# 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.

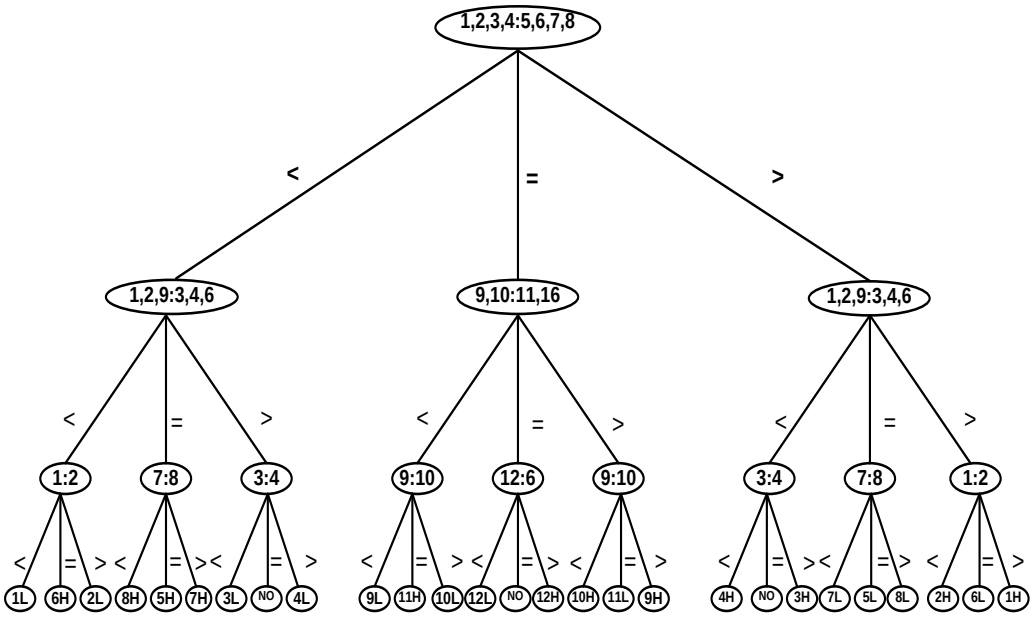


**Figure 2.1:** The solution of the twelve coins problem in the form of a decision tree.

## 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 $\lambda$ be the weight of a good coin and $\mu$ 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.

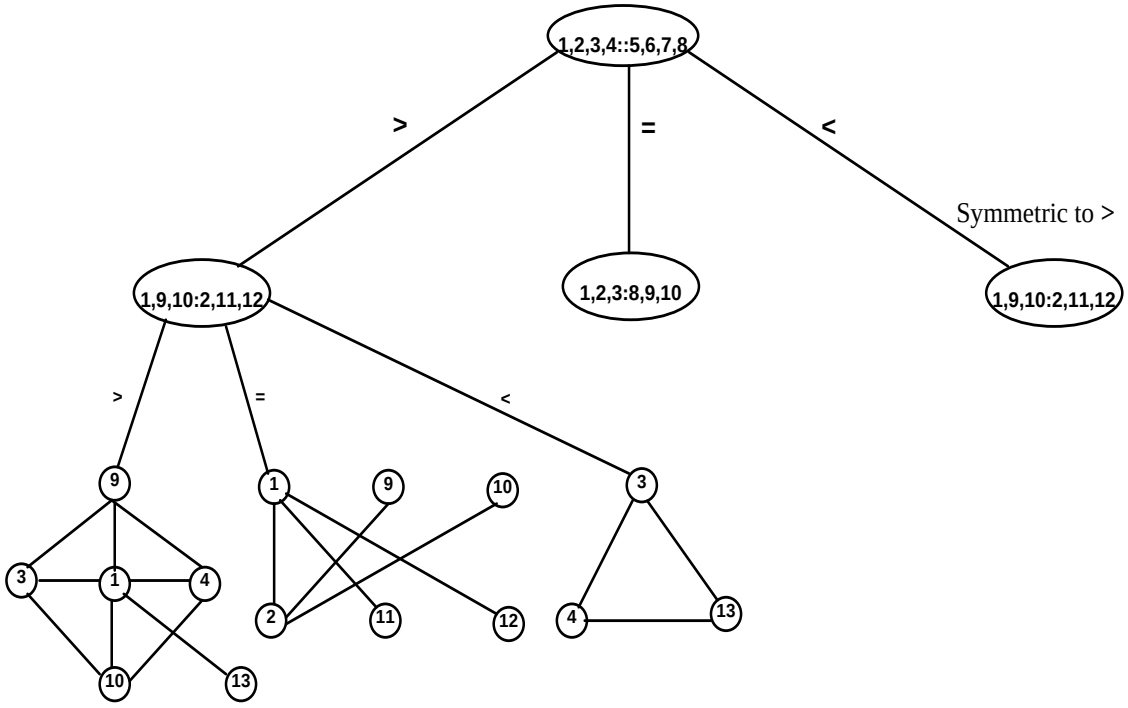


**Figure 2.2:** The solution of the two counterfeit coins problems among thirteen coins.

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.

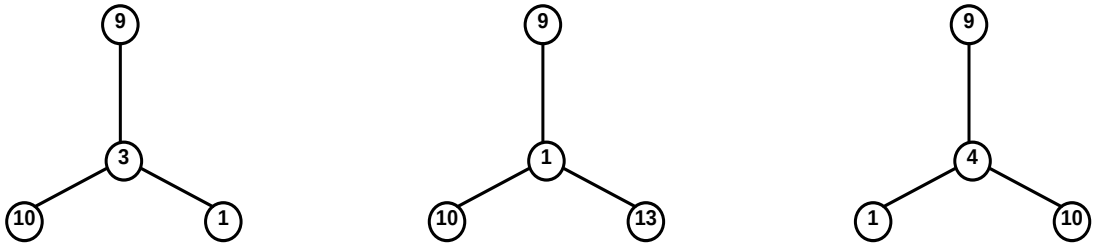


**Figure 2.3:** Explanation of Figure 2.2 when we get *greater than* answer.

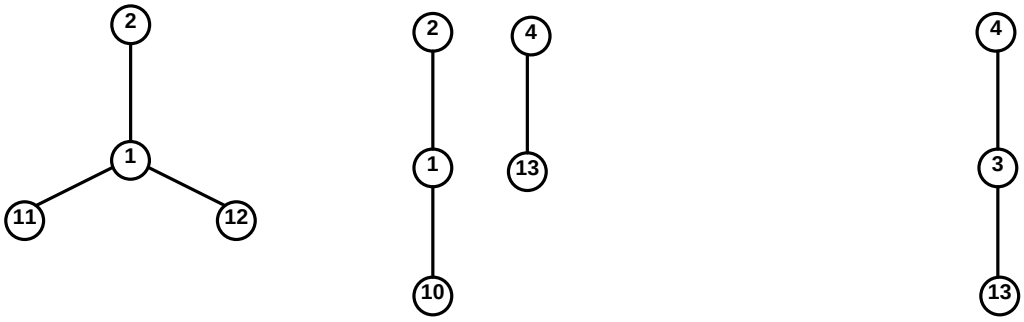


**Figure 2.4:** Explanation of Figure 2.2 when we get *less than* answer in the second symmetric.

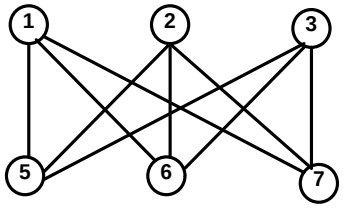


**Figure 2.5:** Explanation of Figure 2.2 when we get three answers *balance*, *less than*, and *greater than*.

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*).

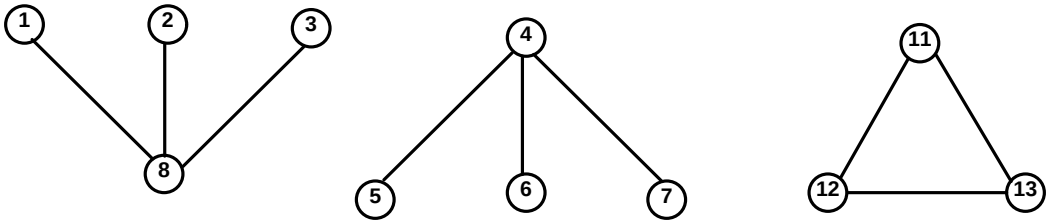


**Figure 2.6:** Explanation of Figure 2.2 when we get *direct equal* answer following comparison will be considered.

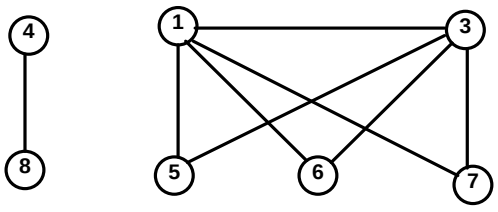


**Figure 2.7:** Explanation of Figure 2.2 when we get *less than* answer following comparison will be considered.

### 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 $\pi$ 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 $\pi$ define a directed graph $G_\pi$ 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 $2^{\alpha}$. If the result is negative, the $2^{\alpha}$ 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 $i$th 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.