are $2\times{}^{n}C_2$ permutations as ith more heavy coin and jth less heavy coin in a pair of locations is different from ith less heavy coin and jth more heavy coin in the same pair of locations. Therefore, whenever we calculate the average number of comparisons required, we first sum up the total number of comparisons required to find all possible false coin pairs for a given value of n and then divide that value by $(2\times{}^{n}C_2)$. When plotted on a graph with the average number of comparisons along x-axis and the total number of coins along y-axis, the resulting curve shows that the average number of comparison follows O(log n) as shown in Figure 4.16(b).

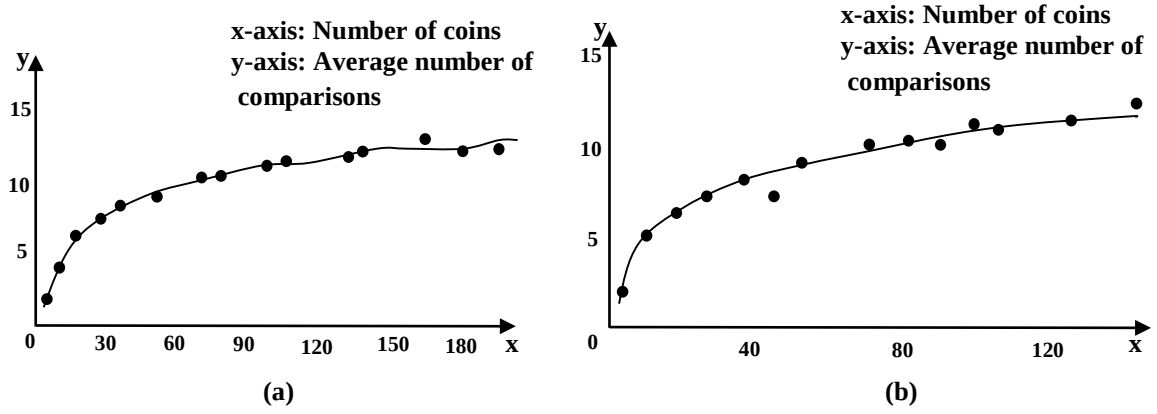


Figure 4.16: Graph with the average number of comparisons along x-axis and the total number of coins along y-axis considering **(a)** Case A (equally heavier or lighter), **(b)** Case B (unequally heavier or lighter).

4.8 Summary

In this chapter we have considered the two counterfeit coins problem, the usual objective of this problem is to find the false coin using a minimum number of comparisons, and in the form of a decision tree. We have developed two new algorithms for solving equally heaver, equally lighter and unequally heaver, unequally lighter counterfeit coins. The nobility of these two algorithms is that we can compute this algorithm in logarithmic time. We have considered all the possible case of the two counterfeit coins when n is divisible 3, as well as $n-1$ and $n+1$ is divisible by 3. We established a clear picture of two counterfeit coins problem without any ambiguity.

Chapter 5

Algorithms for $\Delta(\omega(H)) > \Delta(\omega(L))$ and $\Delta(\omega(H)) < \Delta(\omega(L))$

5.1 Overview

In this chapter, we are going to study about new algorithms for $\Delta(\omega(H)) > \Delta(\omega(L))$ and $\Delta(\omega(H)) < \Delta(\omega(L))$. This chapter is organized into five sections. In Section 5.2, we have discussed the algorithm for $\Delta(\omega(H)) > \Delta(\omega(L))$ (one heavier and the others are lighter, but the difference between the heavier and the original coin is greater than the difference between the lighter and the original coin). In Section 5.3, we have discussed $\Delta(\omega(H)) < \Delta(\omega(L))$ (one heavier and the other is lighter, but the difference between the heavier and the original coin is less than the difference between the lighter and the original coin). In Section 5.4, we have talked about the computational complexity of the algorithms. Experimental results have been included in Section 5.5, and a summary of the chapter is presented in Section 5.6.

5.2 $\Delta(\omega(H)) > \Delta(\omega(L))$: One Coin is Heavier and the Other is Lighter while the Difference between the Heavier and the Original Coin is Greater than the Difference between the Lighter and the Original Coin

The algorithm starts by dividing the coins into three sets K_1, K_2, and K_3 such that the sets K_1 and K_2 contain an equal number of coins. At first, K_1 and K_2 are placed on the arms for weighing [8, 9]. In the first case, as the cardinality is divisible by three, it is straightforward to divide it into three exactly equal subsets, say K_1, K_2, and K_3, where $|K_1| = |K_2| = |K_3| = n/3$. Similarly, for the second case, $|K_1| = |K_2| = (n+1)/3$ and $|K_3| = n - 2(n+1)/3 = (n-2)/3$. Thus, there is a difference of one coin between K_1 (or K_2) and K_3. For the third case, $|K_1|$

= $|K_2| = (n–1)/3 + 1 = (n+2)/3$ and $|K_3| = n – 2(n+2)/3 = (n–4)/3$, resulting a difference of two coins between K_1 (or K_2) and K_3.

Case 1:

***n* is divisible by 3, i.e. $\|K_1\| = \|K_2\| = \|K_3\| = n/3$**

The decision tree, in Figure 5.1 shows the flow of an algorithm for this version. At the root node, K_1 and K_2 are compared. If $K_1 = K_2$, we are sure that they contain only true coins and K_3 has two false coins [28, 47]. Therefore, we perform TCP on the set K_3. If $K_1 > K_2$, there are some possibilities:

- K_1 contains heavier coin while K_2 contains the lighter one.
- Both the false coins are in K_1 as the sum of the two false coins exceeds the sum of two true coins.
- K_1 contains a heavier coin and K_3 contains lighter coin while K_2 is the set of true coins only.
- K_1 is the set of true coins, and K_2 contains the lighter coin while the heavier coin is in K_3.

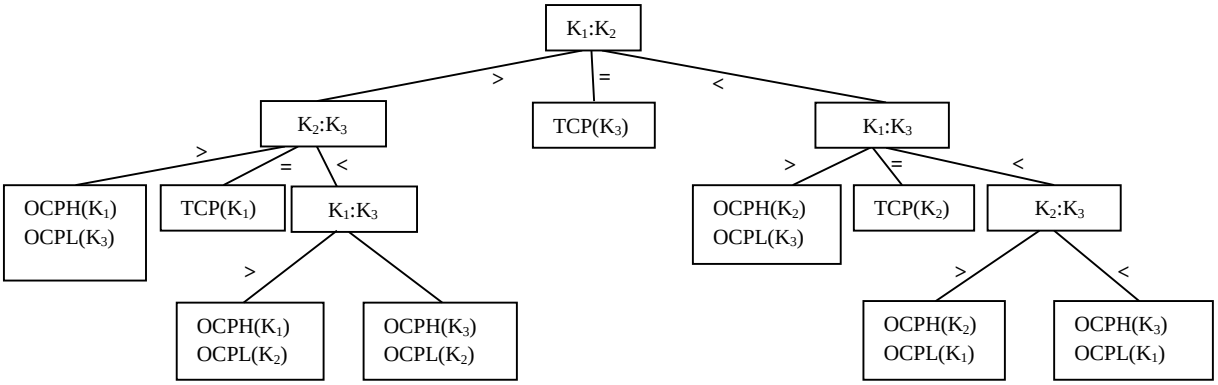


Figure 5.1: Decision tree for finding two false coins given $\Delta(\omega(H)) > \Delta(\omega(L))$ among n coins where $n|3$.

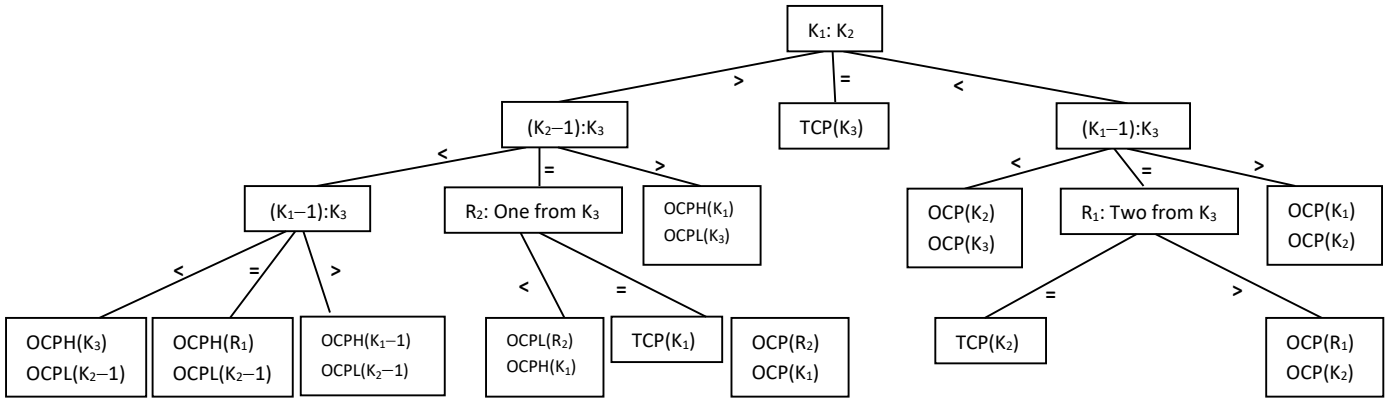


Figure 5.2: Decision tree for finding two false coins given $\Delta(\omega(H)) > \Delta(\omega(L))$ among n coins where $(n+1)|3$.

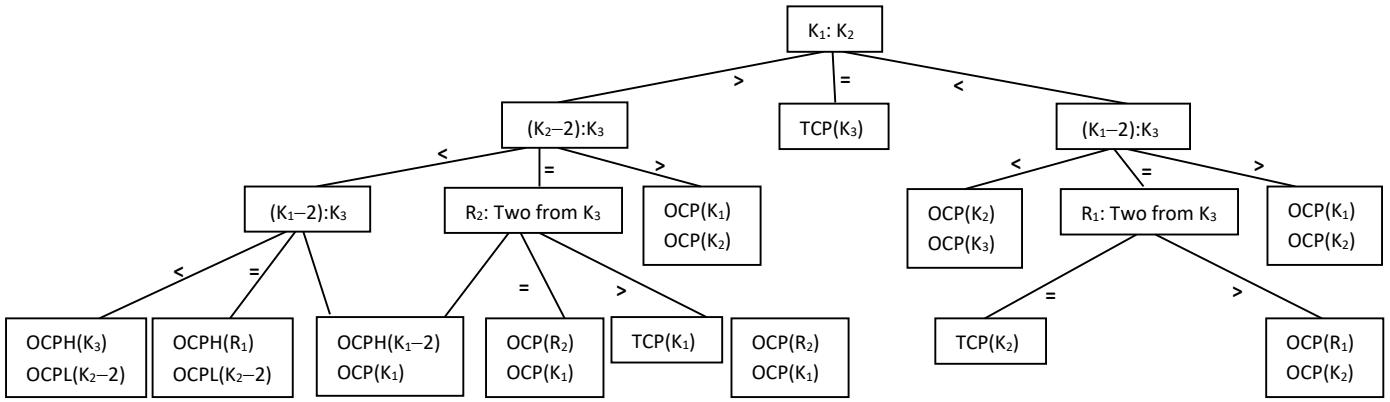


Figure 5.3: Decision tree for finding two false coins given $\Delta(\omega(H)) > \Delta(\omega(L))$ among n coins where $(n-1)|3$.

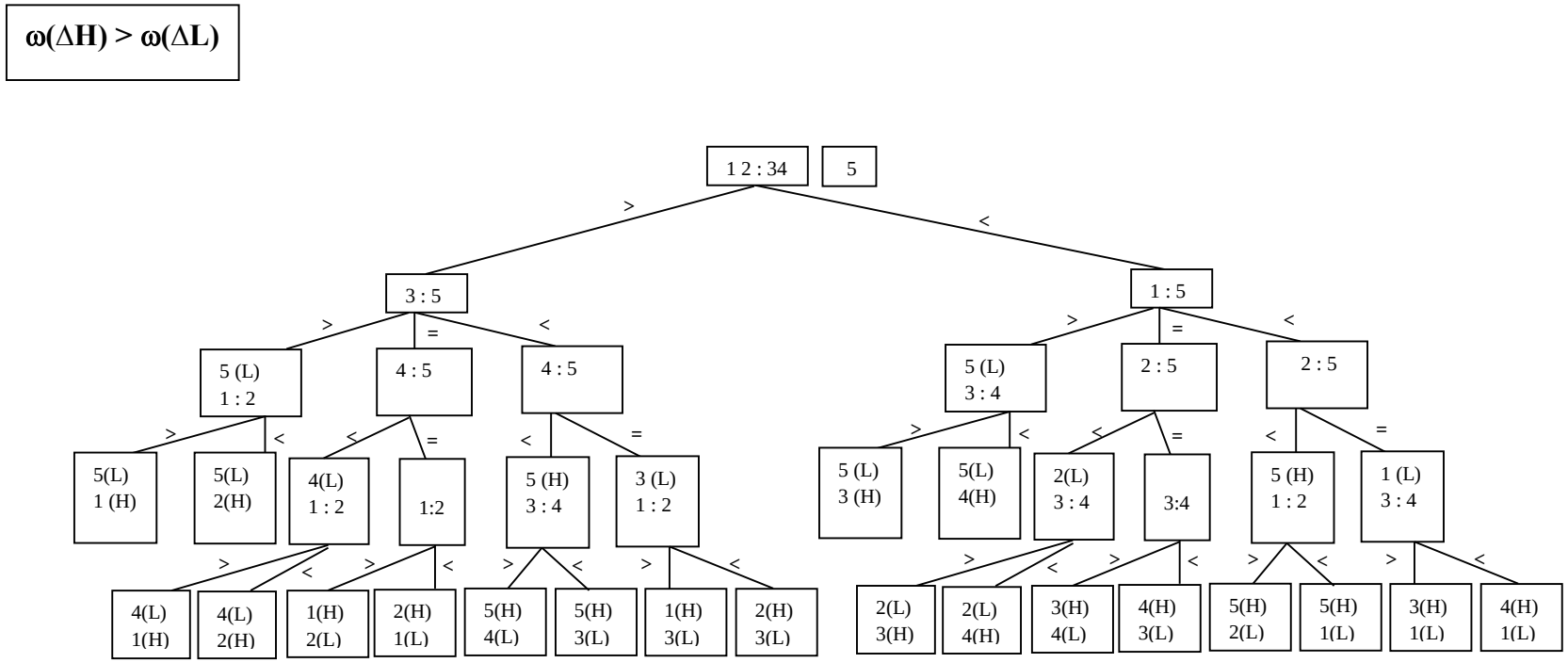


Figure 5.4: Decision tree for finding two false coins given $\Delta(\omega(H)) > \Delta(\omega(L))$ among five coins.

Hence, there is no possibility of containing both the false coin in K_2. Therefore, we check K_2:K_3. If they are equal K_1 conations both the false coins. If it results in inequality decision is taken based on the result of weighing as shown in Figure 5.1. The third branch from the root performs K_1:K_3 and so on [9, 16, 46].

Case 2:

***n*+1 is divisible by 3, i.e. $\|K_1\| = \|K_2\| = (n+1)/3$ and $\|K_3\| = n-2(n+1)/3$**

In this case, the algorithm does the same thing as for the version *n*/3 except for the 2nd level of comparisons where it takes (K_2–1) instead of K_2 and (K_1–1) instead of K_1. The operations in the consecutive levels are shown in the decision tree in Figure 5.2.

Case 3:

$(n-1)$ is divisible by 3, i.e. $\|K_1\| = \|K_2\| = ((n-1)/3) + 1 = (n+2)/3$ and $\|K_3\| = n - 2(n+2)/3 = (n-4)/3$

In this case, the algorithm does the same thing as for the version *n*/3 except for the 2nd level of comparisons where it takes (K_2–2) instead of K_2 and (K_1–2) instead of K_1. The operations in the consecutive levels are shown in the decision tree in Figure 5.3. The decision tree as shown in Figure 5.4 shows the base problem for the case $\Delta(\omega(H)) > \Delta(\omega(L))$. The base ($n = 5$) is solved using only four comparisons.

5.3 $\Delta(\omega(H)) < \Delta(\omega(L))$, i.e. One Coin is Heavier and the Other is Lighter while the Difference between the Heavier and the Original Coin is Less than the Difference between the Lighter and the Original Coin

Case 1:

***n* is divisible by 3, i.e. $\|K_1\| = \|K_2\| = \|K_3\| = n/3$**

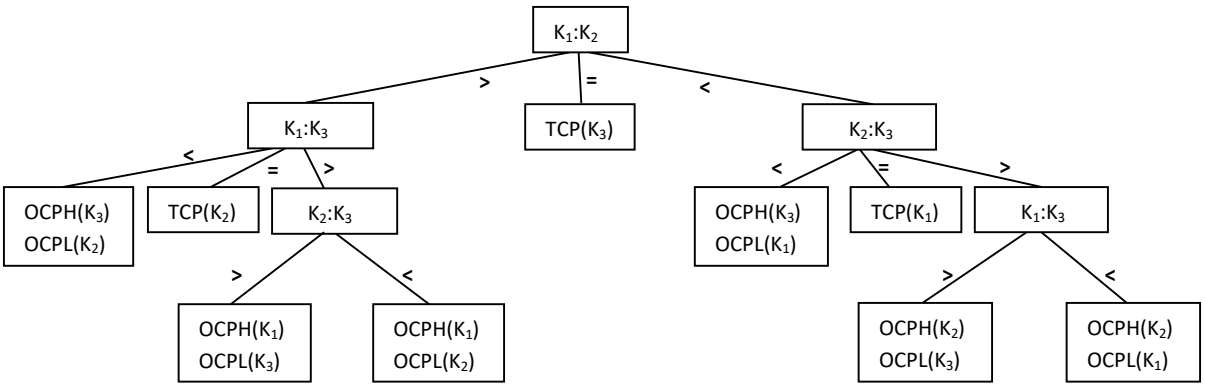


Figure 5.5: Decision tree for finding two false coins given $\Delta(\omega(H)) < \Delta(\omega(L))$ among *n* coins where $n|3$.

The decision tree, in Figure 5.5, shows the flow of an algorithm for this version. At the root node, the comparison between K_1 and K_2 are performed. If $K_1 = K_2$, we are sure that they contain only true coins and K_3 has two false coins [8, 15, 50]. Therefore, we perform TCP on the set K_3. If $K_1 > K_2$ there are some possibilities:

- K_1 contains heavier coin while K_2 contains, the lighter one.
- Both the false coins are in, K_2 as the sum of the two false coins is less than the sum of two true coins.
- K_1 contains a heavier coin and K_3 contains lighter coin while K_2 is the set of true coins only.
- K_1 is the set of true coins, and K_2 contains the lighter coin while the heavier coin is in K_3.

Hence, there is no possibility of containing both the false coins in K_1. Therefore, we check K_1:K_3. If they are equal, K_2 contains both the false coins. If it results in inequality, the decision is taken based on the result of weighing as shown in Figure 5.5. The third branch from the root performs K_2:K_3, and so on.

Case 2:

***n* +1 is divisible by 3, i.e. $\|K_1\| = \|K_2\| = (n+1)/3$ and $\|K_3\| = n-2(n+1)/3$**

In this case, the algorithm does the same thing as for the version $n|3$ except for the second level of comparisons where it takes (K_2-1) instead of K_2 and (K_1-1) instead of K_1. The operations in the consecutive levels are shown in the decision tree in Figure 5.6.

Case 3:

$(n-1)$ is divisible by 3, i.e. $\|K_1\| = \|K_2\| = ((n-1)/3) + 1 = (n+2)/3$ and $\|K_3\| = n-2(n+2)/3 = (n-4)/3$

In this version, the algorithm is same as for the version $(n+1)|3$ except for the second level of comparisons where it takes (K_2-2) instead of K_2 and (K_1-2) instead of K_1. The operations in the consecutive levels are shown in the decision tree in Figure 5.7. The decision tree as shown in Figure 5.8 shows the base problem for the case $\Delta(\omega(H)) < \Delta(\omega(L))$. The base ($n = 5$) is solved using only four comparisons.

5.4 Computational Complexity of the Algorithms

At each iteration, n is divided into nearly three equal parts, and the cardinality of the set on which the operations are performed, always reduced by a factor of 3. Let us consider the case $n|3$. As we observe at the ith level, each set is of cardinality $n/3^i$. Now, if we reach the set with five coins, then we can solve it using only four comparisons [6, 7, 14]. Therefore, at the ith level, the cardinality of the set reduces to five. Thus, $n/3^i = 5$, i.e. $3^i = n/5$ and $i = \log_3(n/5)$. Again, if TCP(K_i) is applied at each iteration before reaching a set with five coins, then only $2\times i$ comparisons are required resulting in $2\times i + 4$ comparisons in total.

On the other hand, if OCPH(K_i) (or OCPL(K_i)) is applied at the kth level of comparison, it is definite that prior to that iteration, TCP() is performed for $k-1$ times. We know that OCPH() (or OCPL()) requires $\lceil \log_2 n \rceil$ comparisons and at the kth level it is to be applied to $n/3^k$ coins. It requires a total number of $2|K| + 2\log_3(n/3^k)$ comparisons.

Therefore, in the worst-case it would take $O(2|K| + 2\log_3(n/3^k)) + O(2\times\log_3(n/5) + 4)$, that reduces to $O(\log n)$ comparisons as a whole [5].

5.5 Experimental Results of the Algorithms

In this section, we discuss our algorithm in the analysis of the methods used, i.e. TCP() and OCP() with the purpose of showing the performance of the algorithm. In this section, we discuss our algorithm in the analysis of the methods used, i.e. TCP() and OCP() with the purpose of showing the performance of the algorithm. As we observe in the decision tree structures in the previous section, at the leaf nodes we have applied OCP() or TCP() on some specific set of coins. Here, it is worth mentioning that to attain the leaves starting from its root, the number of comparisons required is constant; hence, the computational complexity of the algorithm depends on the complexity of these functions.

To incorporate generalization, we choose some values of *n* so that it covers all the three categories for the subdivision of *n* and calculate the average number of comparisons requisite in the case of TCP(). As we have developed four such algorithms to cover all possible false coins combination among the search space, we individually show the performance of the algorithms in terms of average number of comparisons required for different values of *n*. Table 5.1 shows the variation of required number of weighing with the number of coins under consideration. Table 5.1 is shown in the graphical representation in Figure 5.9.

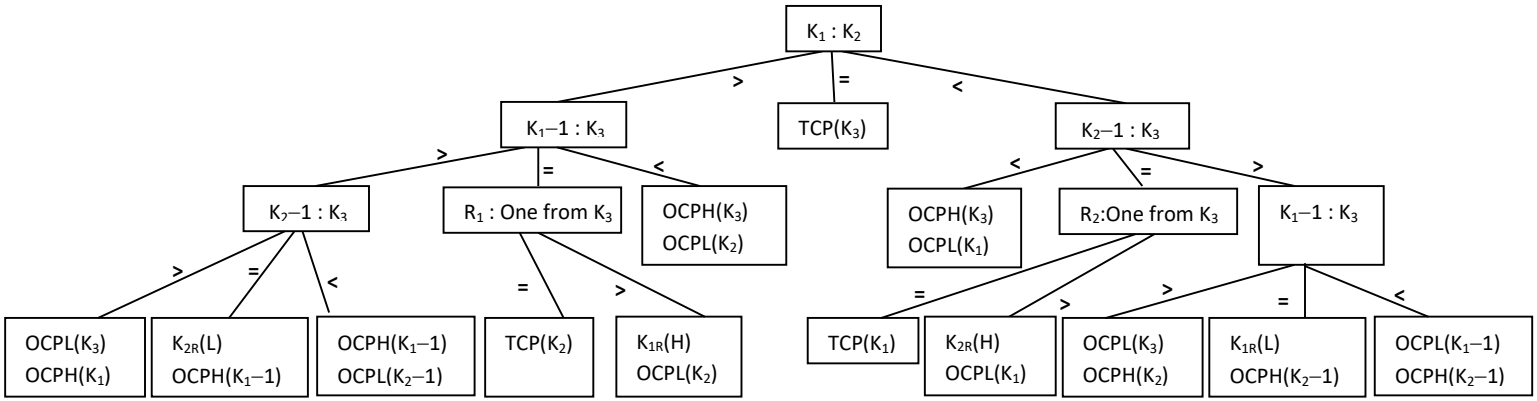


Figure 5.6: Decision tree for finding two false coins given $\Delta(\omega(H)) < \Delta(\omega(L))$, among n coins where n+1|3.

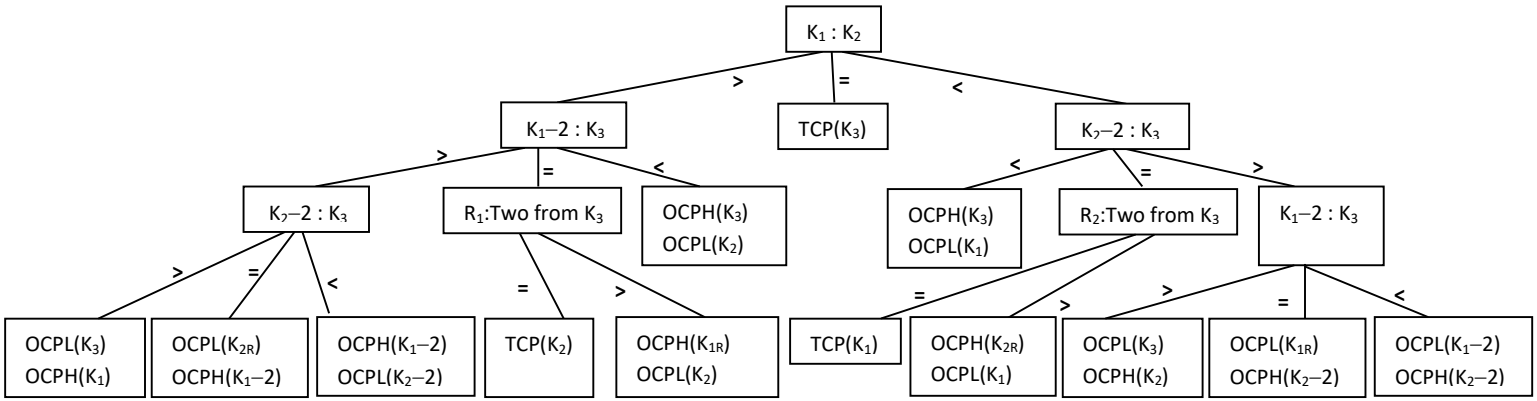


Figure 5.7: Decision tree for finding two false coins given $\Delta(\omega(H)) < \Delta(\omega(L))$ among n coins where (n–1)|3.

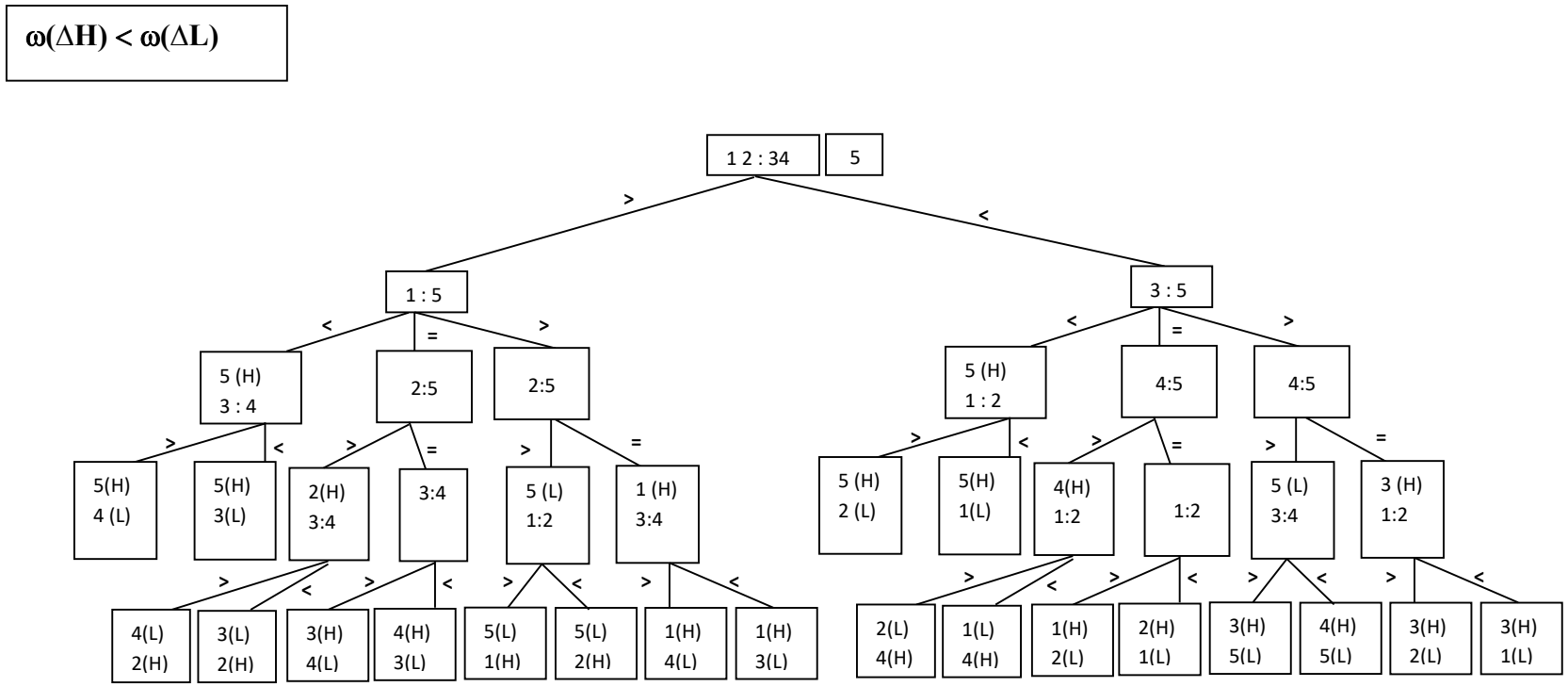


Figure 5.8: Decision tree for finding two false coins given $\Delta(\omega(H)) < \Delta(\omega(L))$ among five coins.

Table 5.1: Average number of comparisons for different values of n considering (one heavier and one lighter coin with $\Delta(\omega(H)) > \Delta(\omega(L))$.

Number of coins (n)	Total number of comparisons (S)	Possible number of false coin combinations ($C = 2\times{}^{n}C_2$)	Average number of comparisons (AVG = S/C)
10	484	90	5
18	1974	306	6
20	2354	380	6
46	17966	2070	8
56	28172	3080	9
72	52572	5112	10
82	66314	6642	9
100	113582	9900	11
108	133056	11556	11
144	252852	20592	11
198	520602	39006	13
200	517606	39800	13

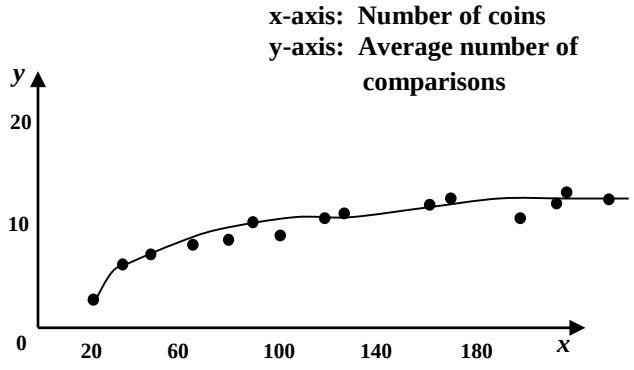


Figure 5.9: Graph with the average number of comparisons along the x-axis and the total number of coins along y-axis considering $\Delta(\omega(H)) > \Delta(\omega(L))$.

5.6 Summary

In this chapter, we have considered the two counterfeit coins problem, the usual objective of this problem is to find the false coin using a minimum number of comparisons, and in the form of a decision tree. We have developed new algorithms for solving. One heavier and the others are lighter, but difference between the heavier and the original coin is greater than the difference between the lighter and the original coin denoted as $\Delta(\omega(H)) > \Delta(\omega(L))$, and One heavier and the other is lighter, but difference between the heavier and the original coin is less than the difference between the lighter and the original coin denoted as $\Delta(\omega(H)) < \Delta(\omega(L))$. The nobility of the algorithms is that we can compute this algorithm in logarithmic time. We have considered all the possible case of the two counterfeit coins when n is divisible 3 when $n-1$ is divisible by and $n+1$ is divisible by 3. We established a clear picture of two counterfeit coins problem without any ambiguity.

Chapter 6

An Algorithm for Solving Two Coins Counterfeiting with $\Delta\omega(H) = \Delta\omega(L)$

6.1 Overview

In this chapter, we are going to study about new algorithms for $\Delta(\omega(H)) = \Delta(\omega(L))$. This chapter is organized into five sections. In Section 6.2, we have discussed about the formulation of the problem. In Section 6.3, we have discussed about the algorithm $\Delta(\omega(H)) = \Delta(\omega(L))$ (difference between the heavier and the original coin is equal to the difference between the true and the lighter coin). In Section 6.4, we have discussed the computational complexity of the algorithms. In Section 6.5, we have shown the experimental results. A summary of the chapter is presented in Section 6.6.

6.2 Formulation of the Problem

The problem under consideration conveys that in the search space, there are *n* coins all of which are identical in exterior. By a standard or true coin, we indicate that its weight is precise to a value, say *x* unit, and a forged (or false or counterfeit) coin is a coin which only differs from a standard one in terms of its weight. If the weight of a fake coin is y unit, then one of the following situations may arise: $x > y$; $x < y$, i.e. the anomalous coin is either lighter or heavier than a true coin. Now, it is restricted through the problem statement that we do not have the information about the original weights of the coins. We are only provided with a single arm balance using which the relative weights of the coins can be found out.

Now, depending on the number of fake coins in the search space the problem shows variation in the outcomes of the weighing process [7, 42, 45]. To explain this, let us take one example. Say, only one coin is false among ten coins. We put five coins in the left pan

and five coins in the right pan, and say, the left pan goes upside and right pan downside. From this weighing we may assume two things: Either the fake coin is heavier residing on the right pan while the coins on the left pan are all true, or the fake coin being lighter than a standard one, resides on the left pan. Thus, from this single weighing we cannot conclude anything else. Whatever be the case, we can definitely conclude that at least one false coin is there in the coin set. Hence, to detect the counterfeit coin categorically we have to proceed through further weighing and our objective is to focus on meaningful weighing instead of redundant comparisons, i.e. each comparison can reduce the search space and finally lead us to the conclusion.

To formulate the problem we have to define the cases emphatically. Let T, H, and L denote a true coin, a heavier false coin, and a lighter false coin, respectively, whereas their weights are denoted as $\omega(T)$, $\omega(H)$, and $\omega(L)$, respectively. For one counterfeit coin problem as only single coin is anomalous, there are two possibilities: $\omega(T) > \omega(L)$ and $\omega(T) < \omega(H)$. The scenario is not so straightforward in case of two coins problem as similar outcome of weighing may claim different false coins combination and thus introduces conflicts. For an example, if the left pan is heavier than the right, we cannot conclude immediately that both the false coins (heavier) are in the left pan or both of them (lighter) are in the right pan as there are several false coins combinations as discussed below [7, 8, 38, 43].

One of the false coins is heavier whereas another is lighter than a standard coin. In this case, one of the variations occurs when the false coins are equally heavier and lighter, i.e. the difference in weight of the heavier coin and a standard coin is equal to that of a standard coin and the lighter coin, $\omega(H) - \omega(T) = \omega(T) - \omega(L)$. We are defining this special problem as $\Delta\omega(H) = \Delta\omega(L)$.

6.3 Algorithm for $\Delta\omega(H) = \Delta\omega(L)$ (Difference between the Heavier and the Original Coin is Equal to the Difference between the True and the Lighter Coin)

From our earlier discussion it is evident that to solve two counterfeit coins problem in general, one needs to consider all the variations of the problem and a minimal set of

algorithms must be developed that covers all those alternatives. In this context, as we deal with the weight of the false coins as well as a standard coin as a real number only, more precisely an integer, we keep in mind the following assumptions or facts [7, 8, 51].

- Any positive integer n can be fallen into either of the three categories as follows: (i) n is divisible by 3, i.e. $n|3$, (ii) $n+1$ is divisible by 3, i.e. $(n+1)|3$, and (iii) $n-1$ is divisible by 3, i.e. $(n-1)|3$.

- Each weighing is performed taking two sets of coins having equal cardinal number into two pans of a single arm balance.

- Each level of weighing between two sets essentially decreases the search space and a series of such weighing eventually forms a tree where an intermediate node represents weighing, an edge between two successive nodes represents the outcome of the weighing (i.e., left pan is heavier or lighter or equal with respect to the right pan), and a leaf node represents a false coin combination.

As any instance may belong to any of the three variants of the problem, i.e., the value n satisfies either of the three cases, $n|3$ or $(n+1)|3$ or $(n-1)|3$, The algorithm starts by dividing the coins into three sets K_1, K_2, and K_3 such that the sets K_1 and K_2 contain equal number of coins. At first K_1 and K_2 are placed on the arms for weighing. Depending on the outcome of this weighing and the specification of the false coins, we select the sets that are to be weighed further to detect the false coin(s).

Here our objective is to divide the coins into three almost equal parts such that two such subsets are on weighing whereas the third one is left outside, and after a constant number of weighing we can reduce our suspected coins set analyzing the series of outcomes of subsequent weighing. For the first case as the cardinality is divisible by three, it is straightforward to divide it into three exactly equal subsets, say K_1, K_2, and K_3, where, $|K_1| = |K_2| = |K_3| = n/3$. Similarly, for the second case, $|K_1| = |K_2| = (n+1)/3$ and $|K_3| = n - 2(n+1)/3 = (n-2)/3$. Thus, there is a difference of one coin between K_1 (or K_2) and K_3. For the third case, $|K_1| = |K_2| = (n-1)/3 + 1 = (n+2)/3$ and $|K_3| = n - 2(n+2)/3 = (n-4)/3$, resulting a difference of two coins between K_1 (or K_2) and K_3.

Case 1:

***n* is divisible by 3, i.e. $\|K_1\| = \|K_2\| = \|K_3\| = n/3$**

The decision tree, in Figure 6.1, shows the flow of algorithm for this version. At the root node, K_1 and K_2 are compared. At the internal node either K_1 or K_2 is compared with K_3. And at the leaf node OCPH is applied to the respective sets or OCPL is further applied taking its input either K_1 or K_2 or K_3, depending on the information it has got from the previous level of comparisons. If K_1 is equal to K_2 then TCP is applied on K_1, K_2, and K_3. If $K_1 > K_2$, there are some possibilities,

- The heavier coin is in K_1 and the lighter coin is in K_2.
- The heavier coin is in K_1 and K_2 contains only true coins.
- The lighter coin is in K_2 and K_1 contains only true coins.

So, in the second level K_2 is weighed with K_3. If $\omega(K_2) > \omega(K_3)$, we are sure that the heavier coin is in K_1, while the lighter coin is in K_2. If $\omega(K_2) < \omega(K_3)$ we are sure that the lighter coin is in K_2, but we have to again compare K_1 to K_3 to conclude whether the heavier coin is in K_1 or in K_3.

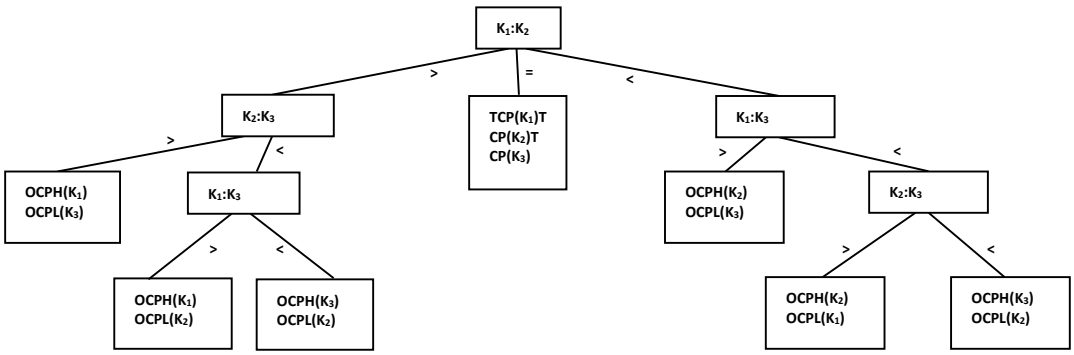


Figure 6.1: Decision tree for finding two false coins given $\Delta(\omega(H)) = \Delta(\omega(L))$, among *n* coins where $n|3$

Case 2:

(n+1) is divisible by 3, i.e. $|K_1| = |K_2| = (n+1)/3$ and $|K_3| = n-2(n+1)/3$

In this case, the algorithm does the same thing as for the version $n|3$ except for the second level of comparisons where it takes (K_2–1) instead of K_2 and (K_1–1) instead of K_1. The operations in the consecutive levels are shown in the decision tree in Figure 6.2. If $\omega(K_1) = \omega(K_2)$, we can have no idea in which the false coins may reside, because in this case we only can conclude that both the false coins are in one set. So, we must apply TCP(K_i) for i = 1, 2, 3. In the right most branches same operations are performed as has been discussed for the left branch. The difference is that we use K_1 instead of K_2 here and K_2, instead of K_1.

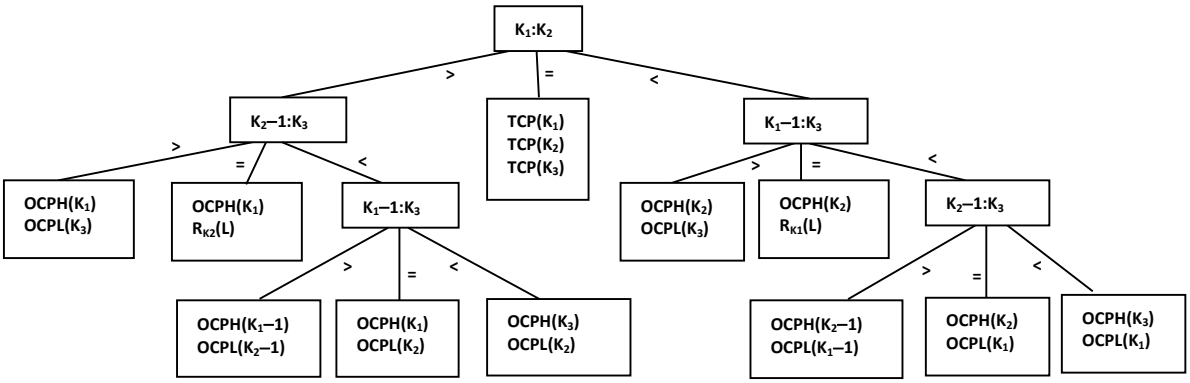


Figure 6.2: Decision tree for finding two false coins given $\Delta(\omega(H)) = \Delta(\omega(L))$, among n coins where $(n+1)|3$.

Case 3:

(n–1) is divisible by 3, i.e. $|K_1| = |K_2| = ((n-1)/3) + 1 = (n+2)/3$ and $|K_3| = n-2(n+2)/3 = (n-4)/3$

In this version, the algorithm is same as for the version (n+1)|3 except for the second level of comparisons where it takes (K_2–2) instead of K_2 and (K_1–2) instead of K_1. The operations in the consecutive levels are shown in the decision tree in Figure 6.3.

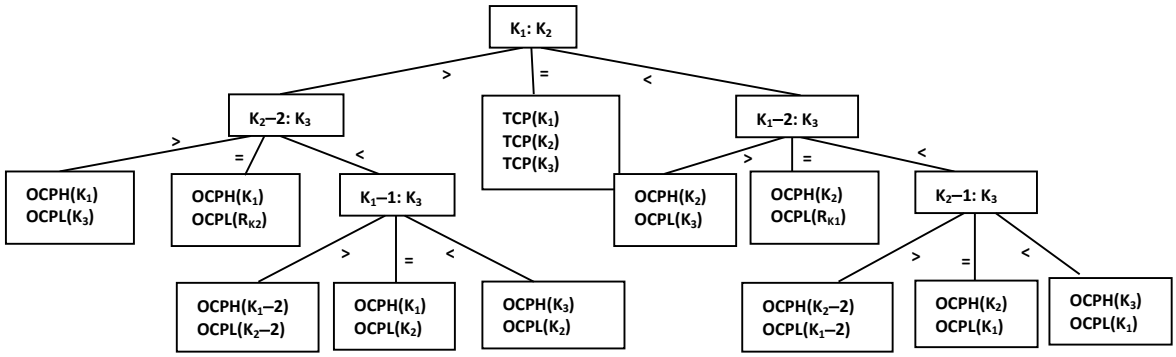


Figure 6.3: Decision tree for finding two false coins given $\Delta(\omega(H)) = \Delta(\omega(L))$, among n coins where (n–1)|3.

Now, we are discussing a classical case of five coins out of which two are false coins, first we are considering four coins two in each pan and keeping fifth coin separately; proceeding as per our algorithm we are getting information about the false coins as shown in the Figure 6.4.

6.4 Computational Complexity of the Algorithm

In each iteration, n is divided into three nearly equal parts, and the cardinality of the set of coins on which the operations are essentially performed, always reduces by a factor of three [13, 44, 49]. We notice that at the ith level each reduced set has cardinality $n/3^i$. The best case occurs when the root level comparison results in inequality. The subsequent comparisons lead to a leaf node where only OCPH() and OCPL() are applied on two different sets and each of them takes O(log n) time. OCPH() or OCPL() can be applied at one of the leaves that takes at most three comparisons from the root vertex which is a constant. Hence the best case complexity is O(log n). On the other hand, if the root level comparison results in equality, all the coin sets are suspected to contain the false coin pairs. Thus, the sample space to be further considered is not reduced. In our algorithm, TCP(K_i), TCP(K_i+1), and TCP(K_i+2) are applied sequentially only when the first comparison shows

equality in each recursion of TCP(). As at the beginning we divide the initial set into K_1, K_2, and K_3, we call TCP(K_2) if and only if TCP(K_1) reports 'no false coin', and we call TCP(K_3) if and only if both K_1 and K_2 contain no false coin.

Now, if we reach to a set with only 5, or 4, or 3 coins, then we may conclude whether the corresponding set contains either both the false coins or none of them. It takes at most four comparisons. Thus, in the worst case to find the two false coins we have to apply TCP() on K_1, K_2, and K_3 that requires O(n) time. We assume that the algorithm uses only equal number of coins on both the pans.

The case $\Delta\omega(H) = \Delta\omega(L)$ leads to a confusion as the sum of the weights of two false coins of that Kind results in the weight-sum of two correct coins. If there are n coins in a set, there are at most $2\times{}^{n}C_2$ combinations of the two false coins' position in any indexed sequence of n coins. Now at any instant of the algorithm, there might have equal number of coins shared by two pans of the equal arm balance, and hence both the false coins may reside in any one of the pans.

The purpose of this algorithm is to verify two sets of coins together and depending on the weighing it proceeds to the next step by reducing the number of coins in each set next to be considered for further checking. However, if both the false coins are always in one set, i.e. in one pan, the checking of the sets results in equality. If we divide the given set of coins into $n/2$ subsets and weigh those pairwise, we may get all these sets equal in weight. Thus, the weighing process may result in equality until we reach a case where a set of coins under concern has only one false coin that certainly results in inequality; here the false coins may share the pans or at least one of them must be in one pan. Hence, the sample space cannot be reduced below a factor of n. Therefore, running time of the algorithm is linear.

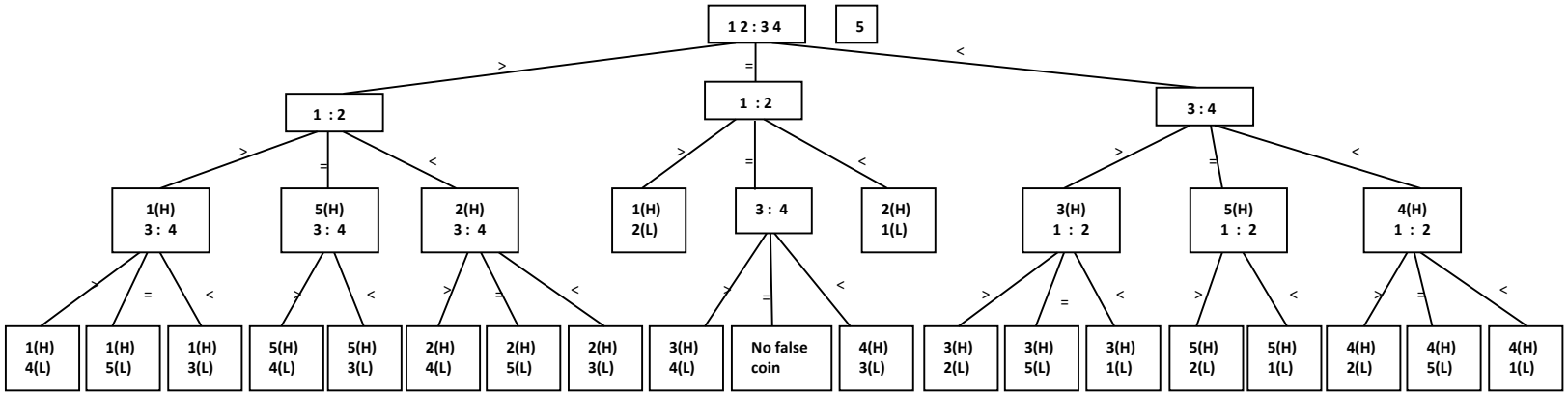


Figure 6.4: Decision tree for finding two false coins given $\Delta(\omega(H)) = \Delta(\omega(L))$ among five coins.

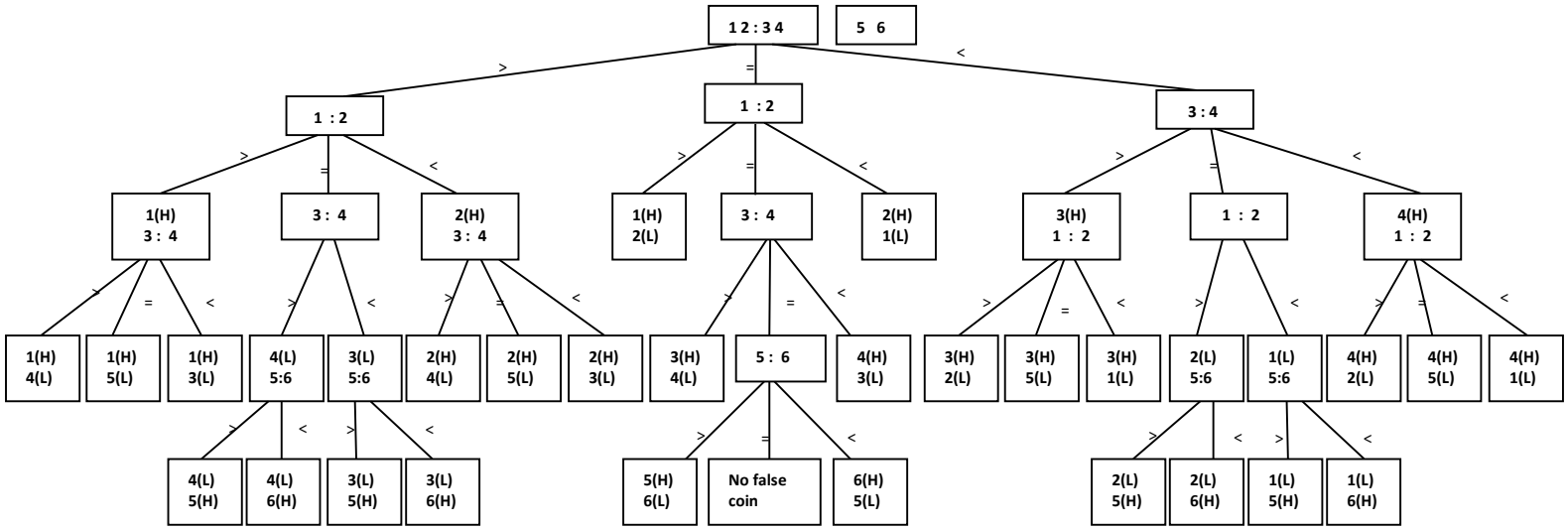


Figure 6.5: Decision tree for finding two false coins given $\Delta(\omega(H)) = \Delta(\omega(L))$ among six coins.

$\Delta(\omega(H)) = \Delta(\omega(L))$

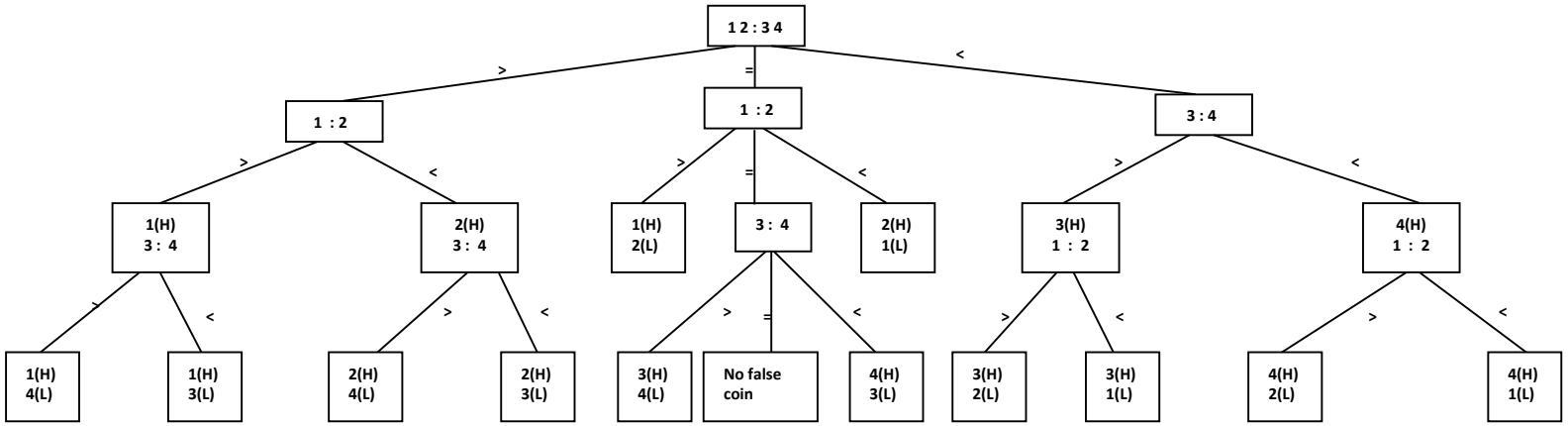


Figure 6.6: Decision tree for finding two false coins given $\Delta(\omega(H)) = \Delta(\omega(L))$ among four coins.

6.5 Experimental Results of the Algorithm

Experimental results of the above discussed algorithm are shown in tabular form (see Table 6.1) as well as in graphical form (see Figure 6.7). We have considered the best possible number of inputs. At first we consider an input size of ten coins; the total number of comparisons is 483, the possible number of false coin combinations is 90, and average number of comparisons is 5. We plot the number of coins along x-axis and the average number of comparisons along y-axis for all the numbers of coins mentioned in the given table.

Table 6.1: Average number of comparisons for different values of n considering one heavier and one lighter coin with $\omega(H) - \omega(T) = \omega(T) - \omega(L)$ (i.e. $\Delta(\omega(H)) = \Delta(\omega(L))$).

Number of coins (n)	Total number of comparisons (S)	Possible number of false coin combinations ($C = 2\times {}^{n}C_2$)	Average number of comparisons (AVG = S/C)
10	483	90	5
18	2064	306	6
26	5130	650	7
45	26436	1980	13
64	47468	4032	11
72	64704	5112	12
91	166586	8190	20
127	492976	16002	30
161	549668	25760	21
181	717268	32580	22
198	872526	39402	22
200	884954	39800	22

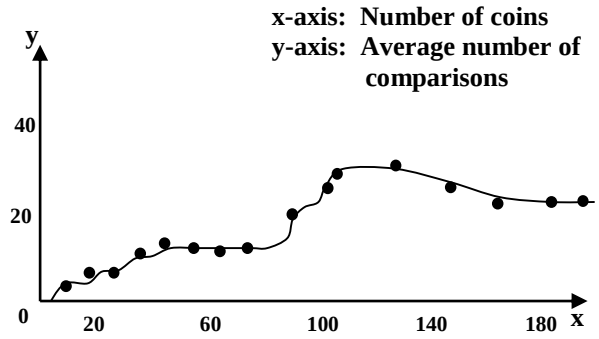


Figure 6.7: Graph with the average number of comparisons along y-axis and the total number of coins along x-axis considering the case of $\Delta(\omega(H)) = \Delta(\omega(L))$.

6.6 Summary

The scenario considered in this chapter about the occurrence of two counterfeit coins is the most critical one. We have devised algorithm to identify the false coins effectively where in the worst case we may have to apply TCP() linear number of times with the input size. To summarize, this can be stated that, given a set of *n* coins of identical in appearance and anomaly information $\omega(H) - \omega(T) = \omega(T) - \omega(L)$ (i.e. $\Delta(\omega(H)) = \Delta(\omega(L))$), algorithm TCP() performs a series of weighing forming a decision tree and finds out the false coin pairs in $O(n)$ time.

Chapter 7

Conclusion

In this chapter, we are going to discuss various applications of counterfeit coins problem and contributions made in this thesis to solve the problem along with the future scopes to deal with in this area of research. Section 7.1 describes various application perspectives that can be inferred from the presented work. In this thesis, we have devised a number of algorithms to solve single and two-counterfeit coins problem along with the evidences of the versatility of the combinatorial reasoning that has been summarized in Section 7.2. In Section 7.3, future scopes of the domain have been discussed.

7.1 Application Perspectives

Several fields of applications are hereby envisaged to gear up with help of algorithms developed to obtain the counterfeit coin solutions. Being a complex search problem in combinatory, counterfeit coins problem has all-round applications in real life problems.

7.1.1 Hidden Graph Learning and Counterfeit Coins

The problems conferred in the thesis accomplish various distinctive areas of research, the most important of which is graph learning [69]. In case of a graph learning problem, a hidden graph is known to belong to a given family of labeled graphs on some predefined vertex set. With reference to the information, the objective is to identify the graph under consideration by edge-detecting queries. Each query tells whether a subset of the vertices induces an edge of the graph. This problem is motivated by applications in DNA physical mapping [70]. Applications of this model extend to bioinformatics, where learning a hidden matching [70] is of extreme importance in DNA sequencing.

In resemblance, if the set of coins is assumed to be the vertices of a graph and there lies an edge between two vertices (say, v_i and v_j), if and only if the coins representing v_i and v_j are equal in weight and each edge weight represents that weight. Hence, in this context, all the true coins in the search space always form a complete graph whereas the

false coins are disconnected from the true coins forming either isolated vertices or connected among themselves depending on their weights, i.e. those are isolated vertices if their weights are different, otherwise, there may be edges among the false coins if all or some of them are of equal weight. Thus, this problem now reduces to graph construction problem or finding hidden graph problem as we have no information of the weights of the coins initially. As all the coins are identical in their appearance and we do not have their weights, we may assume that all the *n* nodes form a complete graph as shown in Figure 7.1(a), i.e. we assume all the coins are of equal weight or true coins. At this point, the graph reconstruction problem is to find out the false edges and accordingly remove those to obtain the preferred graph. In Figures 7.1(b) and 7.1(c), we have depicted the obtained graphs in case of two counterfeit coins problem. In case of Figure 7.1 (b), the counterfeit coins are mutually different in weight, whereas Figure 7.1(c) cites the scenario of equally weighed two false coins.

As reduction of the number of weighings to recognize false coins is a prime challenge, proper subsets of coins must be selected for subsequent weighing that would lead to the conclusion efficiently. Moreover, in case of graph learning, less the number of queries less is the cost of construction of the graph. Thus, query minimization is a key challenge here that is basically analogous to the comparison minimization of the counterfeit coins problem.

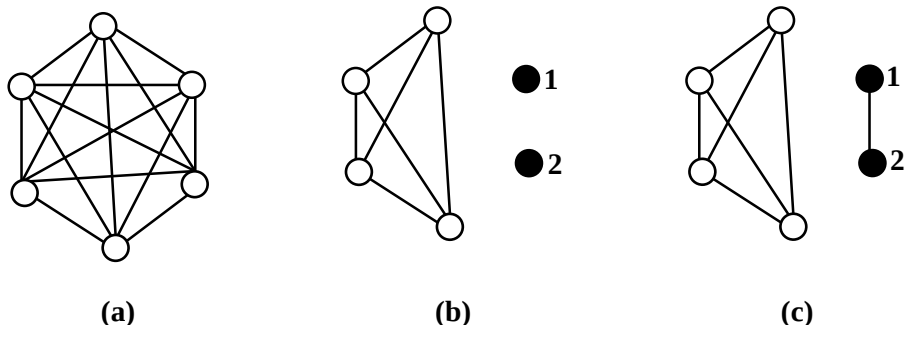


Figure 7.1: (a) 6 coins or nodes with identical appearance form a complete graph**. (b)** Nodes 1 and 2 are false nodes with mutually different weight other than the standard (or correct) weight. **(c)** Nodes 1 and 2 are false having mutually equal weight.

Learning a hidden general graph can be viewed as a variant of group testing that is a well-known combinatorial search problem [17]. Given a set of items, each of which is either positive or negative, a group test on a subset of items determines whether it contains any positive item. The key task of conventional group testing problem is to distinguish positive items by group tests. An extension of group testing is the complex model, where a set of complexes, each of which is a subset of fundamental items, is given and the property (positive or negative) of each complex is not yet determined; the corresponding query to identify positive complexes answers whether a subset of basic items contains at least one positive complex.

In a different background [17], identifying which pairs of chemicals react in a solution is modeled by counterfeit detection queries. Here, the coins, i.e. the vertices correspond to the chemicals, edges assign chemical reactions, and a set of chemicals 'reacts' if and only if, it induces an edge.

7.1.2 Electrical Circuit Analysis and Counterfeit Coins

As counterfeit coins problems possess analogy with graph learning, its application appears in many different contexts. Suppose, for a given circuit containing a set of chips on a board, we must test the resistance between two chips with an ammeter. In as few measurements as possible, our objective is to learn whether the circuit is connected entirely, or whether we need to provide power to the components individually [17]. This can be seen as a counterfeit coins problem, in which the chips are assumed to be the coins and the ammeter measurements are queries like weighing with a scale, which tell whether a set of coins are of same weight, i.e. they are connected. If we are provided an efficient enough ammeter to tell not only whether two chips are connected, but also how far apart they lie in the underlying circuit, we obtain the stronger 'shortest path' queries.

7.1.3 Consumer Products

The word *counterfeit* most frequently describes forgeries of currency or documents, but can also describe software, pharmaceuticals, etc. Counterfeit antique coins are generally made to a very high standard so that they often fool collectors. This is not easy, and many coins still stand out. It is not only limited to coins but potentially used global trade market, archaeological departments and many others. The raising issue of counterfeits violates

intellectual property right and causing damage to both producer and consumer. To identify the counterfeit goods like pirated electronic gadgets, counterfeit batteries used in a digital camera, pharmaceuticals, valuable ornaments solution of counterfeit problem are used.

Counterfeiting of original versions of manufactured products is often observed to be a serious case of fraudulent activities in different ages of industrialization. Proposed algorithms, upon suitable modifications, can be useful to detect the originality of consumer product [65].

Moreover, Machine Learning aspects along with regression statistical techniques are the most promising attribute that can be added on top of counterfeit algorithms to materialize the product detection facilities more vibrant and stable [67]. A case study can be assimilated at this point such as follows. Suppose *X* is a famous women bag producer company. *Y* is another fraud company that produces bags which are very similar to the *X*. A customer goes to market with a smart hand-held device and places the infra-red emitter side of the device on the surface of the bag. The microscopic image as seen by the image detector is instantaneously checked with the pre-loaded counterfeit enabled application. Upon completion of a certain pattern matching formulation on the received image of bag surface is immediately cross-checked with the counterfeit values. Thus, the customer gets assurance about the originality or the manufactured product.

7.1.4 Drug Check

Medicine in form of drugs is indispensable part of human life to get rid of illness [66]. If medicine is going to be distributed in form of swindle format, then it must be dangerous for the patient who is taking it. Counterfeit algorithms are there to solve this type of problem with very effective way. Usually what happens is that the chemical behaviour of some organic compounds such as methanol, ethanol etc. make a practitioner puzzle to detect which substance it is in current form. Thus, medicines that are based on such chemicals are easy to be wrongly picked up than other forms of chemicals.

We may visualize a case study where this type of discrepancy can be sorted out. Suppose *A* is a drug dispenser who dispenses drug to the required unit of medicine design. *A* gets two sets of samples of look-alike chemical products. *A* is in confusion state to which to select for what design chain. Fortunately, *A* is equipped with the advanced Raman-

spectroscopy machine that runs on the counterfeit engine. *A* targets the machine to the sample, the photo-metric values are then easily gets sorted with intervention from counterfeit-engine and the correct set of chemical is momentarily processed with accurate estimation.

7.1.5 Bill Detection

Nowadays, the printer-scanner technology has become too much advanced that can be easily used to recreate some forgery on the original bills. Such problem can be solved with the valuable interventions from the proposed counterfeit algorithms. A scenario may be opted to better understand the case as follows. Suppose an officer performs his duty toward checking of originality of the bills that are submitted to the office for appropriate procurement. This issue can be solved by using counterfeit techniques in prescribed manner. A deep learning model can be developed that inherits the convolution neural networks by comprising rectified as well as max-pooling linear units into the system. The learning network should be fed with the samples from different printer-scanner models. The extraction should be done based a prescribed set of features. The features will then be utilized to accurately predict as well as detection the faulty bills, thus reducing the opportunity of manual monetary loss.

7.1.6 Quantum Query Detection

Quantum computing can be seen as the next big thing in computing paradigm after the Boolean-conceptualization [64]. Researchers are in trouble to answer the question of quantum query complexity for finding k faulty coins, which is responsible to identify the actual input x. This problem can be seamlessly formatted with proper orientation from the balanced-oracle and inner product oracle algorithms. Developed counterfeit algorithms are well framed to realize the effectiveness of the quantum query complexity detection. But, it is known fact that the balanced-oracle algorithm can be simulated to the query by the inner-product algorithm by $O(k^{1/4})$. To do so, an appropriate stochastic lower bound under the random partition assumption can be validated against the $O(k^{1/4})$ complexity. Pan-wise coin distribution could be done in randomized trails.

As counterfeit coins problem initially belongs to combinatorial group testing problem, beside the aforementioned fields, it can be mapped into utilization in medical

field like finding of any odd spike in MRI scan or in technical field to find any set of damaged pixels in a digital image.

7.2 Contributions

This section presents the key contributions made in this research work. Several new algorithms are designed, developed and simulated in this course of action. In this work, we have focused on single and two counterfeit coins problem. First, we have considered the eight coins problem, one that is well-known in the literature. Only one out of eight coins is a false coin, which is either heavier or lighter than a true coin. The usual objective of this problem is to find the false coin using a minimum number of comparisons. We have developed two new solutions for the eight coins problem, and they are as good as or better than the existing classical solution. Moreover, we have generalized it for all values of n, where n is an even number. We categorize the search space in two ways, ones which are powers of two, and the other is the rest of the even numbers. After developing the algorithm for solving the Counterfeit Coin Problem where the number of coins is even, we widen our algorithm for handling the situation of an odd number of coins problem. Thus, the algorithm is generalized.

Next, we have considered the two counterfeit coins problem. The common objective of this problem is to find two false coins among a set of identical coins using a minimum number of comparisons. As there are several variations for two counterfeit coins depending on the mutual relation of the counterfeit coins, we have first discussed all the variations in detail and devised algorithms for solving the problem considering all the cases individually. Here we have followed decision tree method. The nobility of these two algorithms is that the worst case running time of the algorithms are O(log n), except a typical one, i.e. when one false coin is heavier and the other is lighter and the weight difference between the heaver and a true coin is same as the difference of the lighter one and a true coin, the worst case run time becomes linear with respect to the cardinality of the search space. We have considered all the possible cases for any number of coins, i.e. the search space can be a set of coins of no restricted cardinality. Thus, we have generalized the algorithms to make it applicable for all the variations of two counterfeit coins problem. We established a clear picture of two counterfeit coins problem without any ambiguity.

7.3 Future Direction

This research is all about the design and development of several novel counterfeit algorithms to minimize the risk of fraudulent cases in various scene-specific aspects. This research has covered all variants of single and double counterfeit coins problem. As we have observed that the number of variants increases exponentially with the increase in number of false coins; hence the degree of fraudulence becomes very high. For an example, when the number of false coin is three, there can be twenty six different scenarios to take care of. We have investigated all the cases for three counterfeit coins in detail [26] and developed an algorithm to solve a few of those. Still there are several critical scenarios to be resolved.

Moreover, in some cases where volumes or density of any liquid is attackable for forgery, the scenario becomes much more difficult. In that case, we need to incorporate improved graph learning mechanisms to deal with the fact. Thus, the generalized counterfeit identification problem becomes a crucial domain of research. However, the study can be further engraved into interdisciplinary domain of information communication technology some of which have been cited as follows.

Internet of Counterfeit Things [66]: Internet of Things is the most recent bud of smart computing hierarchy as seen by the Gartner periodicals. Unlimited number of "*things*" are envisaged to get co-existed in near future all over the world. Node-oriented fraudulences can be minimized with proper incorporation with the developed counterfeit algorithms.

Big Counterfeit Data Analytics [68]: When there is lots of different data coming into internetwork in various speed and form factors, big data analytics becomes equally crucial. However, the requirement of analytics on the "big counterfeit data" can be taken as a next step of meaningfulness schematic for bringing data-independence in current ear of computing.

Deep Counterfeit Learning [67]: Machine Learning is key to artificially organize any relevant perspectives. Existing machine learning techniques should be intervened with the new counterfeit strategies so that a novel "counterfeit learning" theme could be

visualized. Deep neural networking may act as the key enabler of assimilation of such notion in near future.

We strongly believe that this list will grow in the future, as more researchers involve themselves towards applying the counterfeit coins problem as a tool to solve the problems in their relevant research areas.

References

[1] Guy, Richard K., and Richard J. Nowakowski. “Coin-weighing problems.” *The American Mathematical Monthly* 102, no. 2 (1995): 164-167.

[2] Li, Anping. “On the conjecture at two counterfeit coins.” *Discrete Mathematics* 133, no. 1-3 (1994): 301-306.

[3] Halbeisen, Lorenz, and Norbert Hungerbühler. “The general counterfeit coin problem.” *Discrete Mathematics* 147, no. 1-3 (1995): 139-150.

[4] Ghosh, Joydeb, Papiya Senmajumdar, Srijoni Maitra, Debasis Dhal, and Rajat Kumar Pal. “A generalized algorithm for solving *n* coins problem.” In *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on*, vol. 2, pp. 411-415. IEEE, 2011.

[5] Ghosh, J., P. Senmajumdar, S. Maitra, D. Dhal, and R. K. Pal. “Yet another algorithm for solving *n* coins problem.” *Assam University Journal of Science & Technology: Physical Sciences and Technology* 8, no. II (2011): 118-125.

[6] Ghosh, Joydeb, Piyali Datta, Arpan Chakraborty, Ankita Nandy, Lagnashree Dey, Rajat Kumar Pal, and Ranjit Kumar Samanta. “An endeavour to find two unequal false coins.” In *Electrical and Computer Engineering (ICECE), 2014 International Conference on*, pp. 333-336. IEEE, 2014.

[7] Ghosh, Joydeb, Arpan Chakraborty, Piyali Datta, Lagnashree Dey, Ankita Nandy, Rajat Kumar Pal, and Ranjit Kumar Samanta. “The first algorithm for solving two coins counterfeiting with $\omega(\Delta H) = \omega(\Delta L)$.” In *Electrical and Computer Engineering (ICECE), 2014 International Conference on*, pp. 337-340. IEEE, 2014.

[8] Ghosh, Joydeb, Ankita Nandy, Lagnashree Dey, Piyali Datta, Arpan Chakraborty, Rajat Kumar Pal, and Ranjit Kumar Samanta. “An algorithm for identifying two unequal heavier/lighter coins out of *n* given coins.” In *Computer, Communication,*

Control and Information Technology (C3IT), 2015 Third International Conference on, pp. 1-6. IEEE, 2015.

[9] Ghosh, Joydeb, Lagnashree Dey, Ankita Nandy, A. Chakraborty, Piyali Datta, Rajat Kumar Pal, and Ranjit Kumar Samanta. "An advanced approach to solve two counterfeit coins problem." *Proc. Annals of Pure Applied Mathematics* 7, no. 1 (2014): 77-82.

[10] Tošić, Ratko. "Two counterfeit coins." *Discrete Mathematics* 46, no. 3 (1983): 295-298.

[11] Manvel, Bennet. "Counterfeit coin problems." *Mathematics Magazine* 50, no. 2 (1977): 90-92.

[12] Bellman, Richard, and Brian Gluss. "On various versions of the defective coin problem." *Information and Control* 4, no. 2-3 (1961): 118-131.

[13] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. PHI Learning Pvt. Ltd., 3rd edition, (2011): 1106-1128.

[14] Choi, Sung-Soon, and Jeong Han Kim. "Optimal query complexity bounds for finding graphs." *Artificial Intelligence* 174, no. 9-10 (2010): 551-569.

[15] Choi, Sung-Soon. "Polynomial time optimal query algorithms for finding graphs with arbitrary real weights." In *Conference on Learning Theory*, pp. 797-818. 2013.

[16] Uehara, Ryuhei, Kensei Tsuchida, and Ingo Wegener. "Identification of partial disjunction, parity, and threshold functions." *Theoretical Computer Science* 230, no. 1-2 (2000): 131-147.

[17] Du, Dingzhu, Frank K. Hwang, and Frank Hwang. *Combinatorial group testing and its applications*. Vol. 12. World Scientific, 2000.

[18] Lim, Eldin Wee Chuan. "On Anomaly Identification and the Counterfeit Coin Problem." *arXiv preprint arXiv:0905.0085*(2009).

[19] http://www.numericana.com/answer/weighing.htm#counterfeit.

[20] http://www.geeksforgeeks.org/decision-trees-fake-coin-puzzle.

[21] Aigner, Martin, and Anping Li. "Searching for counterfeit coins." *Graphs and Combinatorics* 13, no. 1 (1997): 9-20.

[22] Levitin, Anany, and Maria Levitin. *Algorithmic puzzles*. OUP USA, 2011.

[23] Aigner, Martin. *Combinatorial search*. BG Teubner, 1988.

[24] Ghosh, Joydeb, S. K. Ghosh, and Rajat Kumar Pal. "Two new solutions of the eight coins problem." In *Computer, Communication, Control and Information Technology (C3IT), 2009 First International Conference on*, pp. 85-92, 2009.

[25] Ghosh, J., S. K. Ghosh, and Rajat Kumar Pal. "A Revisit to the Eight Coins Problem." *International Journal of Computing and Information Technology (IICIT)* 2, no. 1 (2010): 1-14.

[26] Chakraborty, Arpan, Joydeb Ghosh, Piyali Datta, Ankita Nandy, and Rajat Kumar Pal. "Anomaly Detection and Three Anomalous Coins Problem." In *Advanced Computing and Systems for Security*, pp. 303-320. Springer, New Delhi, 2016.

[27] Alon, Noga, and Dmitry N. Kozlov. "Coins with arbitrary weights." *Journal of Algorithms* 25, no. 1 (1997): 162-176.

[28] Alon, Noga, and Văn H. Vũ. "Anti-Hadamard matrices, coin weighing, threshold gates, and indecomposable hypergraphs." *Journal of Combinatorial Theory, Series A* 79, no. 1 (1997): 133-160.

[29] Alon, Noga, Dmitry N. Kozlov, and Van H. Vu. "The geometry of coin-weighing problems." In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pp. 524-532. IEEE, 1996.

[30] Dyson, Freeman J. "1931. The Problem of the Pennies." *The Mathematical Gazette* 30, no. 291 (1946): 231-234.

[31] Bošnjak, Ivica. “Some new results concerning three counterfeit coins problem.” *Discrete Applied Mathematics* 48, no. 1 (1994): 81-85.

[32] De Bonis, Annalisa. “A predetermined algorithm for detecting a counterfeit coin with a multi-arms balance.” *Discrete Applied Mathematics* 86, no. 2-3 (1998): 181-200.

[33] De Bonis, Annalisa, Luisa Gargano, and Ugo Vaccaro. “Optimal detection of a counterfeit coin with multi-arms balances.” *Discrete Applied Mathematics* 61, no. 2 (1995): 121-131.

[34] Newbery, E. V. *The penny problem*. Vol. 130. Note 2342, Math. Gaz., 37, 1953.

[35] Schwartz, Benjamin L. “Letter: Truth about false coins.” *Math. Mag* 51 (1978): 254.

[36] Graham, Louis A. *Ingenious mathematical problems and methods*. Vol. 545. Courier Corporation, 1959.

[37] Schell, E. D. “Problem E651—Weighed and found wanting.” *Amer. Math. Monthly* 52 (1945): 42.

[38] Du, Ding-Zhu, and Frank K. Hwang. “Competitive group testing.” *Discrete Applied Mathematics* 45, no. 3 (1993): 221-232.

[39] Gargano, Luisa, János Körner, and Ugo Vaccaro. “Search problems for two irregular coins with incomplete feedback: the underweight model.” *Discrete Applied Mathematics* 36, no. 2 (1992): 191-197.

[40] Graham, R. L., and N. J. A. Sloane. “Anti-Hadamard matrices.” *Linear algebra and its applications* 62 (1984): 113-137.

[41] Halbeisen, Lorenz, and Norbert Hungerbühler. “The general counterfeit coin problem.” *Discrete Mathematics* 147, no. 1-3 (1995): 139-150.

[42] Hu, Xiao-Dong, P. D. Chen, and Frank K. Hwang. “A new competitive algorithm for the counterfeit coin problem.” *Information Processing Letters* 51, no. 4 (1994): 213-218.

[43] Hu, X. D., and F. K. Hwang. “A competitive algorithm for the counterfeit coin problem.” In *Minimax and Applications*, pp. 241-250. Springer, Boston, MA, 1995.

[44] Kozlov, Dmitry N., and Van H. Vu. “Coins and cones.” *journal of combinatorial theory, Series A* 78, no. 1 (1997): 1-14.

[45] Li, Anping. “Three counterfeit coins problem.” *Journal of Combinatorial Theory, Series A* 66, no. 1 (1994): 93-101.

[46] Linial, Nathan, and Michael Tarsi. “The counterfeit coin problem revisited.” *SIAM Journal on Computing* 11, no. 3 (1982): 409-415.

[47] Manas, G. J., and D. H. Meyer. “On a problem of coin identification.” *SIAM Review* 31, no. 1 (1989): 114-117.

[48] Pyber, László. “How to find many counterfeit coins?” *Graphs and Combinatorics* 2, no. 1 (1986): 173-177.

[49] Schrijver, Alexander. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[50] Wan, Peng-Jun, and Ding-Zhu Du. “A ($\log_2 3$+ 12) competitive algorithm for the counterfeit coin problem.” *Discrete Mathematics* 163, no. 1-3 (1997): 173-200.

[51] Wan, Peng-Jun, Qifan Yang, and Dean Kelley. “A (3/2)log 3-competitive algorithm for the counterfeit coin problem.” *Theoretical computer science* 181, no. 2 (1997): 347-356.

[52] http://www.coinauthentication.co.uk/newsletter10.html

[53] http://coinsguide.reidgold.com/counterfeits.html

[54] Chang, Gerard J., and Frank K. Hwang. “A group testing problem.” *SIAM Journal on Algebraic Discrete Methods* 1, no. 1 (1980): 21-24.

[55] Chang, Gerard J., and Frank K. Hwang. “A group testing problem on two disjoint sets.” *SIAM Journal on Algebraic Discrete Methods* 2, no. 1 (1981): 35-38.

[56] Chang, Gerard J., Frank K. Hwang, and Shen Lin. “Group testing with two defectives.” *Discrete Applied Mathematics* 4, no. 2 (1982): 97-102.

[57] Moon, John W., and M. Sobel. “Enumerating a class of nested group testing procedures.” *Journal of combinatorial theory, series B* 23, no. 2-3 (1977): 184-188.

[58] Riccio, Laura, and Charles J. Colbourn. “Sharper bounds in adaptive group testing.” *Taiwanese Journal of Mathematics* 4, no. 4 (2000): 669-673.

[59] Sobel, Milton, and Phyllis A. Groll. “Group testing to eliminate efficiently all defectives in a binomial sample.” *Bell Labs Technical Journal* 38, no. 5 (1959): 1179-1252.

[60] Hwang, F. K. “A method for detecting all defective members in a population by group testing.” *Journal of the American Statistical Association* 67, no. 339 (1972): 605-608.

[61] Li, Chou Hsiung. “A sequential method for screening experimental variables.” *Journal of the American Statistical Association* 57, no. 298 (1962): 455-477.

[62] Dorfman, Robert. “The detection of defective members of large populations.” *The Annals of Mathematical Statistics* 14, no. 4 (1943): 436-440.

[63] Hwang, F. K. “A minimax procedure on group testing problems.” *Tamkang J. Math* 2 (1971): 39-44.

[64] Iwama, Kazuo, Harumichi Nishimura, Rudy Raymond, and Junichi Teruyama. “Quantum counterfeit coin problems.” *Theoretical Computer Science* 456 (2012): 51-64.

[65] Tom, Gail, Barbara Garibaldi, Yvette Zeng, and Julie Pilcher. “Consumer demand for counterfeit goods.” *Psychology & Marketing* 15, no. 5 (1998): 405-421.

[66] Blackstone, Erwin A., Joseph P. Fuhr Jr, and Steve Pociask. “The health and economic effects of counterfeit drugs.” *American Health & Drug Benefits* 7, no. 4 (2014): 216.

[67] Zeinab, Kamal Aldein Mohammed, and Sayed Ali Ahmed Elmustafa. “Internet of Things applications, challenges and related future technologies.” *World Scientific News* 2, no. 67 (2017): 126-148.

[68] Elgendy, Nada, and Ahmed Elragal. “Big data analytics: a literature review paper.” In *Industrial Conference on Data Mining*, pp. 214-227. Springer, Cham, 2014.

[69] Chang, Huilan, Hung-Lin Fu, and Chih-Huai Shih. “Learning a hidden graph.” *Optimization Letters* 8, no. 8 (2014): 2341-2348.

[70] Bouvel, Mathilde, Vladimir Grebinski, and Gregory Kucherov. “Combinatorial search on graphs motivated by bioinformatics applications: A brief survey.” In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pp. 16-27. Springer, Berlin, Heidelberg, 2005.

[71] Horowitz, Ellis, Sartaj Sahni, and Susan Anderson-Freed. *Fundamentals of data structures*. London: Pitman, 1976.

Appendix A

List of Publications

[1] **Joydeb Ghosh** (with S. K. Ghosh and R. K. Pal). A new algorithm to represent a given *k*-ary tree into its equivalent binary tree structure. *Journal of Physical Sciences*, Vol. 12, pp. 253-264, Dec. 2008.

[2] **Joydeb Ghosh** (with S. K. Ghosh and R. K. Pal). An algorithm for converting a given *k*-ary tree into its equivalent binary tree. *Proc. of the First International Conference on Computer, Communication, Control and Information Technology* (*C3IT 2009*), pp. 56-62, 2009.

[3] **Joydeb Ghosh** (with S. K. Ghosh and R. K. Pal). Two new solutions of the eight coins problem. *Proc. of the First International Conference on Computer, Communication, Control and Information Technology* (*C3IT 2009*), pp. 85-92, 2009.

[4] **Joydeb Ghosh** (with S. K. Ghosh and R. K. Pal). A revisit to the eight coins problem. *International Journal of Computing and Information Technology*, vol. 2, no. 1, pp. 1-14, 2010.

[5] **Joydeb Ghosh** (with D. Dhal and R. K. Pal). A generalized algorithm for solving n coins problem. *Proc. of the IEEE International Conference on Computer Science and Automation Engineering* (*CSAE 2011*), Shanghai, China, vol. 2, pp. 411-415, 2011.

[6] **Joydeb Ghosh** (with P. Senmajumdar, S. Maitra, D. Dhal, and R. K. Pal). Yet another algorithm for solving *n* coins problem. *Assam University Journal of Science & Technology: Physical Sciences and Technology*, vol. 8, no. 2, pp. 118-125, May 2011.

[7] **Joydeb Ghosh** (with L. Dey, A. Nandy, A. Chakraborty, P. Datta, R. K. Pal, and R. K. Samanta). An advanced approach to solve two counterfeit coins problem. *Annals of Pure and Applied Mathematics*, vol. 7, no. 1, pp. 77-82, 2014.

[8] **Joydeb Ghosh** (with P. Datta, A. Chakraborty, A. Nandy, L. Dey, R. K. Pal, and R. K. Samanta). An endeavour to find two unequal false coins. *Proc. of the Eighth International Conference on Electrical and Computer Engineering* (*ICECE 2014*), Dhaka, Bangladesh, pp. 333-336, 2014.

[9] **Joydeb Ghosh** (with A. Chakraborty, P. Datta, L. Dey, A. Nandy, R. K. Pal, and R. K. Samanta). The first algorithm for solving two coins counterfeiting with $\omega(\Delta H) = \omega(\Delta L)$, *Proc. of the Eighth International Conference on Electrical and Computer Engineering* (*ICECE 2014*), Dhaka, Bangladesh, pp. 337-340, 2014.

[10] **Joydeb Ghosh** (with A. Chakraborty, P. Datta, L. Dey, A. Nandy, R. K. Pal, and R. K. Samanta). An algorithm for identifying two unequal heavier/lighter coins out of n given coins. *Proc. of the Third International Conference on Computer, Communication, Control and Information Technology* (*C3IT 2015*), pp. 1-6, IEEE, 2015.

[11] **Joydeb Ghosh** (with A. Chakraborty, P. Datta, A. Nandy, and R. K. Pal). Anomaly detection and three anomalous coins problem. *Advanced Computing and Systems for Security*, University of Calcutta, Kolkata, pp. 303-320, Springer India, 2015.

[12] **Joydeb Ghosh** (with P. Datta, A. Chakraborty, R. K. Pal, and R. K. Samanta). On Variations of Two Anomalous Coins Problem. *Manuscript*, 2018.

A Generalized Algorithm for Solving *n* Coins Problem

Joydeb Ghosh[1], Papiya Senmajumdar[2], Srijoni Maitra[2], Debasis Dhal[3], and Rajat Kumar Pal[4]

[1]Department of Mathematics, Surendra Institute of Engineering and Management, New Chamta, Siliguri, Darjeeling – 734 009, West Bengal, India
[2]Department of Computer Science and Engineering, University of Calcutta, 92 A P C Road, Kolkata – 700 009, West Bengal, India
[3] Dum Dum Subhasnagar High School (HS), 43 S B Road, Rabindranagar, 24 Parganas (North), Kolkata – 700 065, West Bengal, India
[4]Department of Information Technology, School of Technology, Assam University, Silchar, Cachar – 788 011, Assam, India

Abstract— **Eight coins problem is a well-known problem in mathematics as well as in computer science. In this problem eight coins are given, say A, B, C, D, E, F, G, and H, and we are told that only one is counterfeit (or false), as it has a different weight than each of the others. We want to determine which coin it is, making use of an equal arm balance. At the same time we want to identify the counterfeit coin using a minimum number of comparisons and determine whether the false coin is heavier or lighter than each of the remaining.**

In this paper, we develop algorithms for solving the counterfeit coin problem for any given number *n* of coins. The first algorithm is in essence based on the existing classical solution for the eight coins problem (with slight modification) for larger values of *n*, where *n* is a power of two beyond eight, as two and four being base cases. Then we develop an algorithm for solving *n* coins problem, where *n* is even but not power of two, i.e., the numbers are six, ten, 12, 14, 18, 20, etc. At the end, we have extended the same to solve the counterfeit coin problem for odd number of coins as well.

Keywords— Coin counterfeiting; Eight coins problem; Even coin problem; Odd coin problem; Comparison; Equal arm balance; Decision tree; Algorithm.

I. INTRODUCTION

The *eight coins problem* [3, 5, 6] is a well-known problem in mathematics as well as in computer science. In this problem eight coins are given, say A, B, C, D, E, F, G, and H where one is a counterfeit (or false) coin, as it has a different weight than each of the others. We like to determine the counterfeit coin with the help of an equal arm balance. At the same time we like to arrive at the minimum number of comparisons required to determine whether the counterfeit coin is heavier or lighter than each of the remaining coins.

II. LITERATURE SURVEY

Figure 1 shows the only existing classical solution of eight coins problem in literature [3] up to a few years back. The solution is available in the form of a *decision tree*. The tree in this figure represents a set of decisions by which we can get the solution(s) of our problem. This is why it is called a decision tree. We use lower-case *h* or *l* as suffixes to represent the coin as *heavier* or *lighter*, respectively.

In the solution of the eight coins problem in the form of a decision tree, each internal vertex (i.e., other than leaf vertices) symbolizes a comparison between a pair of sets of coins using an equal arm balance. Needless to mention that in each of these comparisons both the sets contain equal number of coins. This tree (in Figure 1) starts with a vertex where three coins are kept on either side of the equal arm balance.

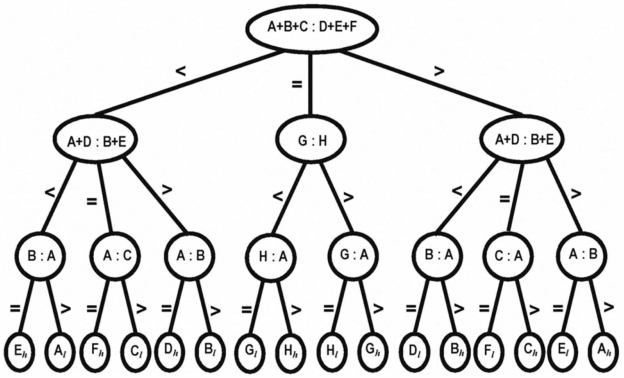


Figure 1. The existing solution of the eight coins problem in the form of a decision tree [3].

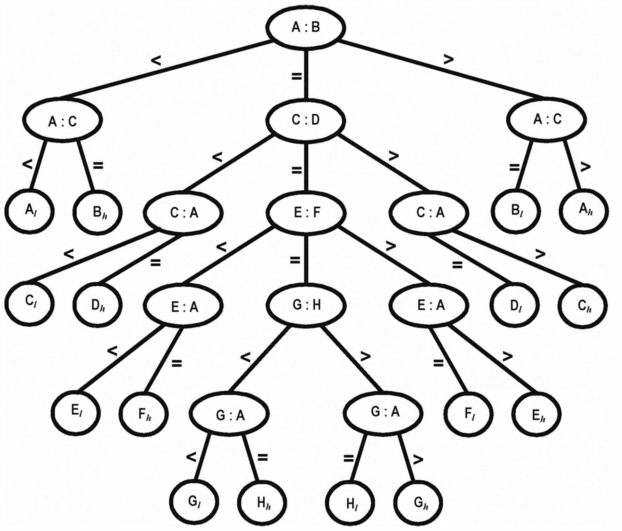


Figure 2. A naive solution of the eight coins problem that has been solved with the help of a decision tree.

If we consider all the eight coins at a time to distribute them into two sets of four coins each to be compared, then it is an useless comparison as one coin out of eight coins is given as counterfeit (or false). So we cannot start with all the coins at the same time to be compared for finding out the false coin as it leads to a redundant comparison. The solution in Figure 1 is self-explanatory.

On the other hand, we may follow a naive way of finding the false coin and consider only pairs of coins as shown in

978-1-4244-8728-8/11/$26.00 ©2011 IEEE

Figure 2, which is not a desired solution of the eight coins problem as the height of the tree has increased.

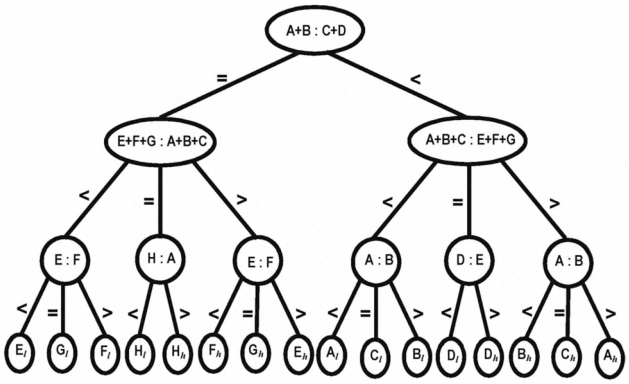


Figure 3. The first new solution of the eight coins problem in the form of a decision tree [2].

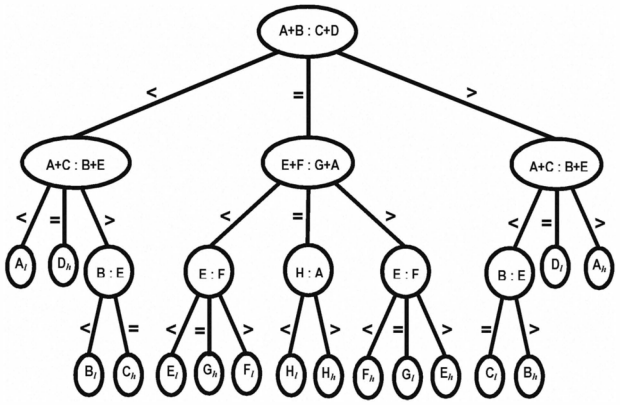


Figure 4. The second new solution of the eight coins problem in the form of a decision tree [2].

Recently two more new solutions of the eight coins problem have been published in literature [2]; the solutions are shown in Figures 3 and 4. All these four solutions are compared in this section based on their maximum number of comparisons towards a solution, their external path length, total number of comparisons required, maximum number of coins in a comparison, and average height of the decision tree. All these results are shown in Table I.

Table I. Relative merits and demerits of different solutions of the eight coins problem based on different parameters of comparisons.

Parameters of making comparisons	Classical solution (in Figure 1)	Naive solution (in Figure 2)	The first new solution (in Figure 3)	The second new solution (in Figure 4)
Maximum number of comparisons	3	5	3	3
External path length	48	56	48	44
Total number of comparisons	12	12	9	9
Maximum number of coins in a comparison	6	2	6	4
Height of the tree	4	6	4	4
Average height of the tree	3	3.5	3	2.75

III. Algorithms for Solving *N* Coins Problem

In this section in developing our algorithm we minimally modify the classical solution, shown in Figure 1, in order to compute a better solution in terms of comparisons as shown in Figure 5. Now to minimize the number of comparisons, for the equality case at the root we do not compare the coins G and H. This is because one out of G and H must be a counterfeit coin, and so they are unequal in their weight. We remove this redundant comparison and compare G with A (which is a known correct coin). If they are of unequal weight it means that G is the faulty coin, and we find out whether it is heavier or lighter from this comparison only. Otherwise, it is understood that H is faulty. In which case we compare H with the known correct coin A, and find out whether it is heavier or lighter.

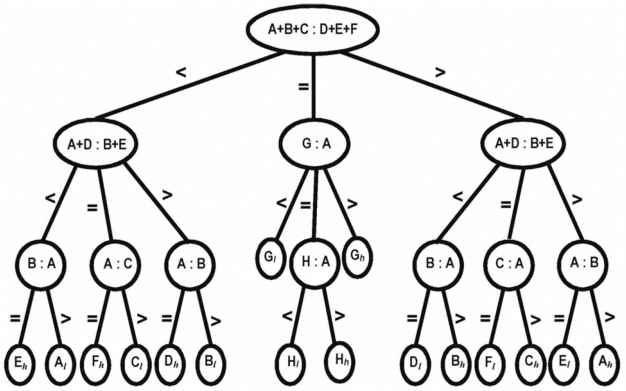


Figure 5. The existing solution in Figure 1 with some modification along the case of equality, following the root of the decision tree.

In this paper, we have developed algorithms for solving *n* coins problem dividing the problem into three cases (for different values of *n*) after extracting an algorithm behind the solution of eight coins problem shown in Figure 5, in three subsequent subsections below. We describe several procedures therein whilst developing these algorithms that we often call *function* as activity; not function as definition.

A. Algorithm for solving n > 8 (powers of 2) coins problem

The *n* coins problem is solved using the procedure *n_coin_problem* as described here in this section. This procedure takes as inputs, the starting index of the *n* coins, and the number of coins. This procedure then calls the *less_than* function or the *greater_than* function depending upon the weight of the pans. The function *less_than* is called when in the first comparison (that is done at the root of the decision tree) the weight of the left pan is less than that of the right pan. Similarly, the function *greater_than* is called when the left pan is heavier than the other. These functions are recursively called within each of the functions until only two suspected coins (of whom one is certainly a counterfeit coin) are left.

As the tree structure goes down, each of these procedures eliminates some coins at each stage, until the number of suspected coins is two. At this stage these functions take the help of the *check* function. This *check* function helps to

identify the counterfeit coin among those two suspected coins, with the help of a known correct coin. When the procedure *n_coin_problem* is being called recursively, and the number of remaining coins is only two or only four, then the procedure calls the *2_coin_problem* or the *4_coin_problem*, respectively (instead of calling the *n_coin_problem* again). Basically, these two procedures act as the base case when we are calling the *n_coin_problem* recursively. Now, let us look at the performance of the generalized algorithm for solving n coins problem, where n is a power of 2, which is greater than eight.

Input: An integer number n of coins (out of which only one coin is counterfeit, either heavier or lighter).

Output: The index z of the counterfeit coin, where z is an integer (location of the false coin), and declaring whether it is heavier or lighter.

In developing the algorithm that actually solves the eight coins problem, as shown in Figure 5, we now consider each of the procedures as outlined above in isolation and explain how they work. We start with the main procedure, i.e., the *n_coin_problem*.

- Procedure *n_coin_problem*

We have been given n coins, out of which only one coin is counterfeit (or false). To find the counterfeit coin (and also to know whether it is heavier or lighter), we use the decision tree structure. We call the *n_coin_problem* with *start* = 1 and number of coins = n. Here, *start* indicates the starting index of the coins to be considered (or compared), and n is the size of the problem. The total number of possible outcomes in case of n coins problem is equal to $2n$. We first determine the number of coins to be kept on each of the pans of the equal arm balance. This is given by $3x$, where $x = n/8$. Thus, $6x$ coins are compared initially. We can have three possible cases as follows:

1. Left pan is lighter than the right pan. This case is handled by the *less_than* function. Since the left pan is lighter, it means that any one of the $3x$ coins kept on the left pan is lighter, or any one of the $3x$ coins kept on the right pan is heavier. From this condition we can reach $2\times3x = 6x$ number of possible outcomes as leaf nodes. The counterfeit coin exists among the coins indexed by *start* and $6x$. The left pan contains the coins indexed from *start* through $3x$, and the right pan contains the coins indexed from $3x+1$ through $6x$. The first coin on the left pan is indexed by *S_left* (i.e., at the beginning it is same as *start*). Similarly, the first coin of the right pan is indexed by *S_right*, (i.e., at the beginning it is the coin same as $3x+1$).

2. Left pan is heavier than the right pan. This case is handled by the *greater_than* function. Since the right pan is lighter, it means that any one of the $3x$ coins kept on the left pan is heavier, or any one of the $3x$ coins kept on the right pan is lighter. Likewise, from this condition we can reach $2\times3x = 6x$ number of possible outcomes as leaf nodes. The indexing is the same as in the previous case.

3. Both the pans are equal. Since the pans are equal, it means that the $6x$ coins compared at the root are correct. The counterfeit coin then lies in the $2x$ number of coins yet to be considered. In this case, i.e., in the case of equality, the n coins problem gets reduced to $2x$ coins problem. To solve this $2x$ coins problem, we call the *n_coin_problem* recursively, with the number of coins now equal to $2x$. Since the first $6x$ coins are correct, now *start* = $6x+1$ and $n = 2x$. From this case of equality, we can reach $2\times2x = 4x$ number of possible outcomes as leaf nodes.

- Procedure *less_than*

In this procedure, we retain the first x coins in the left pan (starting from *S_left*) and interchange the next x number of coins with the first x coins in the right pan (starting from *S_right*), and retain the next x coins of the right pan as it is. Thus in this comparison, the last x coins of both the pans are not considered. Rather, these coins are removed from comparison at this stage. Again in this case there are three possible outcomes– (i) The left pan is lighter: The counterfeit coin exists among the first x number of coins in the left pan, or the middle x number of coins in the right pan. Thus we have to update the value of *S_right*. It now points to the first coin of the middle x number of coins of the right pan. The value of *S_left* remains as it is. (ii) The left pan is heavier: The counterfeit coin exists among the middle x number of coins in the left pan, or the first x number of coins in the right pan. Thus we have to update the value of *S_left*. It now points to the first coin of the middle x number of coins of the left pan. The value of *S_right* remains as it is. (iii) Both pans are equal: The counterfeit coin exists among the last x number of coins of both the left and the right pan. Thus the value of both *S_left* and *S_right* needs to be updated. *S_left* now points to the first coin of the last x coins of the left pan, and *S_right* now points to the first coin of the last x coins of the right pan.

Now, the value of x is halved as we move down one level in the decision tree structure. If the value of x is equal to ½, only two coins are left undecided. At this point we call procedure *check* with these two undecided coins and a known correct coin, as it has been considered for the three coins G, H, and A, where A is a correct coin (see the case of equality in Figure 5 following the root of the tree). If the value of x is more than ½, the *less_than* function is called recursively with the updated values of *S_left* and *S_right*. Case (iii) above does not occur more than once. This is due to the fact that all the undecided coins (among which the counterfeit coin exists) are compared in the next subsequent levels.

- Procedure *greater_than*

This procedure is very much the mirror image of the earlier procedure. Here we retain the first x coins in the left pan (starting from *S_left*) and interchange the next x number of coins with the first x coins in the right pan (starting from *S_right*), and retain the next x coins of the right pan as it is. Thus in this comparison, the last x coins of both the pans are not considered. Again in this case there are three possible outcomes– (i) The left pan is heavier: The counterfeit coin exists among the first x number of coins in the left pan, or the middle x number of coins in the right pan. Thus we have to update the value of *S_right*. It now points to the first coin of

the middle x number of coins of the right pan. The value of *S_left* remains as it is. (ii) The left pan is lighter: The counterfeit coin exists among the middle x number of coins in the left pan, or the first x number of coins in the right pan. Thus we have to update the value of *S_left*. It now points to the first coin of the middle x number of coins of the left pan. The value of *S_right* remains as it is. (iii) Both pans are equal: The counterfeit coin exists among the last x number of coins of both the left and the right pan. Thus the value of both *S_left* and *S_right* needs to be updated. *S_left* now points to the first coin of the last x coins of the left pan, and *S_right* now points to the first coin of the last x coins of the right pan.

Now, the value of x is halved as we move down one level in the decision tree structure. If the value of x is equal to ½, only two coins are left undecided. At this point we call procedure *check* with these two undecided coins, and a known correct coin. The *check* function takes the three coins as parameters, in the following order– lighter or correct coin (for less than condition it would be the coin indexed by *S_right*, and for greater than condition it would be the coin indexed by *S_right*+1), heavier or correct coin (for less than condition it would be the coin indexed by *S_left*+1, and for greater than condition it would be the coin indexed by *S_left*), and known correct coin. This *check* function then finds out the counterfeit coin by comparing these three coins. If the value of x is more than ½, then procedure *greater_than* is called recursively with the updated values of *S_left* and *S_right*. In a similar way, as in the case of procedure *less_than*, here also the third case (i.e., Case (iii) above) does not occur more than once. This is due to the fact that all the undecided coins (among which the counterfeit coin exists) are compared in the next subsequent levels.

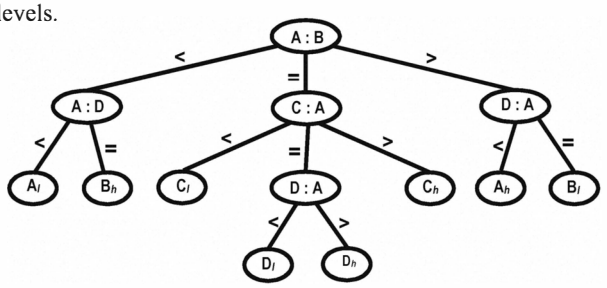


Figure 6. Decision tree for solving 4 coins problem.

- Procedure *4_coin_problem*

This procedure solves the *n_coin_problem* when the problem is reduced to $n = 4$, i.e., there are only four undecided coins out of which one is counterfeit. The parameters passed to this function are *start* (i.e., the index of the coin from which the four undecided coins occur) and the index of a correct coin. It compares the coins indexed by *start* and *start*+1. If they are equal, then it means that these two coins are correct, and the problem reduces to two coins problem. Then the *2_coin_problem* is called with *start* = *start*+2 and the index of a correct coin. Otherwise, if the weight of the initial two coins is unequal, then the *check* function is called with these two coins to find out the counterfeit coin. Figure 6 shows the solution of *4_coin_problem*, with the help of decision tree.

- Procedure *2_coin_problem*

This solves the *n_coin_problem* when the problem is reduced to $n = 2$, i.e., there are only two undecided or unchecked coins. The parameters passed to this function are *start* (i.e., the index of the coin from which two undecided coins occur) and the index of a correct coin. This gives the index of the counterfeit coin, and mentions whether it is heavier or lighter. Figure 7 shows the solution of *2_coin_problem*, with the help of decision tree.

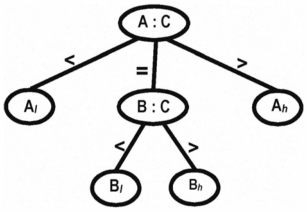


Figure 7. Decision tree for solving 2 coins problem, where C is a correct coin.

- Procedure *check*

This procedure takes three coins as parameters in the following order– lighter or correct coin, heavier or correct coin, and a known correct coin. This procedure tells whether the first coin passed is lighter (which may be the lighter coin or a correct coin), or the second coin passed is heavier (which may be the heavier coin or a correct coin), or the reverse. This is done by comparing the weights of the first and the second coin passed with the known correct coin passed.

B. Algorithm for solving $n \geq 6$ (even) coins problem

In this version of the algorithm, we first find the smallest possible value y depending upon the given value of n such that $y \geq n$, where y is a power of 2. If $y = n$, this version of the algorithm exactly matches to the steps stated in Section *A* above. Needless to mention that if $n = 8$, then $x = 1$ and the number of coins compared at the root of the decision tree is 6 keeping the first 3 coins on left pan and the next 3 coins on right pan. Similarly, for $n = 16$, $x = 2$ and the number of coins compared is equal to 12 keeping the first 6 coins on left pan and the next 6 coins on right pan, for $n = 32$, $x = 4$ and the number of coins compared is equal to 24 keeping the first 12 coins on left pan and the next 12 coins on right pan, and so on.

On the other hand, if n is not equal to y, rather $n < y$ (where $2^{p-1} < n < 2^p = y$, for some integer value of $p \geq 3$), then the number of coins compared in each pan is $2x$, where $x = y/8$. As for example, if $n = 10$, then $y = 16$ and $x = 2$, and the total number of coins compared at the root of the decision tree is 8 keeping the first 4 coins on left pan and the next 4 coins on right pan, if $n = 22$, then $y = 32$ and $x = 4$, and the total number of coins compared at the root of the tree is 16 keeping the first 8 coins on left pan and the next 8 coins on right pan, and so on.

If the left and the right pans are equal in weight at the first comparison made at the root of the tree, then the counterfeit coin lies among the remaining $n–4x$ coins of the n coins problem. For example, if the decision follows the equality branch of the tree for $n = 10$, then the counterfeit coin belongs

to the remaining 2 coins, for n = 22, the counterfeit coin belongs to the remaining 6 coins, and so on. Then following this equality branch of the decision tree we either recursively call procedure *n_coin_problem*, or straightway call procedure *4_coin_problem*, or *2_coin_problem*, as the situation arises.

On the other hand, if the left pan is lighter than the right pan, then the counterfeit coin lies among the first $4x$ coins of which one of the first $2x$ coins can be lighter or one of the next $2x$ coins can be heavier. Then it calls procedure *less_than* to deal with the left subtree of the tree.

If the left pan is heavier than the right pan, then the counterfeit coin lies among the first $4x$ coins of which one of the first $2x$ coins can be heavier or one of the next $2x$ coins can be lighter. Then it calls procedure *greater_than* to deal with the right subtree of the tree.

In this way, each time we move down towards the leaf of the decision tree the value of x is divided by 2. When x = ½, there are only 2 coins undecided. One of these 2 coins is the counterfeit one. This time procedure *check* finds the desired counterfeit coin, and also declares whether it is heavier or lighter.

C. Algorithm for solving n ≥ 7 (odd) coins problem

In our algorithm, this is an obvious checking whether the given number n of coins is even or odd. If it is even, then it is checked whether it is a power of two. If it is a power of two, we follow the algorithm developed in Section *A*; otherwise, we follow the algorithm as it has been developed in Section *B*. On the other hand, if the value of n is odd, we do some prior computations. In this case, just one additional checking is made as it has been detailed below.

We know that if n is odd, then we can write $n = 2m+1$, where $2m$ is obviously an even number. Then we divide these $2m$ coins into two groups, each having m number of coins; each of the groups of m coins is kept on each of the pans of the equal arm balance. If they are equal, then the counterfeit coin must be the last coin, i.e., the nth indexed coin. But still we do not know whether it is heavier or lighter than each of the remaining coins. For that, we compare the last coin (i.e., the nth indexed coin) with the first coin, which is a known correct coin. If the last coin is lighter than the correct coin, we conclude that the nth coin is lighter; otherwise, the last coin must be heavier than the correct coin and we conclude that the nth coin is heavier.

On the other hand, the equal arm balance must show the case of inequality, where m coins in a group are kept on either of the pans, and the counterfeit coin belongs to amongst the first $n-1$ (= $2m$) coins under consideration, which is a case of finding the counterfeit coin for a given set of even number of coins. Incidentally, this algorithm has been developed in Section *B*.

IV. APPLICATIONS OF THE PROBLEM

The application of the problem under consideration is not only limited to coins, but to any kind of valued article that are frequently counterfeited, like ornaments, watches, metal (like gold), jewellery, etc. A *counterfeit* is an imitation that is made usually with the intent to deceptively represent its content or true origin. The word *counterfeit* most frequently describes forgeries of currency or documents, but can also describe software, pharmaceuticals, clothing, and more recently, motorcycles and cars, especially when these result in patent or trademark infringement.

Coin counterfeiting occurs regularly in the antique coin market, but there are various modern forgeries that also make it into general circulation [1, 4]. Counterfeit antique coins are generally made to a very high standard so that they often fool collectors; this is not easy and many coins still stand out.

The importance of solving this problem is more in the theoretical field of computer science and/or mathematics. The very fact that the decision tree structure can be used to solve such problems of large size, by eliminating a part of the solution domain after each step of decision making, is of utmost importance. Especially because, as our algorithm works for any value of n, it does not matter if the value of n is not known a priori. This is, in fact, a new innovative work, and important in solving problems using the decision tree structure.

V. CONCLUSION

At the very beginning, we have made it clear that our objective is to develop algorithms for solving the counterfeit coins problem, where the number of coins can be anything, from two to any large value. We first observe the existing solution for the eight coins problem, and then we modify it slightly to suit our needs. After that, we extract the algorithm behind it and generalize it for all values of n, where n is an even number. We identify two types of even numbers, ones which are powers of two, and the other is the rest of the numbers, six or more. The algorithm works slightly in different way for the two types of cases, but the solution is reached in both cases.

After developing the algorithm for solving the counterfeit coin problem where the number of coins is even, we widen our algorithm for handling the situation of odd number of coins problem. This shows that the algorithm developed herein is truly generalized, and works for all values of n (where n is the number of coins). Furthermore, the algorithm terminates after executing for a finite (and predictable) number of steps.

REFERENCES

[1] Gagg C. R. and P. R. Lewis , "Counterfeit Coin of the Realm – Review and Case Study Analysis", *Proc. of 2nd International Conference on Engineering Failure Analysis (ICEFA-II)*, Toronto, Canada, Sep. 2006, Engineering Failure Analysis, vol. 14, no. 6, pp. 1144 -1152, 2007.

[2] Ghosh J., S. K. Ghosh, and R. K. Pal, "A Revisit to the Eight Coins Problem", *International Journal of Computing and Information Technology (IJCIT) (ISSN: 0974-696X)*, vol. 2, no. 1, pp. 1-14, 2010.

[3] Horowitz E. and S. Sahni, *Fundamentals of Data Structures in Pascal*, Galgotia Booksource, New Delhi, 1984.

[4] Lewis P. R., C. R. Gagg, and K. Reynolds, *Forensic Materials Engineering: Case Studies*, CRC Press, 2004.

[5] Reingold E. M., J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

[6] Rosen K. H., *Discrete Mathematics and Its Applications* (Fifth Edition), Tata McGraw-Hill Publishing Co. Ltd., New Delhi, 2003.

ISSN 2279-087X (Print)
ISSN 2279-0888 (Online)

Annals of Pure and Applied Mathematics

Volume 7, Number 1
2014

Special issue on
Optimization and Fuzzy Mathematics

Editor-in-Chief
Professor Madhumangal Pal

House of Seientific Research
www.researchmathsci.org

nnals of Pure and Applied Mathematics
ol. 7, No. 1, 2014, 77-82
SSN: 2279-087X (P), 2279-0888(online)
ublished on 9 September 2014
ww.researchmathsci.org

Annals of
Pure and Applied
Mathematics

An Advanced Approach to Solve two Counterfeit Coins Problem

Joydeb Ghosh[1], *Lagnashree Dey*[2], *Ankita Nandy*[2], *Arpan Chakrabarty*[2]
Piyali Datta[2], *Rajat Kumar Pal*[2] and *Ranjit Kumar Samanta*[3]

[1]Department of Mathematics, Surendra Institute of Engineering and Management, New Chamta, Siliguri, Darjeeling – 734 009, West Bengal, India
Email: joydeb009@gmail.com

[2]Department of Computer Science and Engineering, University of Calcutta
92, A. P. C. Road, Kolkata – 700 009, West Bengal, India
Email: {rose2009mail, yoursankita.nandy09, arpan250506, piyalidatta150888, pal.rajatk}@gmail.com

[3]Department of Computer Science and Application, North Bengal University
Darjeeling – 734 013, West Bengal, India
E-mail: rksamantark@gmail.com

Received 21 July 2014; Accepted 21 August 2014

Abstract. Though the counterfeit coin problem is well known as a fascinating puzzle it claims great importance in Computer science, Game theory, and Mathematics. In terms of the puzzle the objective is to detect the counterfeit coins which are identical in appearance but different in weight. The word *counterfeit* not only describes forgeries of currency or documents, but can also be applied to software, pharmaceuticals, clothing, and more recently, motorcycles and cars, especially when these result in patent or trademark infringement. Furthermore, the goal in this problem is to minimize the number of weighing, i.e., the number of comparisons required to find the false coin/s and their type (whether heavier or lighter than the original coin). Finding one counterfeit coin among n coins is complex and tricky enough. The problem gets more complicated when the set of n coins contains two false coins as the false coins pair may appear in several different combinations. In this paper, we have developed a new algorithm for solving two counterfeit coin problem in O(log$n$) time, where n is the total number of coins.

Keywords: Counterfeit coin problem, equal arm balance, desgin of algorithm, decision tree, complexity

AMS Mathematics Subject Classification (2010): 05C78

1. Introduction

Computing a solution of the counterfeit coin problem has huge significance in both theoretical and commercial sphere as well as to prevent forgery in different fields. The objective is to minimize the number of weighing for which it is sufficient to determine

the defective coin(s) in a set of n coins using only an equal arm balance, when the number of odd coins is precisely known and they are identical in appearance but different in weight (either heavier or lighter) than a true coin. The complexity of the problem increases with the increment of the number of counterfeit coins in a set. If P is the number of counterfeit coins in a set of n coins, it is not only sufficient to consider whether the counterfeit coins are heavier or lighter in comparison to a genuine coin individually, but we must also take into account their mutual relation like equally heavier or lighter, unequally heavier or lighter, etc. In this paper, we consider that there are two false coins in a set of n coins which are equally heavier (or equally lighter) in comparison to a genuine coin. The objective is to identify the counterfeit coins using a minimum number of comparisons (or weighing).

2. Literature survey

In paper [4], the problem has been introduced in two ways. In the first case, we do not know if there is a fake coin in the given set. The process is supposed to check it first, and if yes, then identify the targeted coin by means of a minimum number of weighing. In the second case, it is told that there is a counterfeit coin and the objective is to find the coin through minimum number of weighing. At times, a standard coin may also be given. In the first case, if a lighter coin is there in the given set S of coins, then it is proved that the least number of weighing to find out the fake coin satisfies $3^{n-1} < |S| \leq 3^n$ for some unique value of n, where $|S|$ is the cardinality of set S. In the second case, we are given a set S of coins plus a standard coin, where only one coin in S is of different weight. Then it is proved that $(3^{n-1}-1)/2 < |S| \leq (3^n-1)/2$.

In paper [1], the problem has been addressed as an application of dynamic programming and the associated analysis has been made through optimal and suboptimal testing policy. Here also only one coin is defective out of n given coins. This technique always assumes $k < n$ coins in each pan for each weighing, where the value of k essentially depends on the value of n. If the two groups balance, the defective coin must be in the remaining $n-2k$ coins; otherwise, the false coin is in one of the k groups. After each weighing, the number of coins to be examined reduces, but the problem remains the same. This allows the authors to apply dynamic programming to this problem.

One classical solution is available in the form of a *decision tree* that represents a set of all possible decisions by which we can acquire the desired solution(s) of the problem [2, 3]. In this solution, each internal vertex (that is not a leaf vertex) symbolizes a comparison between a pair of equal sets of coins using an equal arm balance. Here the problem under consideration is more generalized; the fake coin can either be heavier or lighter. So, for n given coins, there are $2n$ leaf vertices in the tree as probable solutions.

In paper [5], the problem of ascertaining the minimum number of weighing which suffice to determine the counterfeit (heavier) coins in a set of n coins of the same appearance, given a balance scale and the information that there are exactly two heavier coins present, has been considered. Both of heavier coins are of equal weight and they are not heavier than 3/2 times than the true coin. If p is the maximum number of comparisons required to find out two false coins (equally heavier), the paper introduces an algorithm which has the lower bound $\lceil \log_3({}^nC_2) \rceil$. In this paper, an infinite set of n has been determined for which this lower bound is reached, whereas the upper bound is only one unit more than the lower bound.

Problem formulation and algorithm

this section, we formulate and develop an efficient algorithm to solve two counterfeit ins problem. The algorithm finds both the counterfeit coins among n number of coins, hich are indexed sequentially from 1 to n. Finding out two false coins introduces veral cases, i.e., different combinations of false coin pairs, such as heavier-heavier, ghter-lighter, heavier-lighter, etc. The behaviour of the problem changes with the ange in the specification of the problem. In this paper, we consider the case where both lse coins are equally heavier (or equally lighter) than the true coin and any two coins mong n coins may be false. We assume that the heavier (lighter) and the true coin are enoted by H (L) and T, whereas the weight of those are w(H) (w(L)) and w(T) espectively. The issue to be considered in this problem is the minimum value of n such hat we can find two false coins among them without using any extra standard coin.

Lemma 1. To find p counterfeit coins among n coins without taking help of an additional genuine coin, the value of n has to be at least of $2p+1$.

Proof: If there is one false coin among two coins, a standard coin is necessary to detect the false coin unless the weight of the correct coin is given. So if the number of false coins, $p = 1$, the minimum value of $n = 3 = 2p+1$. Now, if there are two false coins we can identify them from four coins if and only if two false coins are of different weight. If they are of same weight, we cannot conclude which set of coins is true, as there are equal numbers of false and true coins. So, for $p = 2$, the minimum value of $n = 5$, i.e., $2p+1$. In general, if there are p counterfeit coins, we can detect them from $p+2$ coins if all the p coins are of distinct weights. But if at least two false coins are of same weight, we cannot distinguish them from the set of all coins. So, to satisfy the above cases specially the case where all the false coins are of same weight, we need at least one more true coin than the false coin, i.e., the minimum number of total coins required is $p+p+1 = 2p+1$. □

The algorithm proceeds by dividing the coins into three sets, say $K1$, $K2$, and $K3$. The results of division depends on three cases: (i) n is divisible by 3, i.e., $n|3$, (ii) $n+1$ is divisible by 3, i.e., $(n+1)|3$, and (iii) $n-1$ is divisible by 3, i.e., $(n-1)|3$. Thus, depending on the value of n it can easily be decided to which group the set of n coins belongs to and precisely which variant of the algorithm can be applied on the given set of coins. We assume that the coins are indexed by natural numbers 1 through n. For the first case, $|K1| = |K2| = |K3| = n/3$. For the second case, $|K1| = |K2| = (n+1)/3$, and $|K3| = n-2(n+1)/3 = (n-2)/3$. So, there is a difference of one coin between $K1$ and $K3$, or $K2$ and $K3$. For the third case, $|K1| = |K2| = (n-1)/3+1 = (n+2)/3$, and $|K3| = n-2(n+2)/3 = (n-4)/3$. There is a difference of two coins between $K1$ and $K3$, or $K2$ and $K3$.

After dividing the set of n coins into $K1$, $K2$, and $K3$, at first $K1$ and $K2$ are put on the arms (or pans) for weighing. Depending on the outcome of the weighing three versions of the algorithm proceeds towards the next weighing taking different sets. Considering the result of its parent nodes some weighing is performed at each internal node. At the leaf nodes we perform either one TCP(K_i) operation or two OCPH(K_i) operations (OCPL(K_i) in case of equally lighter coins) to find out both the counterfeit coins in set K_i. Whenever we are sure that there is only one heavier (lighter) false coin in K_i, we call OCPH(K_i) (OCPL(K_i)). When we are convinced that there are two false coins in K_i, the algorithm is recursively applied to K_i, considering its version. We denote this operation as TCP(K_i) in general. Figure 1 shows the decision tree for the version of the

algorithm when $n|3$ (in case of H1 = H2). At the root node, $K1$ and $K2$ are compared and the result is analyzed as follows. If $w(K1) = w(K2)$, either $K1$ and $K2$ contain one false coin each or $K3$ contains both the counterfeit coins. So next we compare $K2$ and $K3$. There are two possibilities.

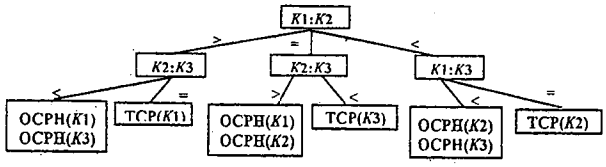
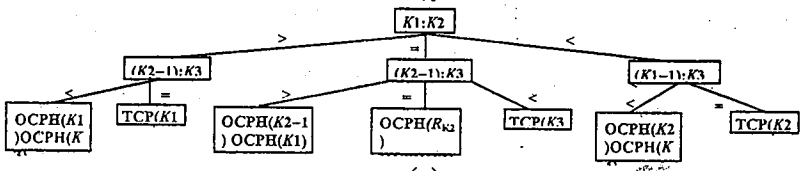
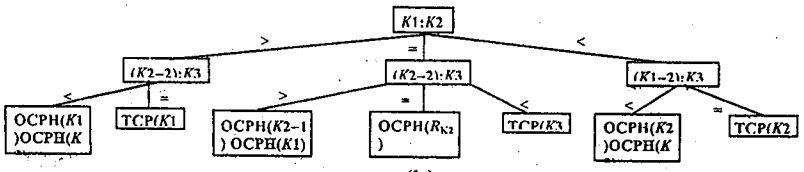


Figure 1: Decision tree of the algorithm for



(a)



(b)

Figure 2: **(a)** Decision tree for $(n+1)|3$. **(b)** Decision tree for $(n-1)|3$.

If $K2 > K3$, then we are certain that one false coin is in $K1$ and the other is in $K2$. So, we perform OCPH($K1$) and OCPH($K2$). If $K2 < K3$, then we are sure that both the false coins are in $K3$; thus, we perform TCP($K3$). If $w(K1) > w(K2)$, then either both the false coins are in $K1$ or one false coin is in $K1$ and another is in $K3$. So, we compare $K2$ and $K3$ in the next step. The former case arises when $w(K2) = w(K3)$. So, TCP($K1$) is applied. But if $w(K2) < w(K3)$, then OCPH($K1$) and OCPH($K3$) are performed. Figure 2(a) shows the decision tree when $(n+1)$ is divisible by 3. In this case the algorithm does the same thing as for the version $n|3$ except the second level of comparisons. It takes $(|K2|-1)$ coins instead of $|K2|$ and $(|K1|-1)$ coins instead of $|K1|$. For the sake of determinism, we always prefer the coins from set K_i except the last coin in the set. The subsequent operations in the levels followed are shown in the decision tree.

For $(n-1)|3$, the algorithm proceeds in the same way as for the version $(n+1)|3$ with a little difference in the second level of comparisons in the all the branches. As $K3$ contains two coins less than that of $K1$ or $K2$, it receives $(|K2|-2)$ coins instead of $|K2|$ and $(|K1|-2)$ coins instead of $|K1|$. The operations in the subsequent levels are shown in the decision tree in Figure 2(b). When we call OCPL(K_i) or OCPH(K_i), the algorithm proceeds dealing with that set K_i of coins and we divide it into two equal subsets for further weighing. If this set contains even number of coins we put half of them on the left pan and the remaining half on the right pan and weigh, i.e., comparison is performed. Again at this stage, if we are searching for a heavier coin, i.e., in case of OCPH(), after the weighing we deal only with the coins in the heavier pan. If the number of coins is odd, we divide them into two equal halves and one coin remains out of weighing. If the

ighing results in inequality, we focus on either the left pan or the right pan depending the outcome we examine.

So far we have developed an algorithm to find out two false coins among a set of coins where both the coins are equally heavier, i.e., w(H1) = w(H2). For the variant here two coins are equally lighter, i.e., w(L1) = w(L2), the algorithm works as well.

Experimental results

n this section, we choose some values of n so that it would cover all the three categories or the subdivision of n and calculate the average number of comparisons required. To ompute the average case complexity, we have to consider pair of false coins in all ossible pairs of indexed locations. Hence for a given value of n, there are ${}^nC_2$ ombinations as the combination of ith heavier and jth heavier is same as the combination of jth heavier and ith heavier being the two false coins equally heavier, we cannot listinguish them. Hence, there are $O({}^nC_2)$ leaves in the decision tree. The average number of comparisons required against the number of given coins are shown in Table 1 and the same is plotted in Figure 3.

Table 1: Average number of comparisons for some values of n.

Number of coins (n)	Total number of comparisons (S)	Possible number of false coin combinations ($C = {}^nC_2$)	Average number of comparisons (AVG = S/C)
9	171	36	4
11	277	55	5
20	1254	190	6
27	2565	351	7
29	2939	406	7
36	5508	630	8
46	9201	1035	8
54	13365	1431	9
72	27396	2556	10
82	34267	3321	10
100	54926	4950	11
108	65232	5778	11
144	130842	10296	12
198	251883	19503	12
200	254788	19900	12

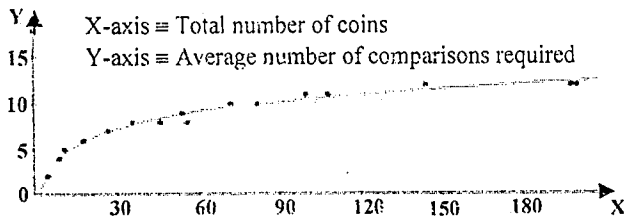


Figure 3: Plot of average number of comparisons required against the number of coins.

5. Computational complexity

At each iteration, n is divided into nearly three equal parts and the cardinality of the set on which the operations are actually performed, always reduces by a factor of three. Let us consider the case $n|3$. As we see at the ith level, each set is of cardinality $n/3^i$. Now if we reach a set with five coins, then we can solve it using exactly four comparisons. So, let at the ith level of comparison the cardinality of the set reduces to five. Thus, $n/3^i = 5$, i.e., $3^i = n/5$. Hence, $i = \log_3(n/5)$. Again, if TCP(K_i) is applied at each iteration before reaching a set with five coins, $2\times i$ comparisons are required resulting in $2\times i+4$ comparisons in total. If OCPH(K_i) or OCPL(K_i) is applied at kth level of comparison, it is definite that before that iteration TCP() is applied for $(K-1)$ times. We know that OCPH() or OCPL() requires $\lceil \log_2 n \rceil$ comparisons and at kth level it is to be applied on $n/3^k$ coins. Hence, it requires total number of $2|K|+2\log_3(n/3^k)$ comparisons. So, in the worst case it would take $O(2|K|+2\log_3(n/3^k)) + O(2\times\log_3(n/5)+ 4)$, i.e., $O(\log n)$ comparisons as a whole.

6. Conclusion and applications

The raising issue of counterfeits violates intellectual property right and also causing damage to both producer and consumer. To identify the counterfeit goods like pirated electronic gadgets, counterfeit batteries used in a digital camera, pharmaceuticals, valuable ornaments, the solution of counterfeit coin problem are used. In this paper, we have developed an algorithm to identify two false coins among a set of n coins that are identical in appearance. In this case we have assumed that both the false coins are equally heavier (or lighter) than the weight of a true coin, and developed algorithms for identifying the same. The algorithm solves the problem with time complexity $O(\log n)$. The most important fact is that the decision tree structure can be used to solve such problems of large size, by eliminating a part of the solution domain after each step of decision making. Especially, as our algorithm works for any value of n, it does not matter if the value of n is not known a priori.

REFERENCES

1. R.Bellman and B.Gluss, on various versions of the defective coin problem, *Information and Control*, 4(2-3) (1961) 118-131.
2. J.Ghosh, P.Senmajumdar, S.Maitra, D.Dhal and R. K. Pal, A generalized algorithm for solving n coins problem, *Proc. of 2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE 2011)*, Shanghai, China, vol. 2, pp. 411-415, Jun. 10-12, 2011.
3. J.Ghosh, P.Senmajumdar, S.Maitra, D.Dhal and R.K.Pal, Yet another algorithm for solving n coins problem, *Assam University Journal of Science & Technology: Physical Sciences and Technology*, 8(II) (2011) 118-125.
4. B.Manvel, Counterfeit coin problems, *Mathematics Magazine, Mathematical Association of America*, 50(2) (1977) 90-92.
5. R.Tošić, Two counterfeit coins, *Discrete Mathematics*, 46 (1995) 295-298.