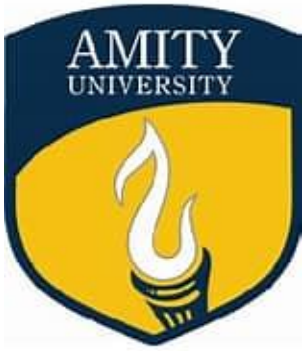
NTCC Report
on
Theory and Algorithms

Submitted to
Amity university Uttar Pradesh



In partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology In
Computer Science and
Engineering By

Amulya Singh / Vaibhav Kumar / Nishant Yadav
A41105221042 / A41105221007/ A41105221024

Under the guidance of

Mr. Pradeep Kumar Kushwaha &
Mr. Bhanu Prakash Lohani

Department of Computer Science and Engineering

Amity School of engineering
Amity University Uttar Pradesh

Amity School of engineering
Amity University Uttar Pradesh

AMITY UNIVERSITY

-----GREATER NOIDA CAMPUS-----

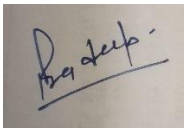
NTCC COMPLETION CERTIFICATE

This is to certify that

Amulya Singh

student of B.Tech. CSE, 3rd semester, 2021-25 batch has successfully completed the NTCC ***"Term Paper (ETTP100)"*** from 20th June 2022 to 17th July.

Faculty Guide



Mr. Pradeep Kumar Kushwaha
Mr. Bhanu P Lohani

Declaration

We, students of B-Tech (C.S.E.-2021-2025) hereby declare that the project titled “Theory and Algorithms " which is submitted by me to the Department of Computer Science and Engineering, Amity School of Engineering Technology, Amity University Uttar Pradesh, in partial fulfilment of requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering, has not been previously Formed the basis for the award of any degree, diploma or other similar title or recognition.

The Author attests that permission has been obtained for the use of any copyrighted material.

1. Appearing in the Dissertation / Project report other than brief excerpts requiring only proper acknowledgment in scholarly writing and all such use is acknowledged.

Date: 14-07-2022

Amulya Singh / Vaibhav Kumar / Nishant Yadav

A41105221042 / A41105221007/ A41105221024

B-tech CSE (2021-2025)

Certificate

This is to certify that Amulya Singh / Vaibhav Kumar / Nishant Yadav student of B-Tech in Computer Science and Engineering has carried out work presented in the project of the Term paper entitle "Theory And Algorithms" as a part of second year program of Bachelor of Technology in Computer Science and Engineering from Amity University, Uttar Pradesh, under my supervision.

Mr. Pradeep Kumar Kushwaha &
Mr. Bhanu Prakash Lohani

[Department of Computer Science and engineering]

ASET

Acknowledgment

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success.

I would like to thank prof Dr Head of Department-CSE and Amity University for giving me the opportunity to undertake this project.

I would like to thank my faculty guide Mr. Pradeep Kumar Kushwaha who is the biggest driving force behind my successful completion of the project. He has been always there to solve any query of mine and also guided me in the right direction regarding the project. Without his help and inspiration, I would not have been able to complete the project.

Also, I would like to thank my batch mates who guided me, helped me and gave ideas and motivation at each step.

Amulya Singh / Vaibhav Kumar / Nishant Yadav
A41105221042 / A41105221007 / A41105221024

Theory and Algorithms

- Amulya singh

Contents

Introduction to Algorithms

- Priori Analysis and Posteriori Testing.
- Characteristics of Algorithms
- How Write and Analyze Algorithms
- Frequency Count Method
- Time Complexity
- Classes of Functions
- Compare Class of Functions
- Asymptotic Notation -Big Oh- Omega -Theta
- Properties of Asymptotic Notations
- Comparison of Functions
- Best Worse and Average Case Analysis
- Disjoint Sets Data Structure – Weighted Union and Collapsing Find

Divide and Conquer

- Recurrence Relation
- Masters Theorem Decreasing Function
- Recurrence Relation Dividing
- Masters Theorem in Algorithms for Dividing Function
- Root function
- Binary Search
- Merge-Sort Algorithms
- Quick-Sort Algorithms
- Strassens matrix Multiplications

Greedy Method

- Knapsack Problem
- Job Sequencing with Deadlines
- Optimal Merge Pattern
- Huffman Coding
- Prims and Kruskals Algorithms
- Dijkstra Algorithms

Principle of Optimality

- Multi-Stage graph
- All Pairs Shortest path
- Matrix chain Multiplication
- Bellman Ford Algorithms
- Knapsack-two methods
- Optimal binary search tree
- Traveling Salesman
- Reliability Design
- Longest Common Subsequences (LCS)

Graph transversals – BFS & DFS -Breadth First Search and Depth First Search

- Articulation Point and Biconnected Components

Introduction to Backtracking – Brute Force Approach

- N queens
- Sum of subsets
- Graph coloring
- Hamiltonian cycle

Branch and Bound Introduction

- Job sequencing with deadline
- Knapsack
- Traveling salesman

NP - Hard and NP – Complete

- NP – hard graph

Knuth - Morris – Pratt (KMP) String Matching Algorithms

Rabin - Karp String Matching Algorithms

AVL Tree – Insertion and Rotations

B Trees and B+ Trees

About

Ada Lovelace, an English mathematician, wrote the first algorithm for a machine in the 1800s and is considered the first computer programmer. An algorithm is a set of steps or instructions for completing a task. We are using algorithms in our daily life (example. You make tea you follow few steps like first of all you will take a pen add some water in it you will put that pan etc.). In this we will use in formal way. you might already know how to code but if you don’t know about algorithm you're not a real developer. interview contained algorithm questions. our goal is to get you comfortable with the basics of algorithms. To design a better program algorithm are required first of all algorithms are written then programs are written. So in this we will learn algorithm from beginner to advance level cover mostly all the topics with theoretical questions. Firstly we will start from basic information and then learn few methods we can use these for simplifying algorithms.

Introduction to Algorithms.

ALGORITHMS.

Algorithm is step by step solving procedure for solving computational problems. It is written at design time so point is first you design and then you write the program. Having domain knowledge about algorithm is important before writing it. We can use any language (like c++, c, java, python etc.). It does not depend on hardware and software (means independent of operating system).

Priori Analysis and Posteriori Testing.

Priori Analysis	Posteriori Testing
1. pre-analysis means we will do the analysis of an algorithm.	1. Testing is done over program.
2. Independent of language.	2. dependent of language.
3. It is hardware independent.	3. It is hardware dependent.
4. we will find out the time and the space consumed by an algorithm	4. we will find out how much time and also amount of memory consuming in term of bytes.

Characteristics of Algorithm.

1.Input

May not take any input but algorithm can take zero or more input.

2.Output

Algorithm must generate one output otherwise its no use of writing algorithm. It must generate some result.

3.Definiteness

Every statement should be an ambitious and should have a single and exact meaning. You cannot write any statement which cannot be understood or which cannot be solved.

4.Finiteness

Algorithm is just like a function. function will have some limited set of steps and it will stop and returns the result. Same way algorithm must terminate at some point. It must have finite sets of statement. It must stop

5.Effectiveness

Algorithm is like a procedure. It not having unnecessary statement in algorithm. We use algorithm for getting some effective result.

How Write and Analyse Algorithms.

You can use any language to write an algorithm.

Example:

```
Algorithm Swap (a, b)
 {
    Temp = a;
     a = b;
     b = temp;
  };
```

We all are familiar to data type so for first you write program

Then decide what data types are required.

You can right it in different way like

```
Algorithm Swap (a, b)
 begin
    Temp: a;
    a: b;
    b: temp;
 end;                etc.
```

Write in the way that anyone can understand. First you have to design and then you write the program.

How to analyze an Algorithm

1. Time:

The algorithms are the procedures for solving problem whether you do it manually using pen and paper or you make a program and let your machine do it whatever the method you use how much time it is taking. if the procedure is very lengthy and time-consuming other whether the very fast and quickly you can get the results. Lastly we have to find time function. algorithm you write should be time efficient means it must be fast . so time is the important criteria on which we have to analyze.

2. Space:

As the algorithm is going to be computing to the program and it's going to run on the machine then we need to know how much memory space it will consume. So this is the second criteria which we will analyze.

3. Network consumption:

Every application is either internet-based or say cloud base so data transferred or network consumption is also important criteria. how much data is going to be transferred. if the algorithm or procedure that you are writing if it's unnecessary transferring to larger size data or it can be reduced or compressed. so that is one more important criteria that is how much data transfer is done.

4. Power consumption:

Now days most of the devices are hand held or PC's etc we are consuming some amount power. so Power consumption is also important to analyze.

5. CPU Registers:

If you are developing an algorithm for device driver or system level programming. If you are writing some algorithm you also need to know how much CPU registers it is consuming.

[If you have any other criteria or other factor you want to consider you can undoubtedly consider them.]

Frequency Count Method

Time taken can be calculated by frequency count method or time taken by an algorithm can be known by assigning one unit of time for each statement and if any statement is repeating for some number of times so the frequency of statement of execution of that statement will calculate and we find the time taken by an algorithm.

EXAMPLE:

```
Algorithm sum(A,N)
{
        S = 0 ;                                  ---------- time = 1
        for (i=0; i<n; i++  ){          -------- time = 2n+2 or n+1
        // so i = 0 ----time = 1
        // i<n ------ time = n+1
        // i++ -------time = n
        S= S+ A[i];                     ---------- time = 1
}
        return s;
};                          So, time function = 2n+3
```

: order of 0(n)

For space :

Total variable

A: the size of A is = n

n: the size of n is = 1

s: the size of s is =1

i : the size of I is =1

Space complexity: n+3.

So this polynomial is also of order 0(n).

Classes of Functions

Types of time function:

0(1) -----> Constant

0(log n) -------> Logarithmic

0(n) --------> Linear

0(n^2) ---------->Quadratic

0(n^3) ---------> Cubic

0(2^n) ---------> Exponential

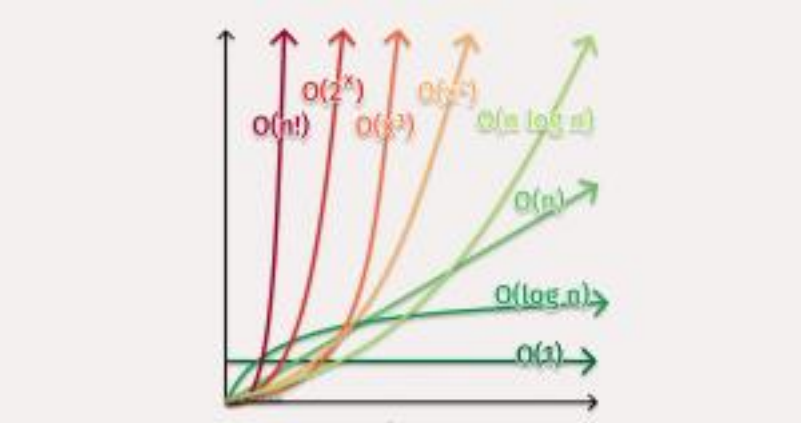
Compare Class of Functions

1 < log n < √n < n < n log n < n^2 < n^3 < ----------< 2^n < 3^n < -----< n^n.

EXAMPLE:

	Log n	n	n^2	2^n
If n = 1	0	1	1	2
If n = 2	1	2	2	2

If n = 4	2	4	16	16
If n = 8	3	8	64	256



Asymptotic Notation

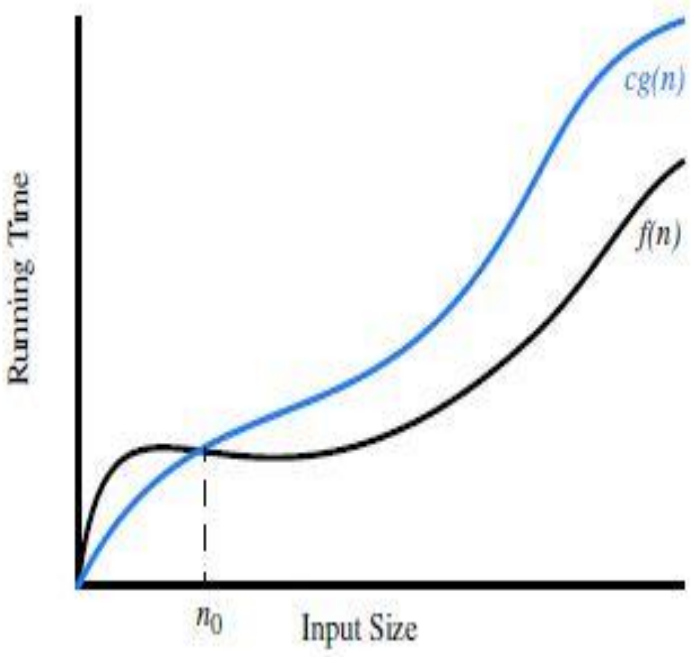
The notation is use for representing the simple form of a function or showing the class of a function. If we have any function with the help of notation we can show that this function belongs to this and this class. We also know the comparison of function which is smaller or greater than which one. We cannot write lengthy function every time we need a simple method for representing time complexity so this why Asymptotic Notation is one of important topic of algorithm.

1) Big-oh (O)

Big oh notation is used to describe an asymptotic upper bound.

Mathematically, if f(n) describes the running time of an algorithm; f(n) is O(g(n)) if and only if there exist positive constants c and n° such that:

$0 \leq f(n) \leq c\, g(n)$ for all $n \geq n°$.



EXAMPLE: f(n) = 2n+3

By big oh notation

$2n+3 \leq 10n$ (if we put n = 1 , 5<10) for all $n \geq 1$.

C= 10, f(n)=2n+3, g(n) = n

f(n)= O(n)

Always chose closest function

2) Big Omega (Ω) :

Ω notation provides an asymptotic lower bound.

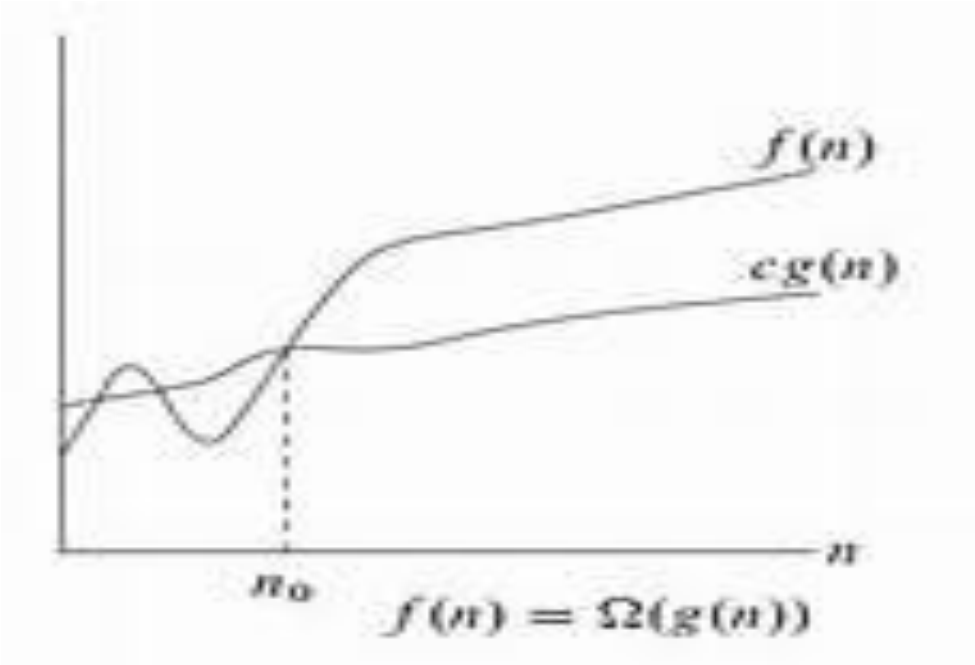
Let f(n) define the running time of an algorithm; f(n) is said to be Ω (g(n)) if and only if there exist positive constants c and n° such that:

$0 \leq c\, g(n) \leq f(n)$ for all $n \geq n°$.

EXAMPLE: f(n) = 2n+3

$2n+3 \geq 1*n$ $f(n)= \Omega (n)$

Now from increasing function always take less than part



3) Big theta (θ)

Let f(n) define the running time of an algorithm.

F(n) is said to be θ (g(n)) if f(n) is O (g(n)) and f(x) is Ω (g(n)) both.

Mathematically,

$$0 \le f(n) \le c_1 g(n) \quad \forall \quad n \ge n_o$$

$$0 \le c_2 g(n) \le f(n) \quad \forall \quad n \ge n_o$$

for sufficiently large value of n

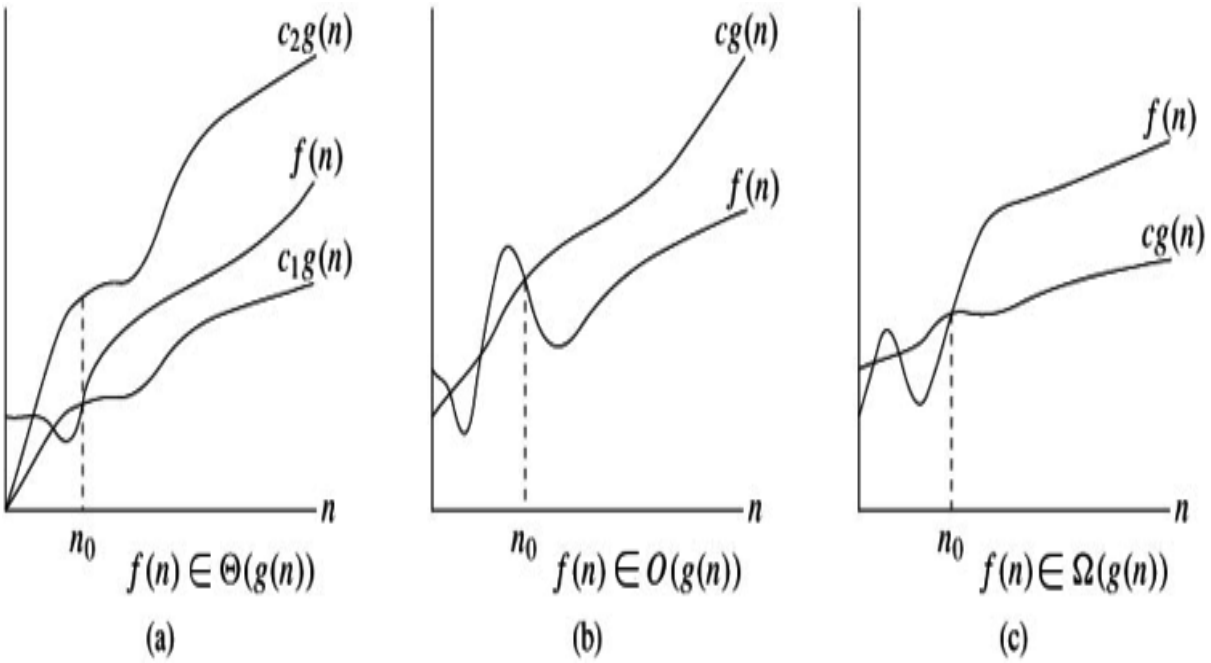
Merging both the equations, we get:

$0 \le c2\ g(n) \le f(n) \le c1\ g(n) \quad \forall \quad n \ge no.$

EXAMPLE: f(n) = 2n+3

$1n \le 2n+3 \le 10n$

$f(n) = \theta(n)$



Comparison of Functions

Method 1:

n	<	n^2	<	n^3
2		4		8
3		9		27
4		16		64

Method 2:

Applying log both side

log n^2 log n^3

2log n < 3log n

Logarithm Properties

$$\log_a xy = \log_a x + \log_a y$$

$$\log_a \frac{x}{y} = \log_a x - \log_a y$$

$$\log_a x^n = n \log_a x$$

$$\log_a b = \frac{\log_c b}{\log_c a}$$

$$\log_a b = \frac{1}{\log_b a}$$

The following can be derived from the above properties.

$$\log_a 1 = 0$$

$$\log_a a = 1$$

$$\log_a a^r = r$$

$$\log_a \frac{1}{b} = -\log_a b$$

$$\log_{\frac{1}{a}} b = -\log_a b$$

$$\log_a b \log_b c = \log_a c$$

$$\log_{a^m} a^n = \frac{n}{m}, m \neq 0$$

Best Worse and Average Case Analysis

Let's take an array

arr = [8 , 7, 4 , 3, 2 , 9 , 1, 5]

Best case: if you are searching for a key element which is present in first index or beginning index then that is best case.

In arr[8] is first index

Best case time taken = 1

$$B(n) = O(1)$$

Worst case: if you are searching for a key element which is present in last index then that is worst case.

Worst case time taken = n

$$W(n)= 0(n)$$

Average case: all possible case time / number of cases

This type of analysis is very difficult may not be possible for every algorithm. So rarely we do this.

Average tome = 1+2+3+4+5……..+n/n = n(n+1)/2n = n+1/2

A(n) = O(n+1/2)

Disjoint Sets Data Structure – Weighted Union and Collapsing Find

Disjoint sets are similar to sets topic of mathematics but not exactly. They are little bit change for making them useful in algorithm. So famous algorithm that uses disjoint set is kruskal's algorithm which detects a cycle in a graph so let us see how this disjoint is different from sets.

Union of set: two and more set in a set containing all the element of given set, i.e. A ∪ B,

Intersection of set: the set of all those elements which are common to both A and B. A ∩ B

If you take an edge and both the vertices are belonging to the same set then there is cycle in the graph.

Divide and conquer

It is related to a strategy for solving a problem like we have other strategies also in the subject 3D – method, dynamic programming, backtracking, branch and bound etc. Strategy is an approach or a design for solving any problem like computational problems. If the problem is size n then we break problems into sub problems like P1, P2, P3……Pk etc. they can be solved to obtain their solution and can be solved individually and after that you can sum them and get the solution for main problem. If the sub problem is large then we can apply divide and conquer in them also. The main part about divide and conquer whatever the problem is the sub problems will be the same. It is recursive in nature. General method of divide and conquer:

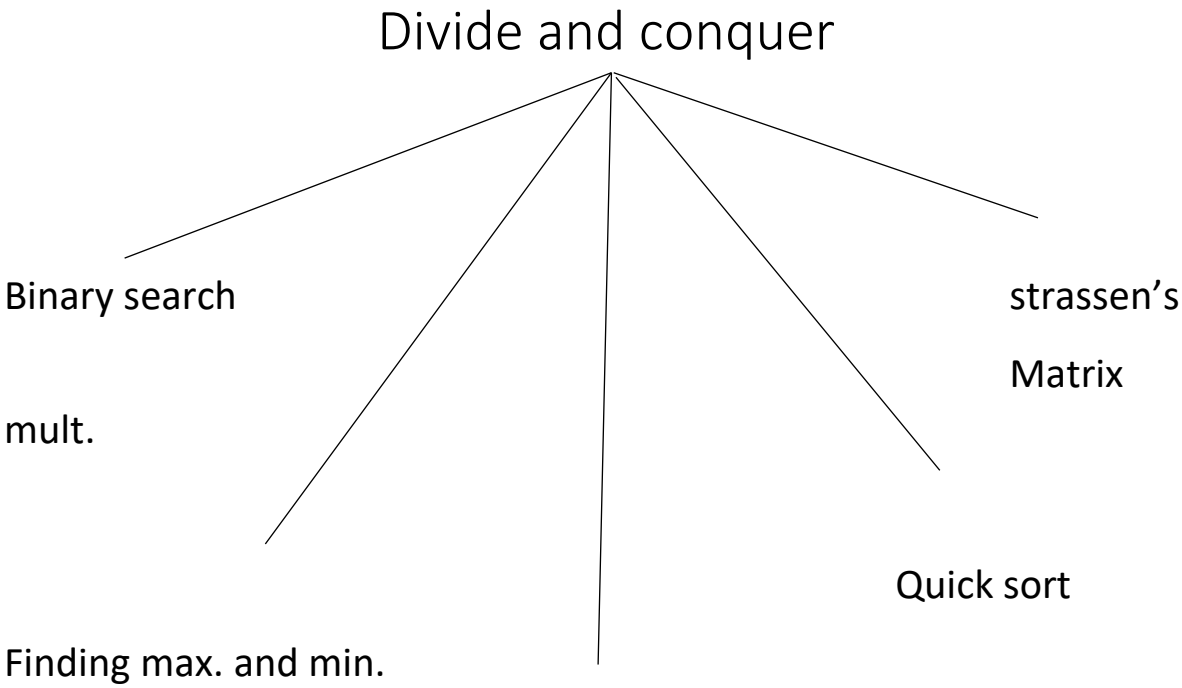
```
DAC (P)
{
    If (small(P))
      {
          S(P);
      }
    Else
      {
          Divide P into P1,P2,P3……..Pk
```

Apply DAC(P1),DAC(P2),…….

Combine (DAC(P1),DAC(P2),……..)

}

}



Reccurence relation

(T(n) = T(n-1) +1)

T(n) = void test (int n)

1 = printf("%d",n)

T(n-1) = test(n-1)

```
Void test (int n )
{
  If (n>0)
  {
     Printf("%d",n);
     Test(n-1);
   }
}
```

Reccurence relation

(T(n) = T(n-1) + n)

```
Void test (int n)
{
  If (n>0)
   {
      For(i=0;i<n;i++)
       {
           Printf("%d',n);
       }
        Test(n-1);
   }
}
```

T(n) = T(n-1) +n

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + n & n > 0 \end{cases}$$

T(n) = T(n-1) +n

T(n) = T(n-1) +n

T(n-1) = T(n-2) + n-1

T(n) = [T(n-2) + n-1]+n

T(n-2) = T(n-3) + n-2

T(n) = T(n-2) + (n-1) +n

T(n) = [T(n-3) + n-2] + (n-1) +n

T(n) = T(n-3) + (n-2) + (n-1) + n

T(n) = T(n-k) + (n-(k-1)) + (n-(k-2))+…….+(n-1) +n

Assume

n-k =0

n=k

T(n)= 1 + n(n+1)/2

Recurrence relation

T(n) = 2T(n-1)+1

```
Algorithm Test(int n)
{
    If (n>0)
    {
        Printf("%d",n);
        Test(n-1);
        Test(n-1);
    }
}
```

Master Theorem for Decreasing Function

General form of recurrence relation is:

T(n) = aT(n-b) + f(n)

a>0 , b>0 and f(n) = n^k where K>= 0

case 1: if a=1

n^k+1

n*f(n)

case 2 : if a<1

n^k

f(n)

case 3 : if a>1

n^k a^n/b

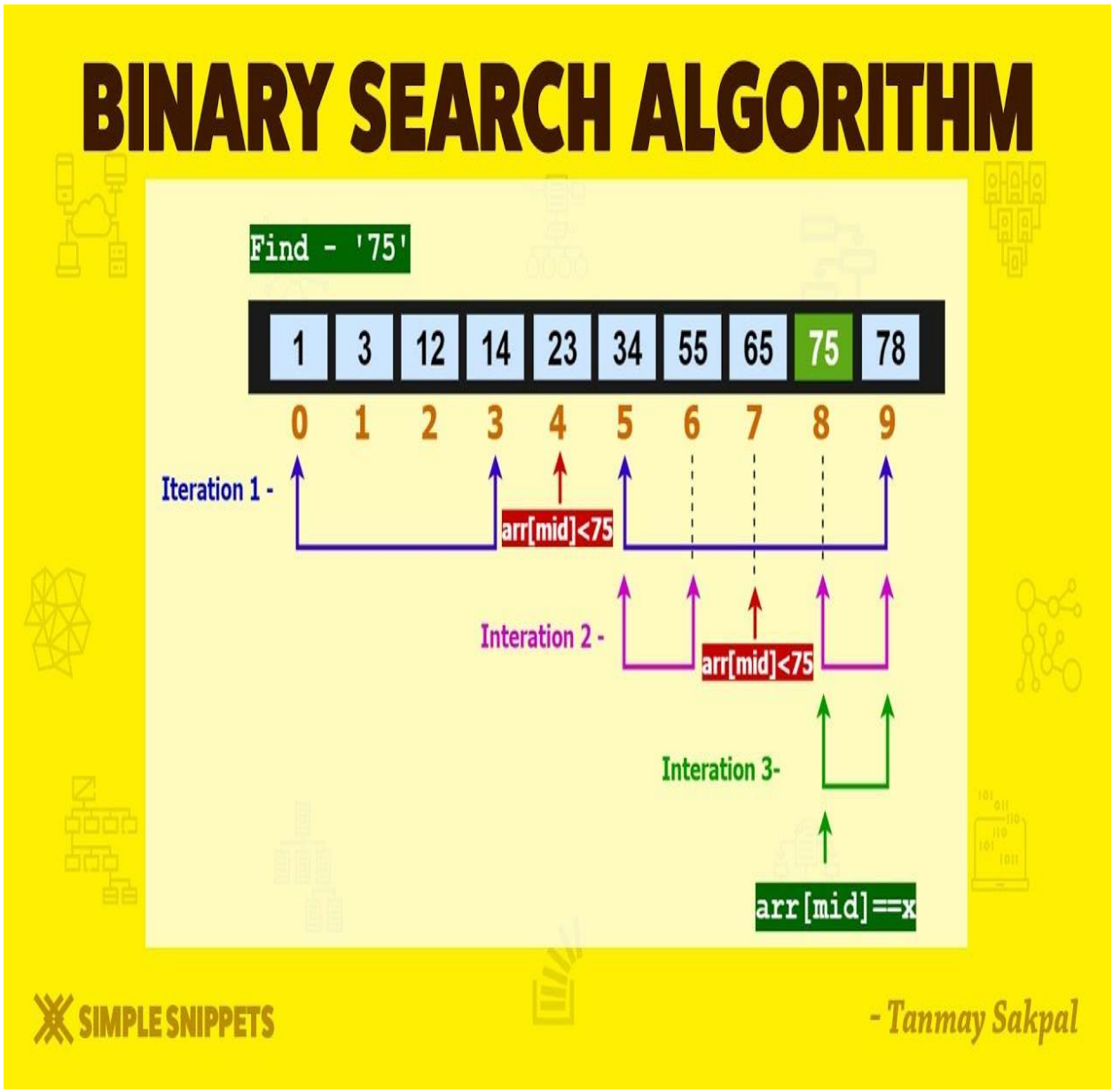
Root Function

In mathematics and computing, a root-finding algorithm is **an algorithm for finding zeros, also called "roots", of continuous functions**. A zero of a function f, from the real numbers to real numbers or from the complex numbers to the complex numbers, is a number x such that f(x) = 0

```
T(n)----------void test ( int n )
                    {
                        If(n>2)
                         {
        1-------------------------stmt;
T(√n)-------------------------test(√n);
                         }
                     }
```

Binary Search

Searching technique surrounding a method for such there are two different method one is linear search and binary search. It follows divide and conquer strategy.



If the low point cross the high point then the key value we are searching for is not in the given sequence.

Algorithm for binary search

```
Int  BinSearch(A,n,key)
{
     L=1;
     h=n;
   while(l <= h)
   {
        mid = ( l + h )/2;
        if(key == A[mid])
             return mid;
        if(key<A[mid])
              h = mid -1;
        else
               l = mid+1;
    }
  Return 0;
}
```

Recursive algorithm for binary search

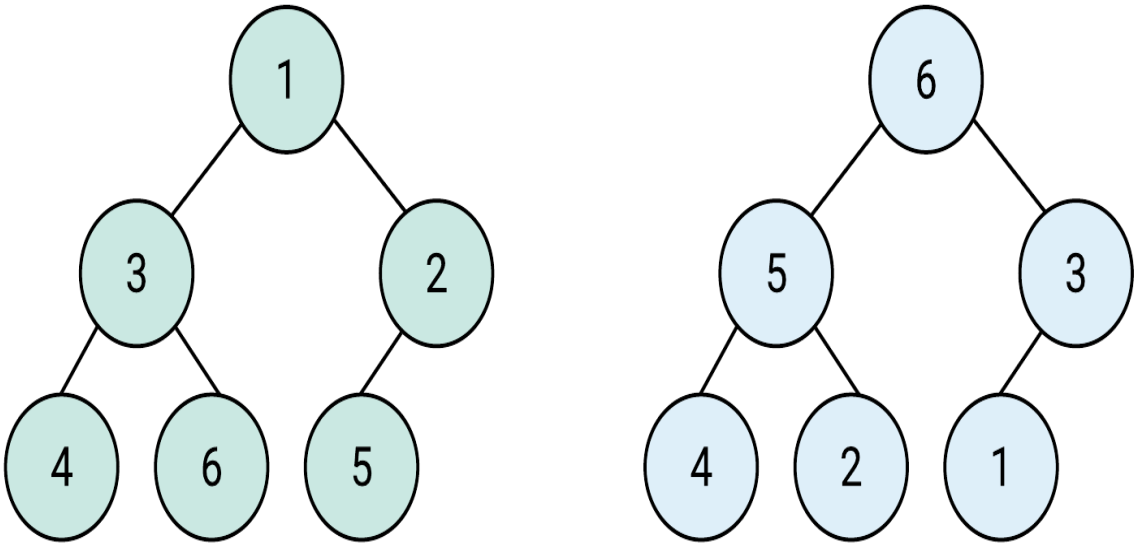
```
T(n)----------algorithm RBinSearch(l,h,key)
                {
                  If( l ==h)
                  {
                    If(A[l]==key)
                      Return l;
                    Else
                      Return 0;
                  }
                 Else
                 {
1----------mid = (l + h)/2;
1----------if(key == A[mid])
                      Return mid;
1----------if(key <A[mid])
                      Return RBinSearch(l,mid-1,key);
                    Else
                      Return RBinSearch(mid+1,h,key);
                  }
                }
```

Heap – heap sort – heapify

Heap is the complete binary tree. There are two types of heaps , maxheap and minheap

Max heap : every node is having the value greater than all its descendants then we will have largest value that is maximum value. So if the elements are arranged in such a manner then it is called max heap.

Minheap : every node having the value smaller or equal to all its descendants then it is called minheap.



Min heap

Max Heap

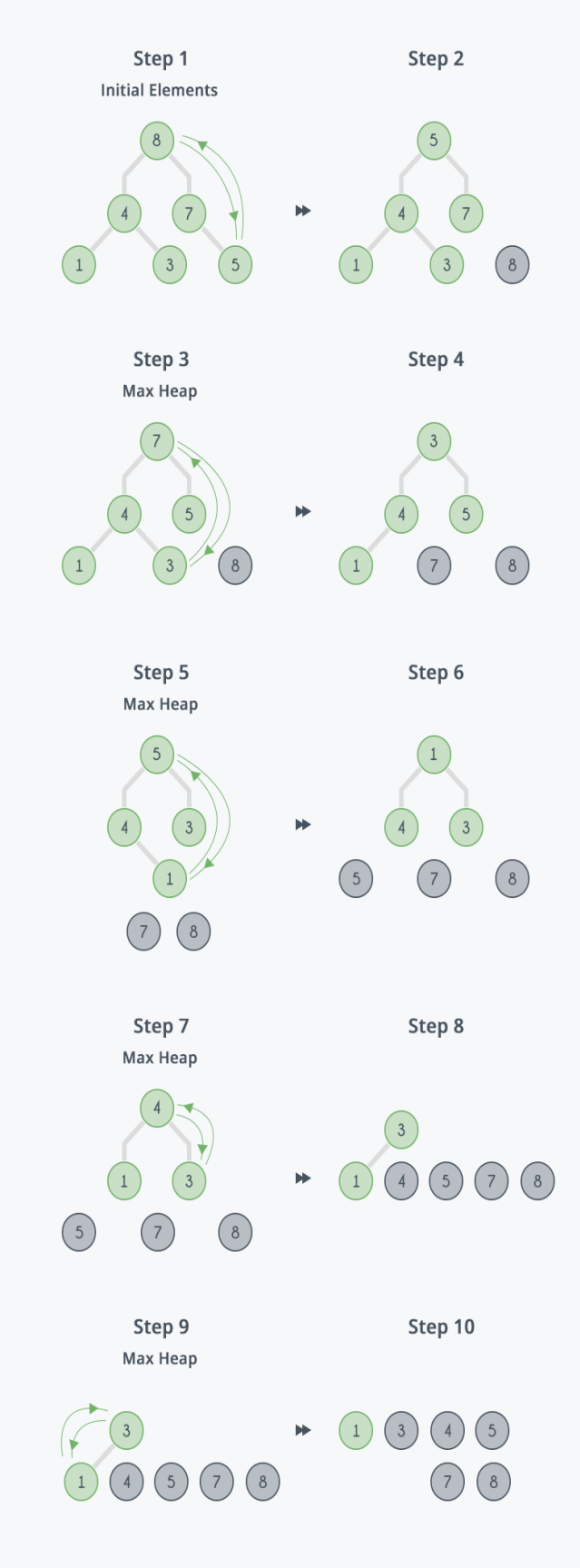
If you have a heap that delete the element and fill it in the empty places obtained after deletion so if go on filling the elements there then automatically gets sorted. So from the heap go on deleting the elements and start filling them in free spaces known as heap sort.

Heap sort have two steps :

1.for a given set of elements create a heap by inserting all the elements from the heap one by one.

2.then once the heap is formed delete all the elements from the heap one by one the elements will get sorted.

How to solve heap is contains some steps shown below:



Heapify is a process of creating a heap **The process of reshaping a binary tree into a Heap data structure** is

known as 'heapify'. A binary tree is a tree data structure that has two child nodes at max. If a node's children nodes are 'heapified', then only 'heapify' process can be applied over that node. A heap should always be a complete binary tree.

Merge Sort

Merging is a process of combining two sorted lists into a single sorted list.

A	B	C
2	5	2
8	9	5
15	12	8
18 -i	17 -j	9
		12
		15
		17
		18 -k

m n

```
Algorithm merge (A,B,m,n)
{
      I =1; j =1; k=1
       While( i<=m && j<=n)
       {
            If(A[i]<B[j])
                  C[k++] = A[i++];
             Else
                  C[k++] = B[j++];
       }
      For(;i<=m;i++)
        C[k++] = A[i];
      For (;j<=n;j++)
        C[k++] = B [j];
}
```

```
Algorithm mergesort(l , h)
{
    If(l>h)
      {
            Mid = ( l+ h)/2;
            Mergesort(l,mid);
            Mergesort(mid+1,h);
            Merge(l,mid ,h);
       }
}
```

Pros and cons of merge sort:

Pros of merge sort –

1 large size list

2 linked list

3 external sorting

4 stable

Cons of merge sort –

1 extra space (not inplace sort)
2 no small problem
3 recursive

Quick sort

Quicksort is **a popular sorting algorithm that is often faster in practice compared to other sorting algorithms**. It utilizes a divide-and-conquer strategy to quickly sort data items by dividing a large array into two smaller arrays. Example of quick sort:
Comparing 44 to the right-side elements, and if right-side elements are smaller than 44, then swap it. As 22 is smaller than 44 so swap them. 22 33 11 55 77 90 40 60 99 44 88. Now comparing 44 to the

left side element and the element must be greater than 44 then swap them.

Quick sort algorithm :

```
Quicksort (l ,h)
{
      If(l < h)
       {
         J = partition ( l,h );
         Quicksort( l , j);
          Quicksort( j+1,h);
       }
}
```

```
Partition(l,h)
{
      Pivot = A[l];
      I = l; j = h;
       While(I<j)
        Do
      {
        I++;
      } while(A[i] <= pivot);
       Do
       {
         j--;
       } while(A[j]>pivot);
        If(i<j)
             Swap(A[i] , A[j]);
     }
Swap(A[l],A[j]);
Return j;
 }
```

Strassen’s matrix multiplication

It belongs to divide and conquer so it follows divide and conquer strategy. Strassen's Matrix multiplication **can be performed only on square matrices where n is a power of 2**. Order of both of the matrices are n × n. algorithms of strassen’s matrix multiplication.

$$P_1 = A_{11} \cdot (B_{12} - B_{22})$$
$$P_2 = (A_{11} + A_{12}) \cdot B_{22}$$
$$P_3 = (A_{21} + A_{22}) \cdot B_{11}$$
$$P_4 = A_{22} \cdot (B_{21} - B_{11})$$
$$P_5 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$
$$P_6 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$
$$P_7 = (A_{11} - A_{21}) \cdot (B_{11} + B_{12})$$

$$P_5 + P_4 - P_2 + P_6 = C_{11}$$
$$P_1 + P_2 = C_{12}$$
$$P_3 + P_4 = C_{21}$$
$$P_5 + P_1 - P_3 - P_7 = C_{22}$$

Example of strassen’s matrix multiplication:

Algorithm 3 Strassen's Matrix Multiplication Algorithm

Input: $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ and $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \in \mathbb{R}^{n \times n}$

1: **if** $n = 1$ **then**
2: $C = A \cdot B$
3: **else**
4: $M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$
5: $M_2 = (A_{21} + A_{22}) \cdot B_{11}$
6: $M_3 = A_{11} \cdot (B_{12} - B_{22})$
7: $M_4 = A_{22} \cdot (B_{21} - B_{11})$
8: $M_5 = (A_{11} + A_{12}) \cdot B_{22}$
9: $M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$
10: $M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$
11: $C_{11} = M_1 + M_4 - M_5 + M_7$
12: $C_{12} = M_3 + M_5$
13: $C_{21} = M_2 + M_4$
14: $C_{22} = M_1 - M_2 + M_3 + M_6$

Output: $A \cdot B = C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \in \mathbb{R}^{n \times n}$

GREEDY METHOD

This is one of the strategy for solving Problem, just like divide and conquer and the other strategies. Greedy Method is one of the approaches for solving a Problem or a design which you can adopt for solving similar problems, the problems which fits into this one you can solve all of them. This method is also used for solving optimization problem.

For a problem there are many solutions but these solutions which are satisfying the condition given in the problem then these types of problems become Feasible Solutions though for a given problem there may be many solutions.

When the problem demands our result should be minimum then it is a Minimization Problem.

If a problem requires either minimum or maximum then we call that type of problem as Optimization Problem.

Strategies Used for Solving Optimization Problem

- **GREEDY METHOD**
- **DYNAMIC PROGRAMMING**
- **BRANCH AND BOUND**

KNAPSACK PROBLEM

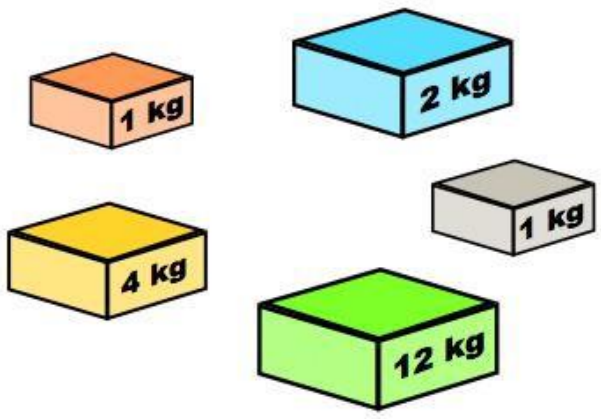
Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

n=7	Objects (O)	1	2	3	4	5	6	7
m=15	Profits (P)	10	5	15	7	6	18	3
	Weights (W)	2	3	5	7	1	4	1
	P/W	5	1.3	3	1	6	4.5	3

We have to fill these objects such that the profit is maximized.

So this is Maximization problem and Optimization problem.

It can follow Greedy method.



Job sequencing with deadlines

1. There are n jobs, Jobs j1, j2, j3….jn and deadline for each d1, d2…dn and profit for each are p1, p2…pn.

2. Profit will only be awarded if job is completed on or before deadline.

3. Assume each job takes unit time to complete.

4. Objective is to get maximum profit by selecting one job at one time.

In job sequencing problem, the objective is to find a sequence of jobs, which is completed within their deadlines and gives maximum profit.

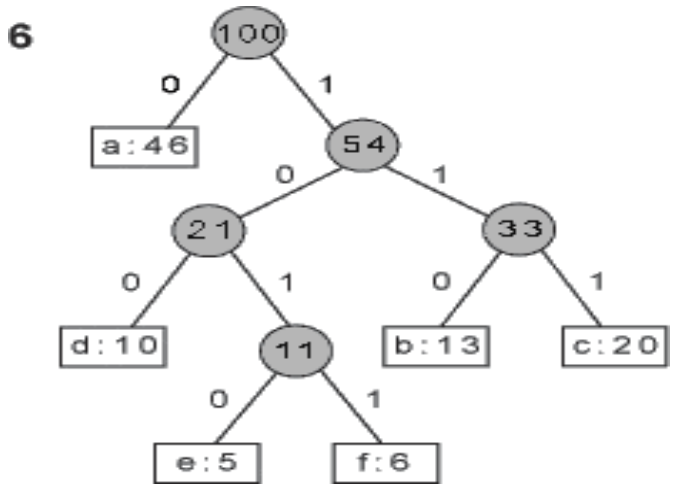
Jobs	J1	J2	J3	J4	J5
Deadlines	2	2	1	3	3
Profits	20	15	1	5	10

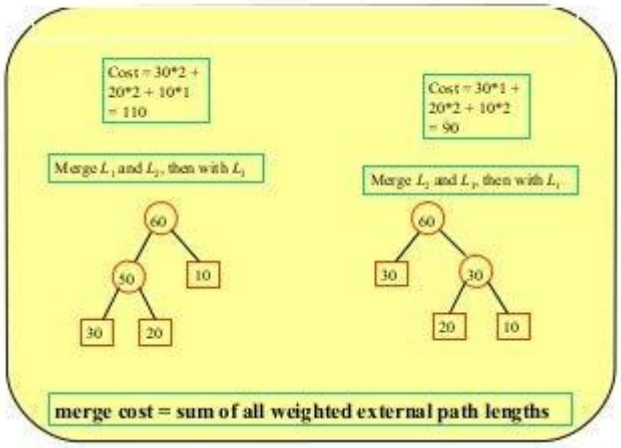
Optimal Merge Pattern

- Optimal merge pattern is a pattern that relates to the merging of two or more sorted files into a single sorted file.
- This type of merging can be done by Two Way Merging Method.
- If we have two sorted files containing ‘m’ and ‘n’ records then they could be merged together to obtain one sorted file in time O (m + n).
- When more than two sorted files are to be merged together, it can be done by repeatedly merging sorted files in pairs.

Thus if files x_1, x_2, x_3, x_4 are to be merged we can follow different pairing methods.

METHOD 1	METHOD 2
Merge x_1 and x_2 to get a file y_1.	Merge x_1 and x_2 to get a file y_1.
Merge y_1 and x_3 to get y_2.	Merge x_3 and x_4 to get a file y_2.
Merge y_2 and x_1 to get a desired sorted file.	Merge y_1 and y_2 to get a desired sorted file.





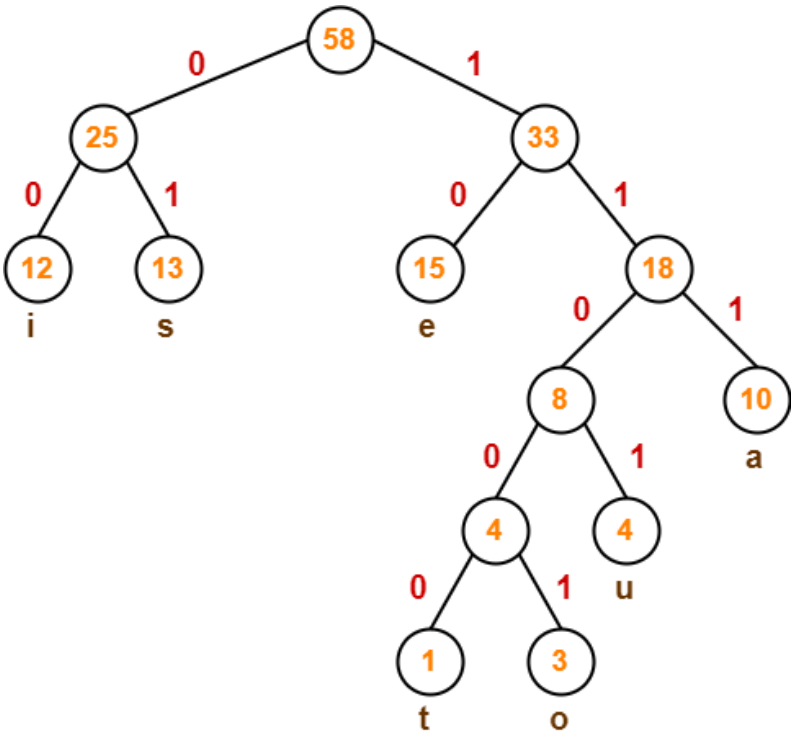
Different pairings require different amounts of computing time.

HUFFMAN COADING

Huffman Coding Algorithm- a bottom-up- approach.

The Huffman coding is a procedure to generate a binary code tree. The algorithm invented by David Huffman in 1952 ensures that the probability for the occurrence or every symbol results in as code length.

Huffman coding could perform effective data compression by reducing the amount of redundancy in the coding of symbols.



Huffman Tree

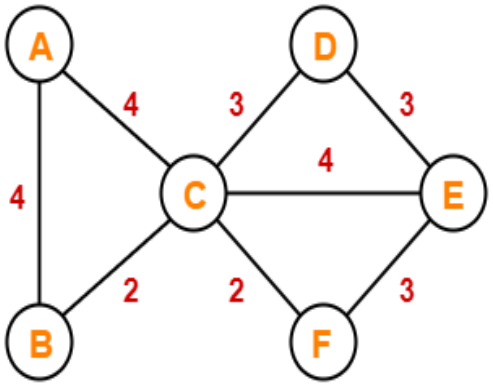
Prisms and Kruskals Algorithm

Prisms Algorithm

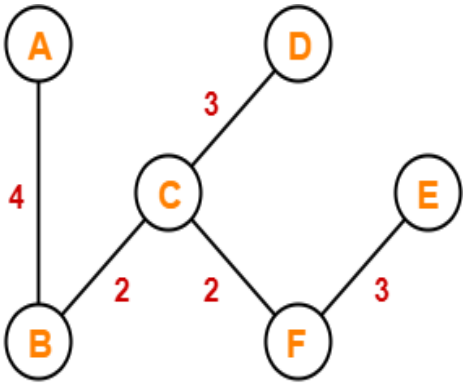
- It starts to build the MST from any of the node.
- Adjencary Matrix, Binary heap or Fibonacci heap is used in Prisms algorithm.
- Prisms Algorithm runs faster in dense graphs.
- Time complexity is O (EV log V) with binary heap and O (E + V log V) with Fibonacci heap.
- The next node included must be connected with the node we traverse.
- It traverses the one node several times in order to get it minimum distance.
- Greedy Algorithm.

Kruskals Algorithm

- It starts to build the MST from minimum weighed vertex in the graph.
- Disjoint set is used in Kruskals Algorithm.
- Kruskals Algorithm runs faster in sparse graphs
- Time complexity is O (E log V)
- The next edge includes may or may not be connected but should not form the cycle.
- It traverses the edge only once and based on cycle it will either reject it or accept it.
- Greedy Algorithm.

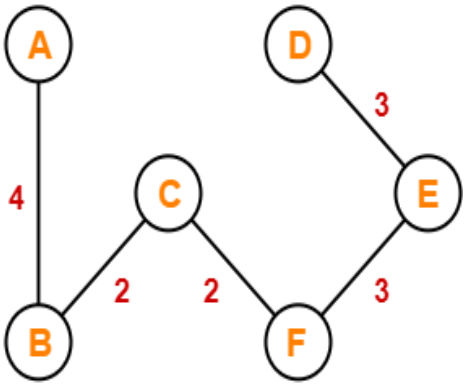


Given Graph



Result from Prim's Algorithm

(Cost = 14 units)

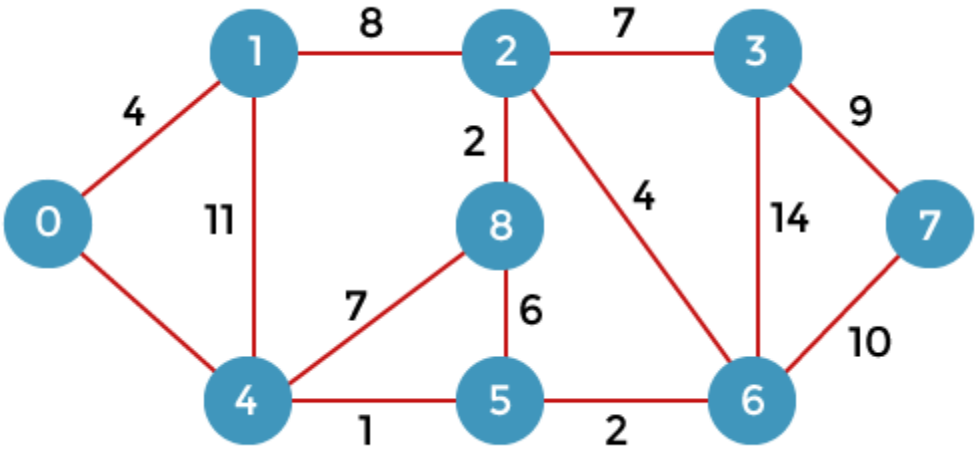


Result from Kruskal's Algorithm

(Cost = 14 units)

Dijkstra Algorithm

Dijkstra's Algorithm can be formally described by the given
the following definitions:
N= set of nodes in the network
s = source node
T = set of nodes so far incorporated by the algorithm w (i) = link cost from node i to node j; w (i, i) = 0; w (ij) = ∞ if two nodes not directly connected; wi, y) ≥ O if two nodes are directly
connected
I (n) = cost of the least-cost path from node s to node n that is currently known to the algorithm; at termination, this is the cost of the least-cost path in the graph from s to n.
The algorithm terminates when all nodes have been added to T.



Principle of Optimality

Greedy method	Dynamic programming
In this we try to follow predefined procedure to get optimal result. the procedure is known to be optimal. We follow procedure to get result like critical method for finding minimum cost spanning tree. Always select a minimum cost edge and that gives us best result or Dijkstra shortest path algorithm always select the shortest path vertex and continue relaxing the vertices so you get a shortest path so there is predefined procedure.	In this we'll try to find out all possible solutions and then pick up best solution the optimal solution this approach is different and is little time consuming compared to greedy method for any problem there may be many solutions which are feasible so we'll try out all those solutions and then pick up best one and mostly dynamic programming problems are solve by using the recursive formulas. though we will not use recursion of programming but the formulas are recursive if we want we can use recursion also but mostly they are solved using iteration and it follows principle of optimality says that a problem can be solved by taking sequence of decisions.

Multi-Stage Graph

Multi-stage graph is a directed weighted graph and the vertices are divided into two stages such that the word adjusts our connecting vertices from one stage to next stage only. first stage and the last stage will have just single vertex to represent the starting point that is source or ending point or sink of a graph this usually useful for representing resource allocation. so here there are various part from source to sink we have to select the path which is giving me minimum cost this is the objective of the problem. It is minimization problem that is optimization problem that problem can be solved using dynamic programming.

All pairs shortest path

We will solve this by using dynamic programming approach. It is similar to Dijkstra algorithm but Dijkstra's algorithm will find out a shortest path from one of source vertex. We can also solve by dynamic programming. It says that the problem

be solved by taking sequence of decision in each stage we will take a decision.

Matrix Chain Multiplication

If we have two matrices and we want to multiply (A*B and A'*B') then B = A' (example 4*5 and 5*6)

Given two matrices A = [ai j] 1≤i, j≤n, B = [bi j] 1≤i, j≤n it is required to calculate the matrix product C = A·B.

By definition we have

$\forall i, j \in [[1,n]] : c_{ij} = n \sum k=1 a_{ik} b_{k\,j.}$

Dynamic programming approach says that you should try all possibility of matrix and pick up the best one.

Bellman Ford Algorithm

The problem is in a directed weighted graph we have to select one of the vertices as source vertex and find out the

shortest path to all other vertices. let first we have to select one vertex and then find shortest path of all other vertices. we also have Dijkstra algorithm to find shortest distance but in this we can shortest distance even if there are negative edges. Dijkstra algorithm may give correct result or not so we cannot use Dijkstra algorithm in presence of negative edges. It follows dynamic programming strategy it says that try all possible solution and pick up the best solution. This algorithm says that you go on relaxing all the edges. Repeatedly relax them for N-1 time.

0/1 Knapsack

In this the capacity of element is given and you have to fill the element in the capacity in the way you have to fill the objects such that the total profit is maximized and we have to give the solution in the form of a set for each object as like which is included or not included in the form of 0/1

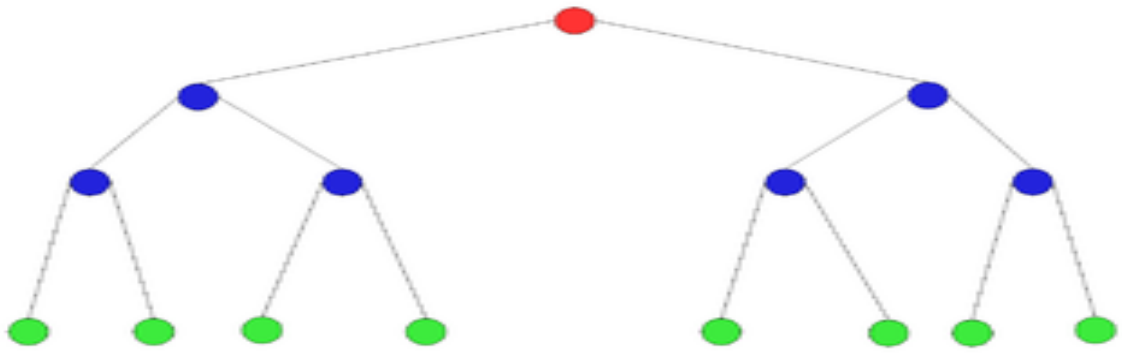
Optimal Binary Search Tree

Any element is given to you convert it in tree form. In given set 2*nCn/n+1 binary search trees are possible. Less height of

balance search tree more balance. More height binary tree means frequencies of that binary tree is minimum. Number of comparisons is required for searching any element in binary search tree depends on number of levels in a binary search tree. If key element is not found then search is unsuccessful if found then successful. We prefer that binary tree which require less number of comparison.

For any pair of indices $i \leq k$, let F(i, k) denote the total frequency count for all the keys in the interval A[i .. k]:

$$F(i, k) := \sum f[j] \quad \text{where } j=I \text{ to } k.$$



Reliability design

What is the chances that it works perfectly is called reliability. Each device is having some reliability.

EXAMPLE: we have to set up a studio we need devices camera light etc.

D1	D2	D3	D4	---------> DEVICES
C1	C2	C3	C4	---------> COST
R1	R2	R3	R4	---------> RELIABILITY
0.9	0.9	0.9	0.9	

So, reliability of entire system is product of all reliabilty

$= 0.9^4 = 0.6561$

Reliability of each device is 0.9 and entire is 0.65 means there is 35% of chances of that system may fail. For not getting fail we have multiple copies of this devices.

Longest Common Subsequence (LCS).

EXAMPLE:

String1: A B C D E F G H I J

String2: E C D G I

String1: A B C D E F G H I J

String2: E C D G I

By first: E G I

By second: C D G I this is **Longest Common Subsequence**

Branch and bound

This is one more problem solving a method by which we can solve a problem. This is similar to backtracking in the sense that it also uses a state – space steam for solving the problem solution is represented like a state space tree but it is useful for solving optimization problem and only minimization problem not maximization problem. We can convert maximization problem into minimization problem. There are many methods such as least cost branch bound , job sequencing etc.

Job Sequencing with Deadline

Example:

Jobs:	1	2	3	4
Penalty:	5	10	6	3
Deadline:	1	3	2	1
Time:	1	2	1	1

U=sum of all penalties except that included in solution.

C^= sum of all penalties till the last job considered

S = set of jobs that are included in solution

U =

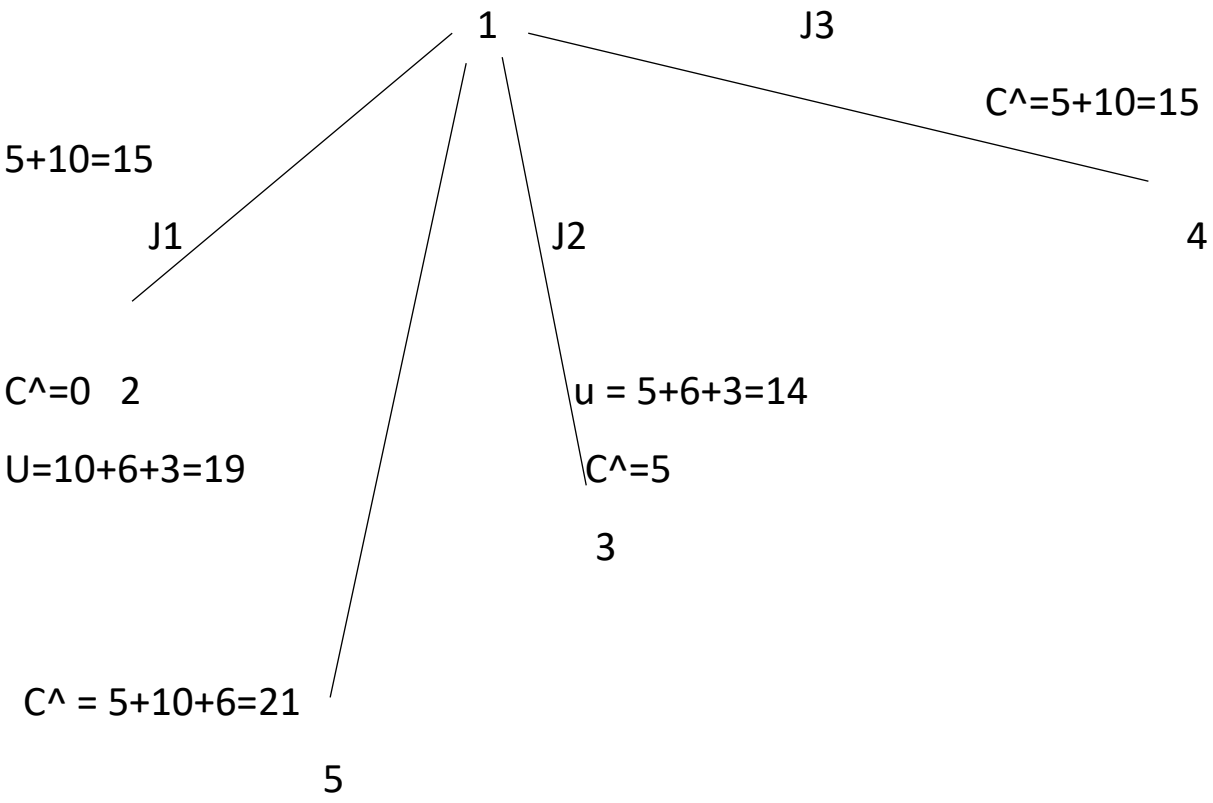
$$\sum_{i \in s} pi$$

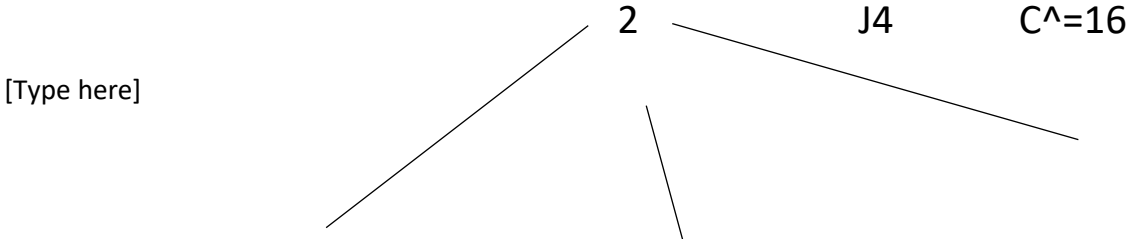
C =

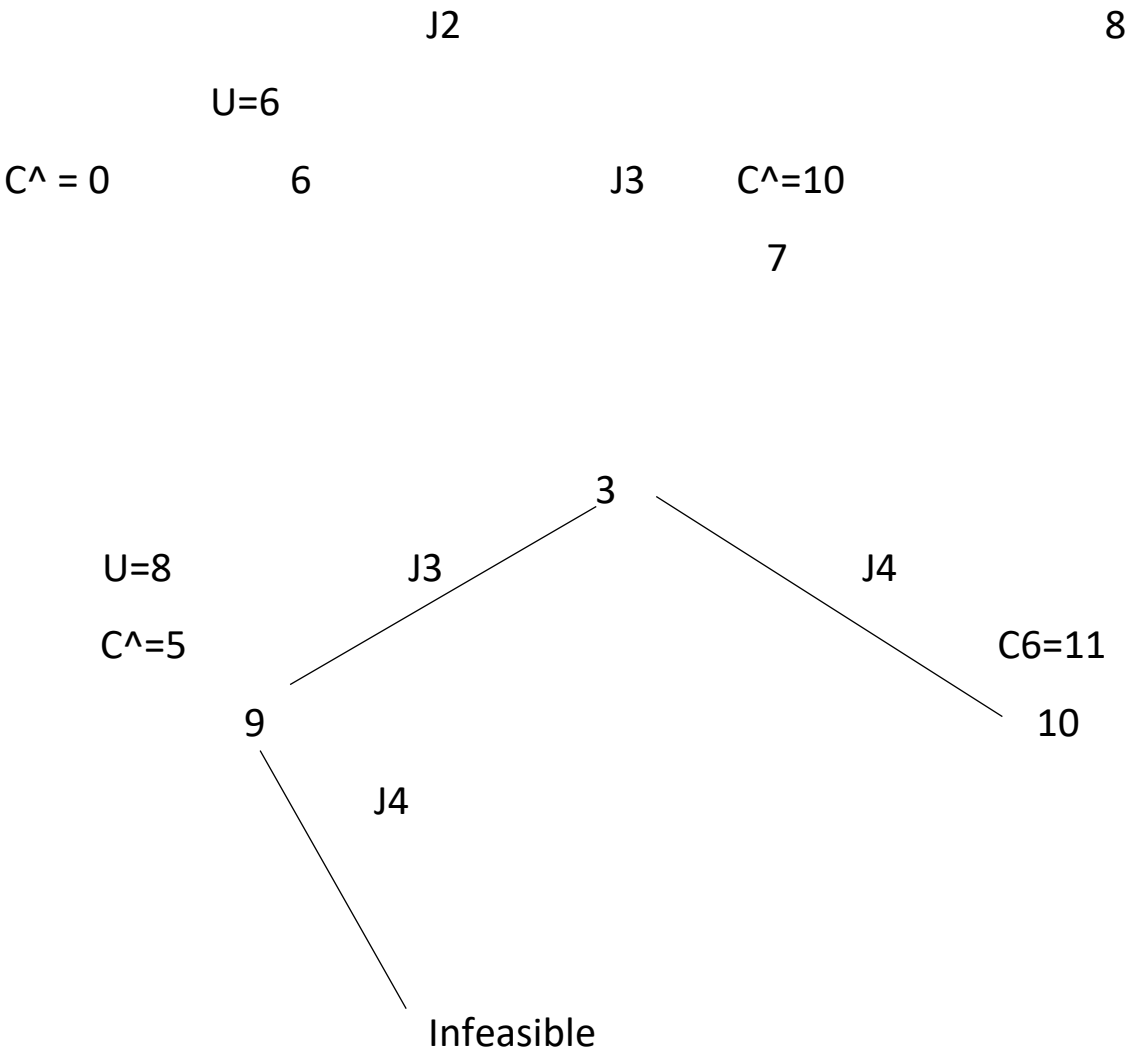
$$\sum_{i \in s} pi$$

U = ∞

C^ = 0







Cost = 5

Penalty = 8

The jobs that will be doing job 2 and job3, then the penalties that will be saving are 10 and 6 which is 16. Job1 and job4 are not done their penalties are 5 and 3 which is 8.

0/1 knapsack

Example :

The objective of the problem is to fill the bags with those objects such that the total weight of the objects included in the bank should be less than or equal to the capacity

	1	2	3	4
Profit:	10	10	12	18
Weight:	2	4	6	9

M=15(bag of capacity 15)

N=4

Using LC-BB

U=

$$\sum_{i=1}^{n} pixi \qquad \leq m \quad (without\ fraction)$$

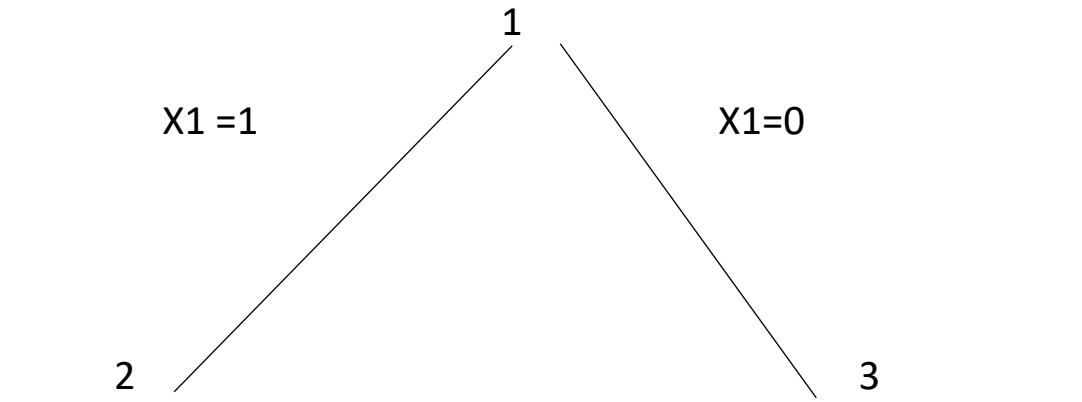
C=

$$\sum_{i=1}^{n} pixi \qquad (with\ fraction)$$

U=-32

Cost = -38 (10+10+12+{18/9*3})

UPPER = ∞



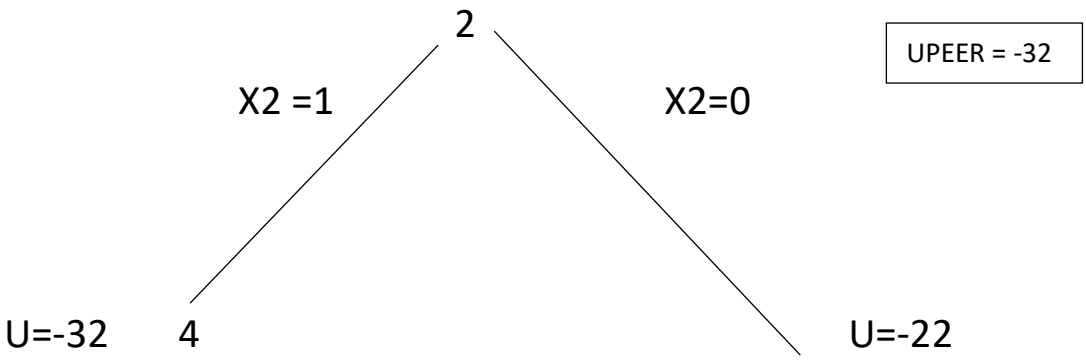
U=-22

U=-32

C= -32

C=-38

UPEER = -32

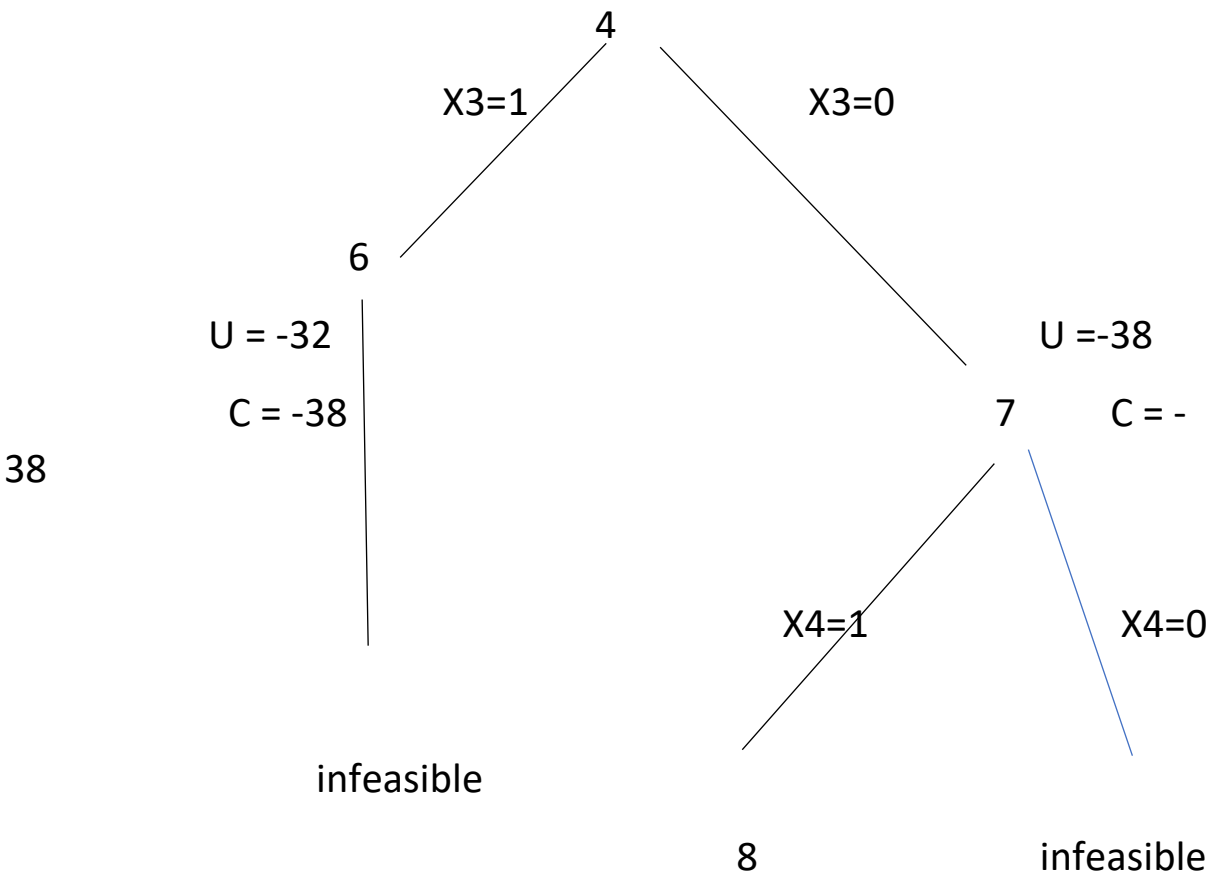


U=-32

U=-22

C = -38 5 C=-36

UPPER = -38



U = -38

C =-38

X = {1,1,0,1}

Maximum profit = 38

2+4+0+9 = 15 (m)

NP-hard and NP-complete

Polynomial Time	Exponential Time
Linear search - n	0/1 knapsack – 2^n
Binary search – logn	Traveling SP – 2^n
Insertion sort - n^2	Sum of Subsets – 2^n
Merge sort – nlogn	Graph coloring – 2^n
Matrix multiplication – n^3	Hamiltonian cycle -2^n

Non – Deterministic Algorithms

By writing non – deterministic algorithms we can preserve our research work so that in future somebody else is working on this same problem he/she can make non-deterministic park deterministic.

Example of non –deterministic algorithms

Algorithms NSearch (A,n,key)

{

j = choice (); → Non - deterministic

if(key = A[j])

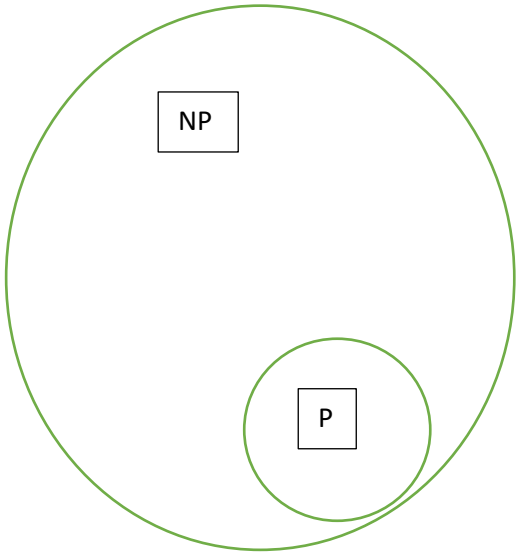
```
    {
       Write (j);
       Success();
    }
    Write(0);
    Failure();
}
```

Non - deterministic

Non- deterministic

As per exponential time we define two classes such as verifying P. P is a set of those deterministic algorithms which are taking polynomial time for example linear search , binary search etc. all this take polynomial time and the algorithms are deterministic we call them as P.

Another class we have is NP, these algorithms are non – deterministic but they take polynomial time. We convert exponential into polynomial but we don't know how it is possible so we leave the statement non – deterministic so the algorithms become non – deterministic



P is the subset of NP it means the deterministic part of the algorithms that we know today is the part of non – deterministic.

Or we can say that deterministic is the subset of non-deterministic algorithms.

We can't solve exponential time problems so at least we can relate them with the base problem and base problem is CNF-Satisfiability.

If satisfiability solve in polynomial time all exponential time problems can be solve by polynomial time.(satisfiability =2^n)

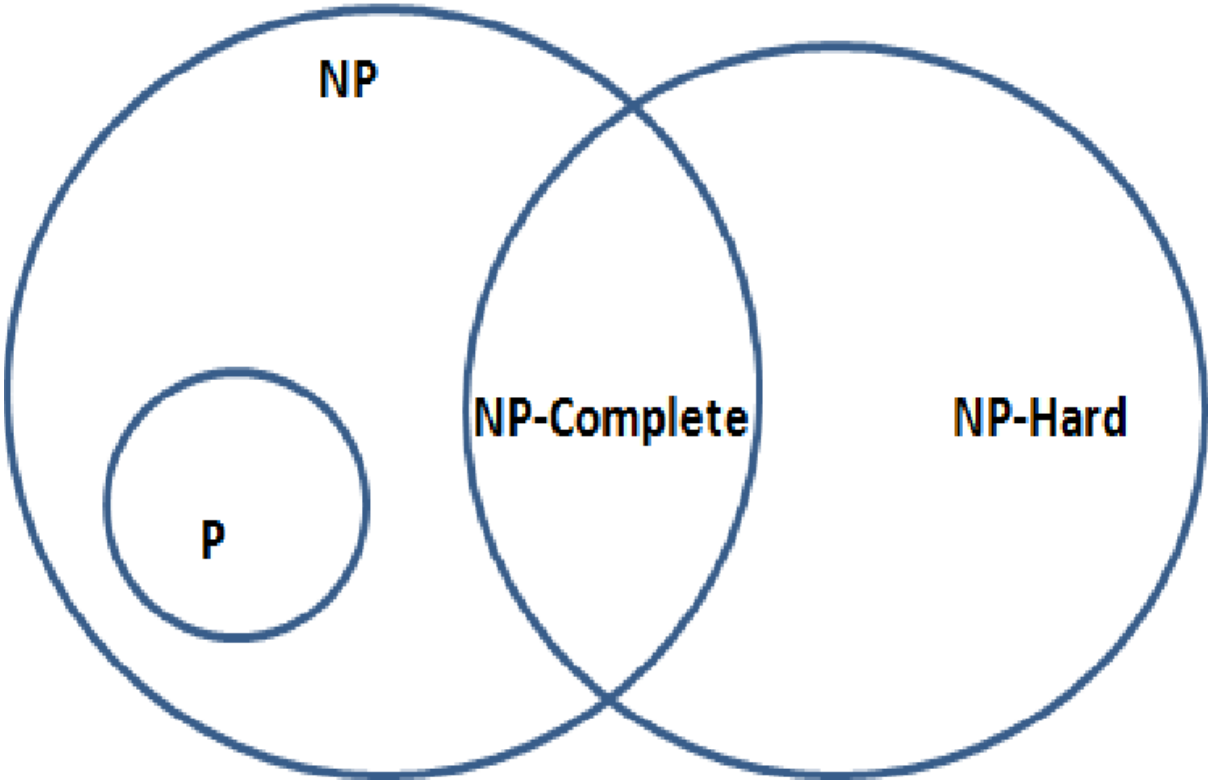
Let us called satisfiability NP-hard. If we reduce satisfiability to some problem like 0/1 knapsack the conversion time is also a polynomial time. The process is called reduction.

Reduction has a transitive property if satisfiability reducing lower exponential time problem. If **satisfiability is reducing to some problem the problem is also becoming NP-Hard.**

Non – deterministic polynomial time algorithm existing, satisfiability is NP- hard as well as NP-complete. If any problem is having non – deterministic time algorithm then that problem also become NP- complete.

Now, diagramactic representation

The p will keep on expanding and become bigger and bigger so that non – deterministic will become deterministic.



To prove non – deterministic algorithm will be converted into polynomial time deterministic algorithm

If we are able to prove that P = NP then it is prove that whatever the non- deterministic we are writing is may become deterministic.

There is cook's theorem , he tried prove that if satisfiability is lying in P if is only if P = NP.

Knuth-morris-pratt (KMP) algorithm

KMP algorithm is used for pattern matching if a string is given and a pattern is given then the problem is to find out whether this pattern is existing inside the string or not if it is there then find out its index. There is also an algorithm nave algorithm. Example:

Pattern : a b c d f

String : a b c d a b c a b c d f

The characters in the above string are compared repeatedly whenever there is a mismatch it is shifting , backtracking .

Now taking another example :

Pattern = abc d abc

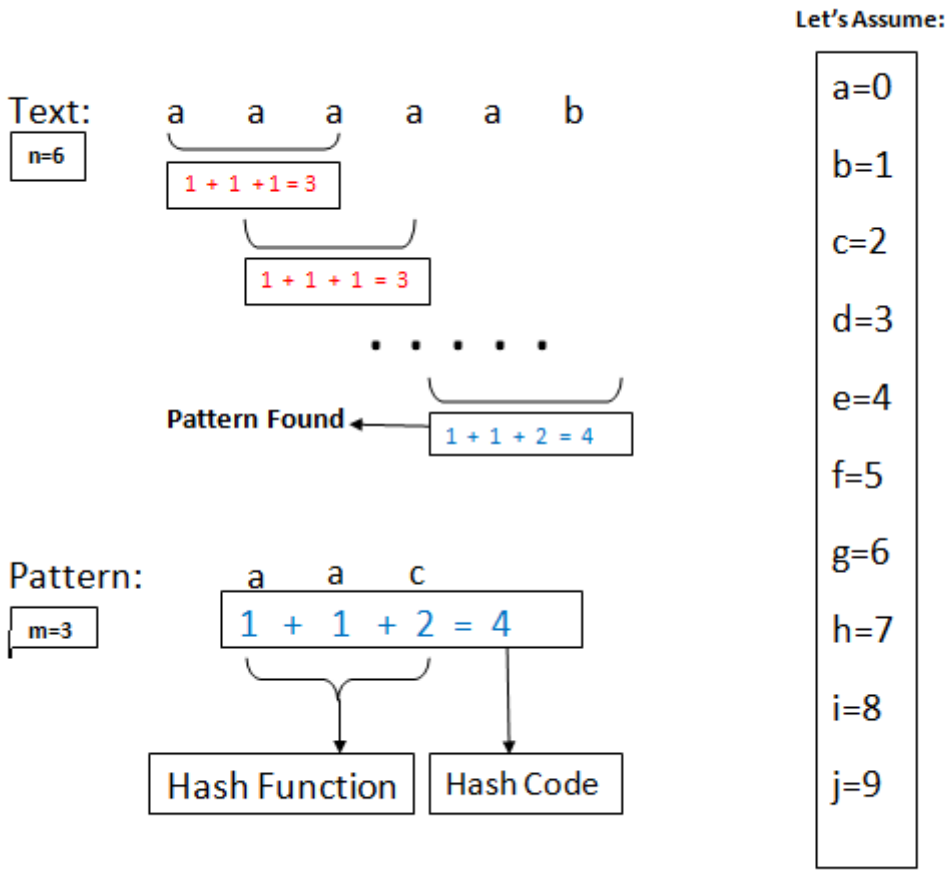
Prefix = a , ab ,abc, abcd

Suffix = c, bc , abc , dabc

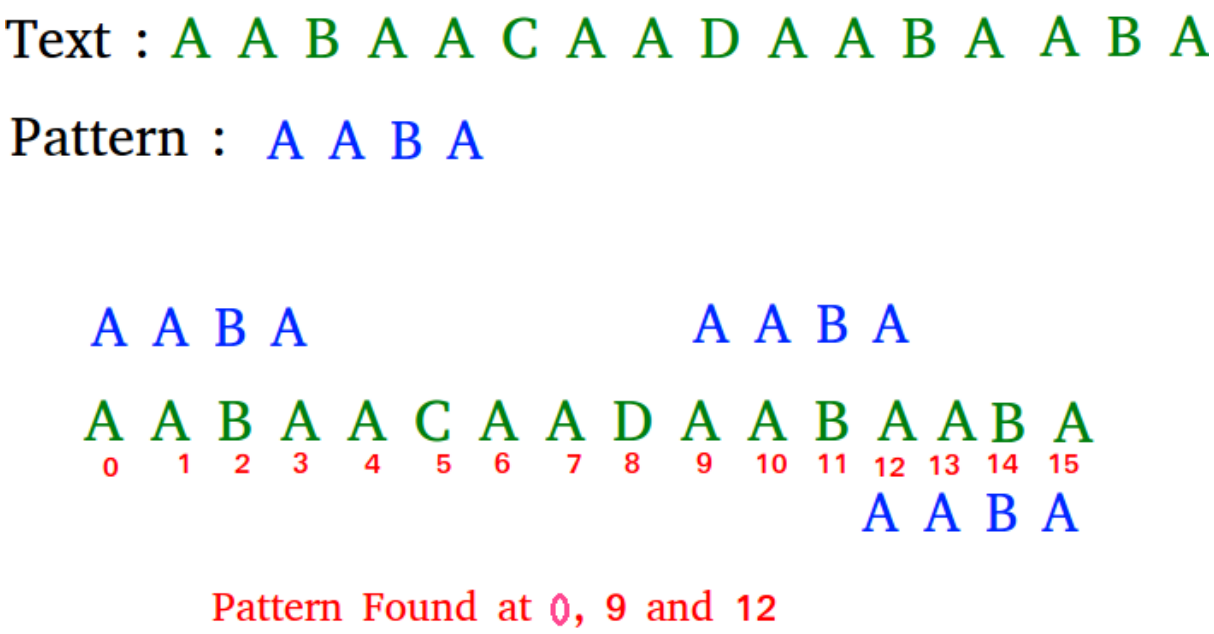
KMP is inside that string is there any perfect same as suffix. So this is the observation what KMP algorithm will do on a pattern to avoid the number of comparisons. The beginning part of the pattern is appearing again everywhere else in the pattern or not that's what we have to observe.

Rabin – karp algorithm

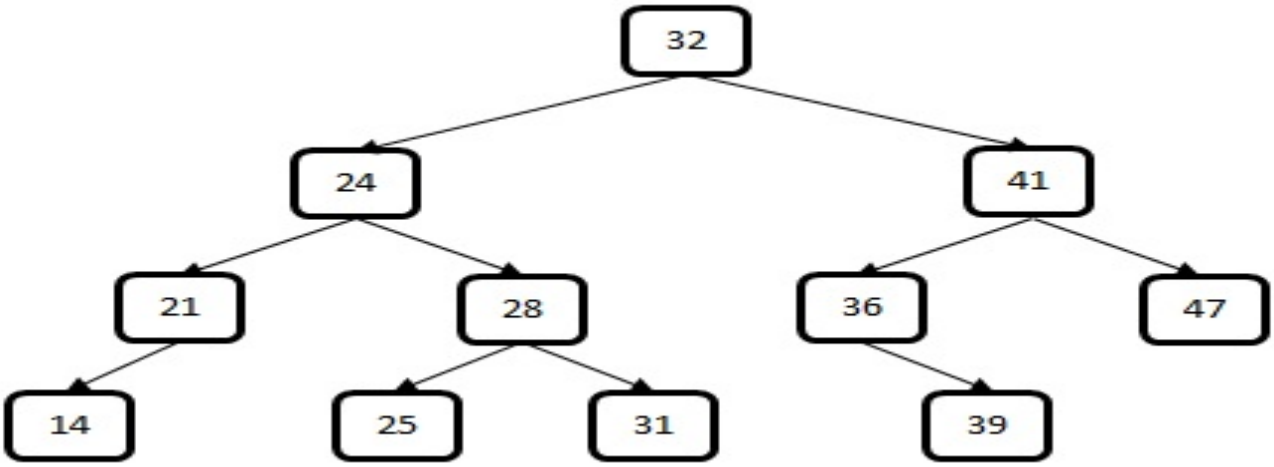
It is a pattern matching algorithm or string matching algorithm. If the text is given and a pattern is given then we have to find out whether this pattern is present in the string or not and in text or not. There are various string or pattern matching but this is one of the pattern matching algorithm.



Now example of Rabin – Karp algorithm:



AVL Tree - Insertion and Rotations.



this is binary search tree the keys are arranged such that all the element on the left hand side are smaller than the node and the element in right hand side is greater than node means the element left in subtree is smaller and right than that element will greater. so, it will be easy to search key element. time taken binary search tree depends upon on

height. height and time taken for searching of binary search tree maximum will be n and minimum logn .

Height is not under control it all depends on how keys are inserted. So, it is drawback of binary search tree.

There are different types of rotation which we can convert binary search into a balance binary search tree so that is the idea of AVL search tree. Rotations are only done in three nodes.

Balance Factor = Height of left subtree – Height of right subtree.

$$BF = HL - HR = \{ -1,0,1\}$$

$$|BF| = |HL - HR| \leq 1.$$

We calculate balance factor for each node. If tree is not balanced means out the range of { -1,0,1} then some rotation are performed to balancing that node.

B Trees and B+ Trees

b trees and b+ trees are originated from M-Way search tree.

M-Way search tree

M-Way search tree in which each node have at most M children and M – 1 key. It is extension of binary search tree.

B Trees

There must be some rules and guidelines for creating M-Way search tree those guidelines are called as B tree. It is M-Way search tree with some rule.

Rules

- You can create a next mode only if the current node is at least half of filled with the children.
- First node/Root have minimum two children.
- All leaf at same level.