unacademy

LET'S CRACK IT!

Algorithm

GATE & ESE

Disclaimer: The content provided herein is created and owned by a third-party service provider and licensed to the Company. The Company disclaims all rights and liabilities in relation to the content. The author of the content shall be solely responsible towards, without limitation, any liabilities, damages, or suits which may arise with respect to the same. The content contained herein is copyright protected and any unauthorised use or distribution is not allowed.

Contents

Chapter 1 Introduction . . . . 4 - 46

Chapter 2 Divide and Conquer . . . . 47 - 103

Chapter 3 Greedy Techniques . . . . 104 - 177

Chapter 4 Dynamic Programming (Principle Of Optimality) . . 178 - 198

Chapter 5 Graph Traversal Algorithm . . . . 199 - 209

Chapter 6 Sorting Algorithm . . . . 210 - 240

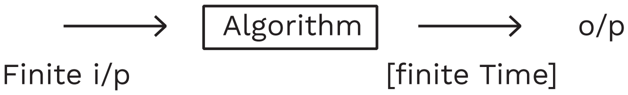
Introduction

Algorithm:

Step by step representation of computer program

Characteristics of Algorithm:-

1. Finiteness :- Algorithm must terminate in finite amount of Time



2. Definiteness :- Each step of Algorithm must have unique solution (also called Deterministic Algorithm)

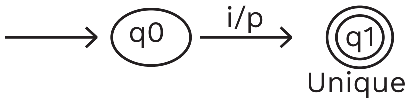
3. Effectiveness:- Each step of the Algorithm must be basic

Deterministic Algorithm

Each step in Algorithm must have only one unique solution.

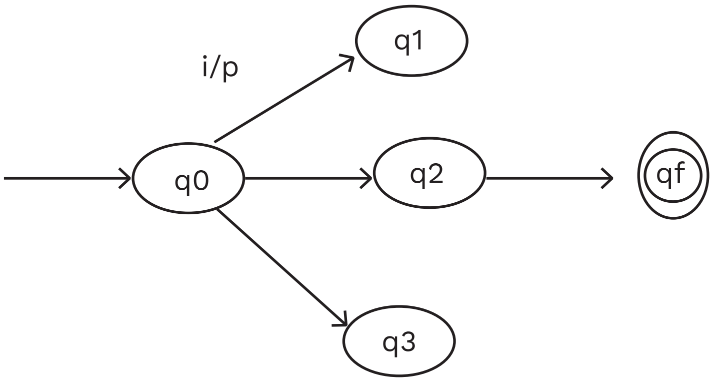
It is possible to implement them in computers

DFA :



Non Deterministic Algorithm not possible to implement in computer. Each step in Algorithm can have finite number of solutions && Algorithm should chose correct solution is first attempt.

NFA:



Deterministic Algorithm:

a[1...n] is array and x is element to be searched

```
for (i = 1 ; i ≤ n ; i++)     number of comp = n

{
    if (x = = a [i] )
     return (i)
}
return (-1)
```

Non Deterministic Algorithm

```
i = choose (1, n)
if (x = = a [i]
     return(i);
  else
     return (-1)
```

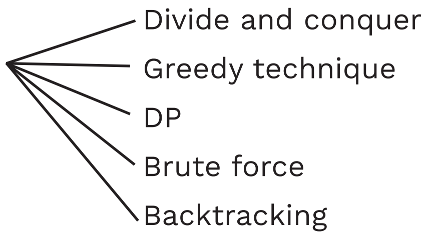
number of comparison = 1

faster compared to non deterministic version

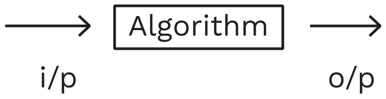
Step to Design Algorithm :

1. Design Algorithm

   Design Algorithm using best Design technique



2. Validation of Algorithm: Test logic of Algorithm correct or not

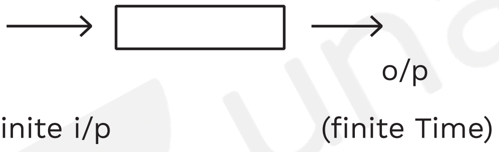


3. Analysis of Algorithm: Estimation of CPU execution Time and main memory space to complete execution of Algorithm.

4. Testing of program : Test program for all possible i/p's by using system testing Methods. [Testing tools]

Decidable Problem

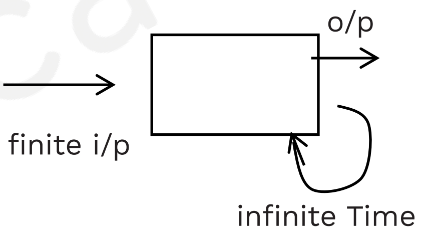
Problem for which there exists Efficient Algorithm



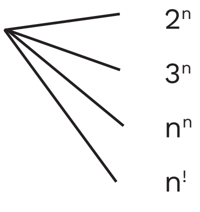
$$\left.\begin{array}{l} n^2 \\ n\log n \\ n^3 \\ nk \end{array}\right\}$$ polynomial Time Algorithm

Un-decidable Problem

Problem for which no efficient Algorithm exists



Exponential Time



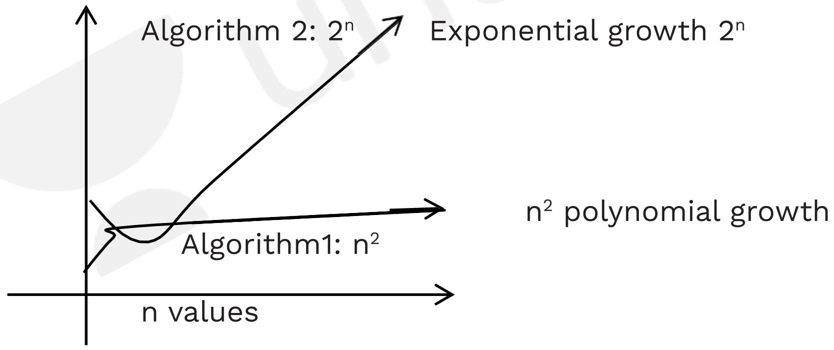
i/p size	Polynomial Algorithm	Exponential Algorithm
n	n^2	2^n
10	100	1024
11	121	2048
12	144	4096
;		,
;		;
49	492	249
50	502	250

Que: For an input of size n, algorithm takes 2 computations. Find time required to terminate algorithm given n=200 inputs?

Assume, fastest computer = 2^{20} instructions/sec

Solution: 2^{20} * 60 * 60x24x365 instructions /year

$\Rightarrow 2^{46}$ instructions/year. $\frac{2^{200}}{2^{46}} = 2^{154}$ year



Algorithm Analysis :-

Estimation of CPU execution Time && main memory space to complete execution of Algorithm

Time complexity:

Estimation of CPU execution Time

= Number of computations required to terminate Algorithm Irrespective of Hardware/ software Time

= Sum of frequency count of each instruction

Ex :- sum of all elements of array

```
    Algorithm sum (a,n)
 {
    sum = 0 → 1
    for (i = 1 ; i < = n ; i++) → 2(n+1) +1
    {
        sum = sum + a[i] ; → n
    }
          return (sum); → 1
    }                       Time Complexity: 3n+5   ≈ θ(n)
```

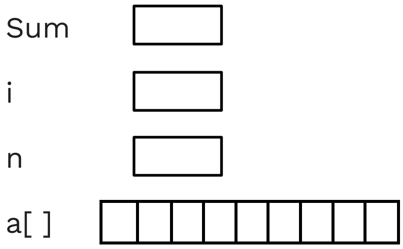
Space Complexity :

Number of main memory Units required to complete execution of Algorithm

Algorithm sum (a, n)

```
    => sum  ⎫
       i    ⎪
       n    ⎬   n + 3 = θ(n)
       a[ ] ⎭   units of Main memory (Space Complexity)
```

Memory Units



Space complexity of Algorithm also includes Data Structure space required

[Stack | Queue | Tree]

Time Complexity | space complexity of loops:-

1. loop
2. Nested loop

1) for (i = 1 ; i ≤ n ; i++) → 2 (n + 1) +1

    Printf (“ Unacademy”); → n

    = 3n+3

    = θ(n)

    Time Complexity : θ (n)

    Space Complexity : θ (1)

2) for (i = n ; i ≥1 ; i--)

    printf (“ Unacademy “) ; → n time

    Time Complexity : θ(n)

    Space Complexity : θ (1)

3) for (i = 1 ; i ≤ n ; i = i+5)

    printf (“Unacademy”) ; → $\left\lceil \frac{n-1}{5} \right\rceil$

i = 1 , 1 + 5 , 1+2*5 , 1+3*5 ,....., 1+K*5

1 + K*5 = n

$K = \frac{n-1}{5} = \theta(n)$

Time Complexity : θ(n)

Space Complexity : θ(1)

4) for (i = n ; i ≥ 1 ; i = i -5)

printf (“ Unacademy”) ; → $\left\lceil \frac{n-1}{5} \right\rceil$

Time Complexity : θ(n)

Space Complexity : $\theta(1)$

5) for (i = 1; i ≤ n ; i = i *2)

printf (“ Unacademy”); → $[\log_2 n] + 1$ Time

$i = 1 , 1*2 , 1*2^2, 1*2^3 , 2^4 \ldots\ldots\ldots 2^k$

$2^k \leq n$

$K \log_2 2 \leq \log_2 n$

$K = [\log_2 n]$

↓

(Number of Time printf executed so, it is always integers)

Time Complexity= θ ($\log_2 n$)

Space Complexity = θ (1)

6) for (i = n ; i ≥ 1 ; i = i/2)

printf (“ Unacademy”) ; → $[\log_2 n] + 1$

$$\underbrace{i = n , \frac{n}{2} , \frac{n}{2^2} \ldots\ldots\ldots \frac{n}{2^k}}$$

Printf → k + 1 Time

$\frac{n}{2^k} \geq 1$

$2^k \leq n$

$K = [\log_2 n]$

Time Complexity= θ ($\log_2 n$)

Space Complexity = θ (1)

7) for (i=2 ; i ≤ n ; i =i²)

Printf (“ Unacademy’) ; → $[\log_2(\log_2 n)+1]$ Time

$i = 2 , 2^2 , (2^2)^2 , \ \ldots\ldots\ldots\ldots\ 2^k$

$2^k \leq n$

K = [log n]

$$\underbrace{i = 2\ ,\ 2^2\ ,\ (2^2)^2\ ,\ \left((2)^2\right)^3,\ \left((2)^2\right)^4\ \ldots\ldots\ldots\ldots\left((2)^2\right)^k}$$

Printf → K +1 Time

runs

$\left((2)^2\right)^k <= n$

$2^k \log_2 \leq \log_2 n$

$2^k \leq \log_2 n$

$K \log_2 \leq \log_2 (\log_2 n)$

K = [log2 (log2n)]

Time Complexity = θ $(\log_2 \log_2 n)$

8) for (i = n ; i ≥ 2 ; i = √i)

Printf (“ Unacademy”) ; → $[\log_2 \log_2 n]+1$

$$\underbrace{i = n\ ,\ (n)^{1/2}\ ,\ (n)\ \frac{1}{2^2}\ ,\ (n)\ \frac{1}{2^3}\ \ldots.\ (n)\ \frac{1}{2^k}}$$

Printf → K + 1 Time

$(n)\ \ \frac{1}{2^k} \geq 2$

$\frac{1}{2^k} \log_2 n \geq \log_2 2$

$\log_2 n \geq 2^K$

$2^K \leq \log_2 n$

$K = [\ \log_2 (\log_2 n)\ ]$

Time Complexity = θ $(\log_2 \log_2 n)$

i. $\log_a a = 1$

ii. $\log_c a^b = b \log_c a$

iii. $\log_c(a*b) = \log_c a + \log_c b$

iv. $\log_c(a/b) = \log_c a - \log_c b$

v. $\log_b a = \dfrac{1}{\log_a b}$

vi. $\log_b a = \dfrac{\log_c a}{\log_c b}$

vii. $a^{\log_c b} = b^{\log_c a}$

EX: - $64^{\log_2 n} = n^{\log_2 64} = n^6$

viii. $1 + 1/2 + 1/3 + 1/4 \ldots\ldots + 1/m \approx \log_e m$

$$\int_{x=1}^{n} (1/x)dx = [\log x]_1^n + c$$

$$= \log n - \log 1 + c$$
$$= \log n$$

Time Complexity

1. for (i=1 ; i ≤ n ; i = i+c)
   or
   for (i=1 ; i ≤ n ; i = i–c)

   $\dfrac{n}{c} = \theta\ (n)$

2. for (i=1 ; i ≤ n i = i * c)
   or
   for (i=1 ; i ≤ n i = i/c)

   $\log_c n \rightarrow \theta((\log n)$

3. for (i = c ; i ≤ n ; i = i^c)
   or
   for (i = n ; i ≥ c ; i =$i^{1/c}$)

   $\log_c \log_c n$
   θ (log logn)

for all the above space comp = θ (1)

Loop Time Complexity depends on

→ Initial value of loop variable

→ Terminated value of loop variable

→ Increment/ decrement value of loop variable

EX:- for (i = 1 ; i ≤ 2^n ; i = i * 2)

Printf (“Unacademy”) ; → n + 1 Time

$$\underbrace{i = 1 , 2 , 2^2 \le 2^n}$$

Printf (k + 1) Time

$2^K \le 2^n$

K = n

Time Complexity = θ(n)

Q. for (i = n^2 ; i ≥ 1 ; i = i/2)

printf (“unacademy”) ; → [$\log_2 n^2$] + 1 Time

$$\underbrace{i = n^2, \frac{n^2}{2}, \frac{n^2}{2^2}, \frac{n^2}{2^3} \text{ ------ } \frac{n^2}{2^k}}$$

Printf → K + 1 Time

$\frac{n^2}{2^k} \ge 1.$

Time Complexity = θ(log n)

$n^2 \ge 2^K$

$2^K \le n^2$

$K \le \log_2 n^2$

$K = [\log_2 n^2]$

EX:- for (i = n^2 ; i ≥ 2 ; i = √i)

printf (“unacademy”) ; → [$\log_2 \log_2 n^2$] + 1 time

$$\underbrace{i = n^2, (n^2)^{1/2}, (n^2)^{1/2^2}, (n^2)^{1/2^3} ...... (n^2)^{1/2^k}}$$

K + 1 Time

$(n^2)^{(1/2)^k} >= 2$

$\frac{1}{2^k} \log_2 n^2 \geq 1.$

$\log_2 n^2 \geq 2^K$

$2^K \leq \log_2 n^2$

$K = [\ \log_2 \log_2 n^2]$

Time Complexity = Θ (log log n)

EX:- for (i = n/2 ; i ≤ n ; i = i * 2)

Printf ("UNACADEMY") ; → 1 + 1 → 2 Time

$$i = \underbrace{\frac{n}{2}, \frac{n}{2} \times 2\ , \frac{n}{2} * 2^2\ , \left(\frac{n}{2}\right)2^3 \ldots\ldots \left(\frac{n}{2}\right)2^K}_{K\ +\ 1\ \text{Time}}$$

$\left(\frac{n}{2}\right)2^K \leq n$

$n2^{K-1} \leq n$

$\log_2(n*2^{k-1}) <= \log_2 n$

$\log_2 n + \log_2 2^{k-1} <= \log_2 n$

$\log_2 2^{K-1} = 0$

Time Complexity = $\Theta(1)$

$K - 1 = 0$

$K = 1.$

EX:- for (i = n; i ≥ n/2 ; i = i/2)

Printf ("UNACADEMY") ; → 2 Time

$$i = \underbrace{\frac{n}{2}, \frac{n}{2^2}, \frac{n}{2^3} \ldots\ldots\ldots \frac{n}{2^k}}$$

(K + 1) Time

$\frac{n}{2^K} \geq n/2$

$\frac{n}{2^K} = \frac{n}{2}$

$1 = 2^{K-1}$

$K-1 = 0$

$K = 1$

EX:- for (i = 3 ; i ≤ 3^n ; i = i^3)

Printf (“UNACADEMY”) ; → $[\log_3 n] + 1$

$$i = 3,\ \underbrace{(3)^3,\ \left((3)^3\right)^2,\ \left((3)^3\right)^3 \text{ x } \left((3)^3\right)^4\ \ldots\ldots\ \left((3)^3\right)^k}_{\text{K + 1 Time}}$$

$\left((3)^3\right)^k \le 3n$ Time Complexity = θ (logn)

$3^K \log_3 3 \le \log_3 3^n$

$3^K \le n$

$K \le \log_3 n \quad K = [\log_3 n]$

Nested Loops

1) Independent Nested Loop:

Inner loop does not depend on the outer loop variable.

```
for (i = 1 ;  i ≤ n ;  i++)
 {
    for  ( j = n ; j ≥ 1 ;  j = j/2)
    {
 Printf (“Unacademy”);
          }
    }
```

Time complexity is multiplication of frequency count of each loop

n*logn

i = 1	i = 2	i = n
j = logn Time	j = logn times	j = log n Time
Printf = log n Time	Printf = logn	Printf = logn

Time Complexity → θ (nlogn)

2) Depending Nested Loop:

Inner loop variable depends of value of outer loop variable

```
for ( i = 1 ;  i ≤ n  ;  i++)
for ( j = 1 ,  j ≤ n ;  j = j + i)
     Printf ((“UNACADEMY”);
```

Much expand loops to find Time Complexity.

i = 1	2	3	…………	n
j = n Time	n/2 Time	n/3 Time		n/n Time

$$\text{Printf} \Rightarrow \left\{ n + n/2 + n/3 + \ldots\ldots + \frac{n}{n} \right\}$$

$$n \left\{ 1 + 1/2 + 1/3 + \ldots\ldots\ldots\ 1/n \right\} = \theta\ (n\log n)$$

```
for ( j = 1 ;  j ≤ n ;  j = j+2)
```

$$\rightarrow \frac{n}{2}\ \text{Time}$$

Q.

```
for ( i = 2 ;  i ≤ n ;  i =i^2)
  {
       for ( j = 1 ;  j ≤ n ;  j++)
            for ( k = 1 ;  k ≤ n ;  k = k + j)
                 Printf (“UNACADEMY”) ;
       }
```

log logn * expand

$\longleftrightarrow$

loop

Time Complexity= log log n * [nlog n]

θ (n log n. log logn)

Q.

```
for ( i = n  ;  i ≥ 2 ;  i = i^(1/3))
    for ( j = 1 ;  j ≤ 2^n ; j = j*2)
    for ( k = 1 ;  k ≤ n ;  k = k+10)
    Printf ( “UNACADEMY”)
```

Time Complexity?

$(n)^{1/3^K} \geq 2$

$\frac{1}{3^k} \log_2 n \geq 1$

$\log_2 n >= 3^K$

$3^K <= \log_2 n$

$K \log_3 3 \leq \log_3 \log_2 n$

$K = [\log_3 \log_2 n]$

Time Complexity = $\log \log n * n * \frac{n}{10}$

$\frac{n^2}{10} \text{loglog } n$

$\theta (n^2 \log \log n)$

2.
```
for ( i = 2;  i ≤ $n^2$ ;  i = i+2)
    for  ( j = n/2 ; j ≤ n ; j = j * 2)

        for  ( k = $n^2$ ;  k ≥ 2 ; k = $\sqrt{k}$ )
    Printf   (“UNACADEMY”) ;
```

time complexity= $O(n^2 \log_2 \log_2 n)$

Time complexity and space complexity of recursive algorithms

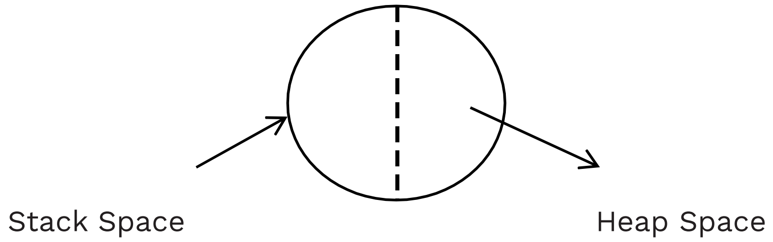
1. Space complexity of recursive algorithm:-

Stack Data Structure used to execute recursive algorithms on all function calls.

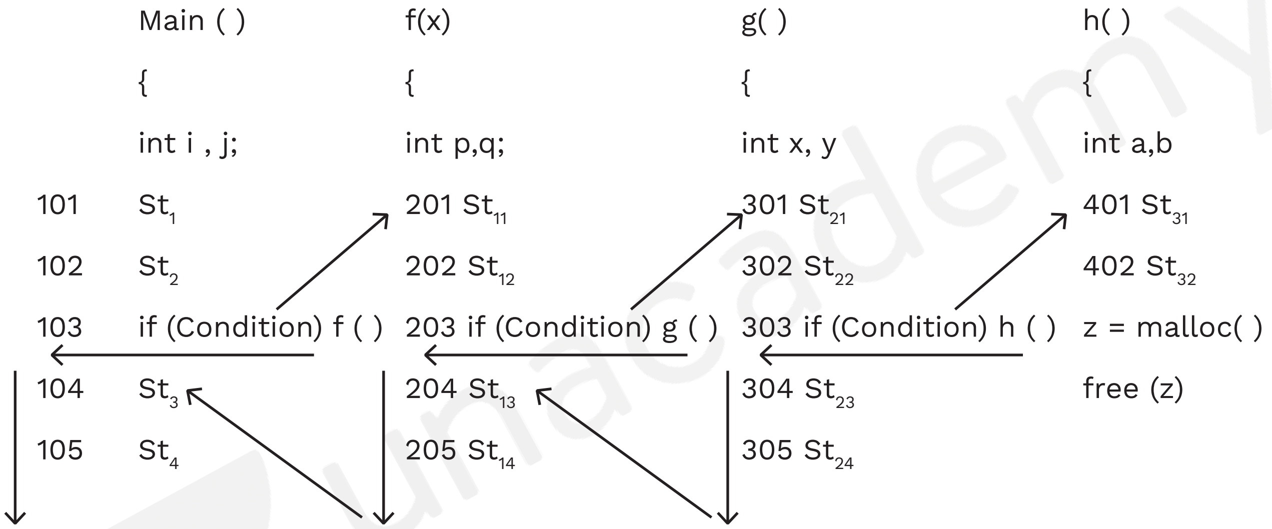
Program under execution: (Process)

→ Stack space

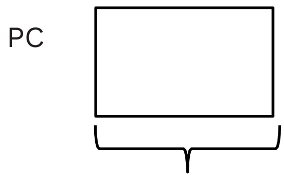
→ Heap space



Stack used for recursive calls for/ function calls to store return address from subroutine

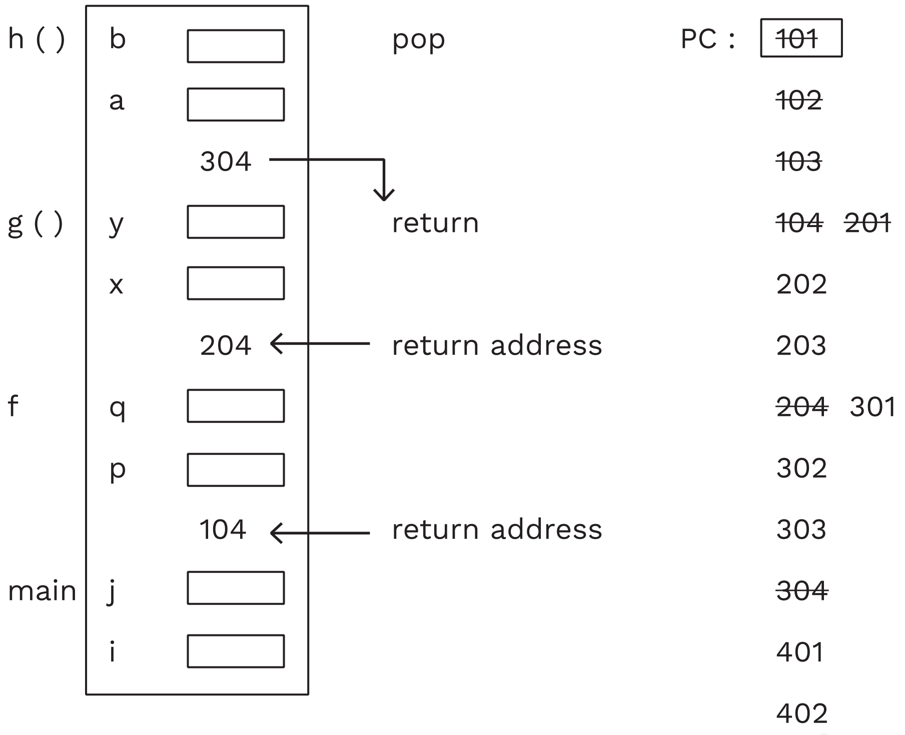


	Main ( )	f(x)	g( )	h( )
	{	{	{	{
	int i , j;	int p,q;	int x, y	int a,b
101	St_1	201 St_{11}	301 St_{21}	401 St_{31}
102	St_2	202 St_{12}	302 St_{22}	402 St_{32}
103	if (Condition) f ( )	203 if (Condition) g ( )	303 if (Condition) h ( )	z = malloc( )
104	St_3	204 St_{13}	304 St_{23}	free (z)
105	St_4	205 St_{14}	305 St_{24}	

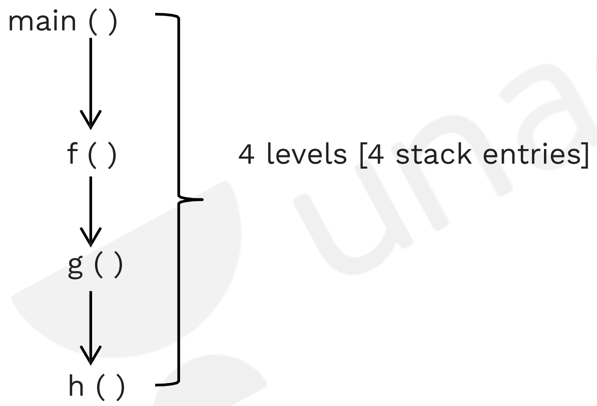


stores address of next executable instruction

- If we do not use free variables it will remain in stack space until termination of the main program. After termination of main program whole memory space is de-allocated:



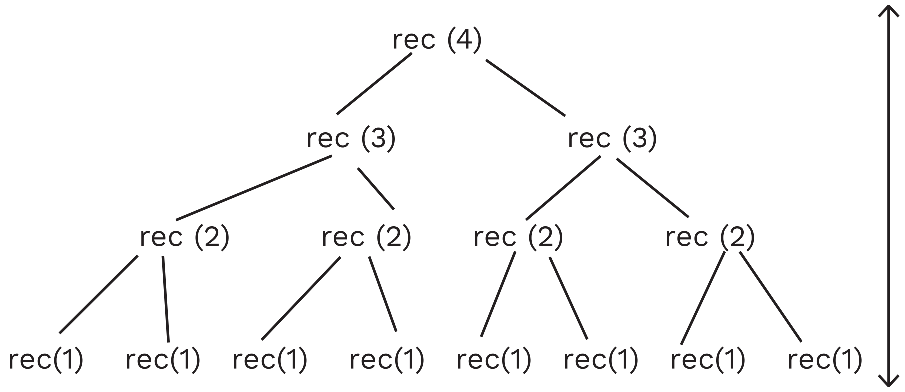
Stack [LIFO]



Space complexity of rec (n) algorithm

```
rec(n)
{
if(n<=1) return n;
else
return(rec(n-1)+rec(n-1));
}
```

if n =4.



n levels is for n inputs

so, depth of recursion is n(4).

Space Complexity of Algorithm : θ (n)

Q. What is the space complexity of a given algorithm?

```
Rec (n)
{
  if (n ≤ 1 )
   return ;
else
     return (rec (n/2) + n ) ;
}
```

Rec (n)

Rec (n/2)

Rec $(n/2^2)$

Rec $\left(\frac{n}{2^k}\right)$

$\frac{n}{2^k} = 1 \rightarrow K = \log_2 n$

depth of recursion K + 1 levels =

$[\log_2 n] + 1$

Space Complexity of rec (n) = θ (log n)

[Stack Space]

Asymptotic Notations:

Used to represent a given function with simplified approximation of other functions.

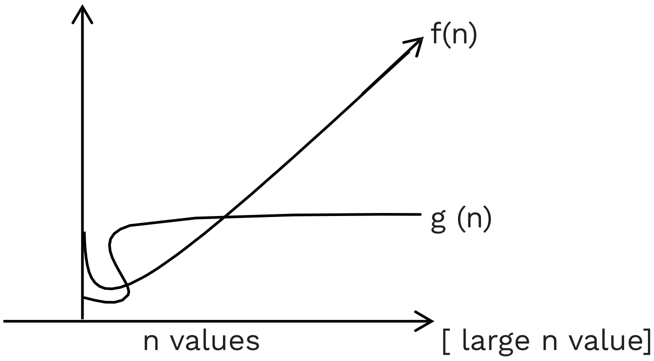
$$\underbrace{2n^2 + 10n + 100}_{\underset{\text{fun}}{\uparrow}} \quad \approx \quad \underset{\underset{\text{approximated}}{\uparrow}}{\theta\ (n^2)}$$

Asymptotic comparison of fun f(n) && g (n) :-

Growth rate comparison of f(n) & g(n) for large “n” values (n → ∞)

(Growth Rate)

- f(n) Asymptotically bigger than g(n)



Growth rate of f(n) is too high compare to growth rate of g(n) far large n values

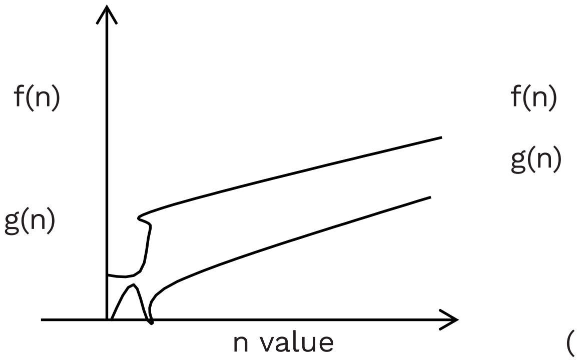
f(n) Asymptotically bigger than g(n) if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

eg. $f(n) = 10n^2$

$g(n) = 100\ n$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

- f(n) Asymptotically equal to g(n) growth rate of f(n) && g(n) are almost parallel.



(ratio of growth rate equal)

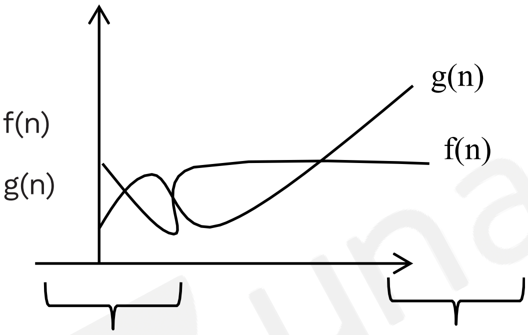
f(n) & g(n) Asymptotically equal if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \text{constant}$$

eg. $\left.\begin{aligned} f(n) &= 10\, n^2 \\ g(n) &= 10\, n^2 \end{aligned}\right\}$ $\lim_{n \to \infty} f(n)/\,g(n) = 10$ [constant]

f(n) & g(n) Asymptotically equal

- f(n) Asymptotically smaller than g(n)



small n value not considered (for Asymptotic comparison) | growth rate for large n value only used for Asymptotic comparison

f(n) Asymptotically smaller than g(n) iff

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

$$\left.\begin{aligned} f(n) &= 10n^2 \\ g(n) &= n^3 \end{aligned}\right\} \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = \frac{10n^2}{n^3} = \frac{10}{n} = 0$$

Big ‘O’ Notation:-

Definition: f(n) and g(n) are non-negative functions

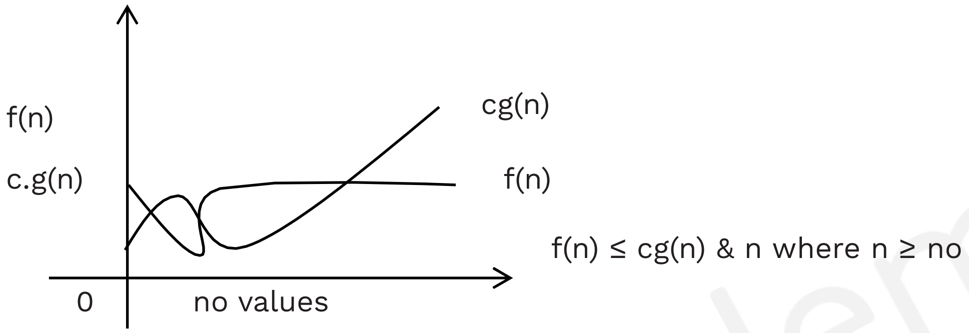
f(n) = 0 (g(n))

if

f(n) ≤ c g(n) for all n values where n ≥ n_0

[C & n_0 are constants].

Graph:



f(n) = 0 (g(n)) iff g(n) Asymptotically bigger or equal to f(n)

1) f(n) = 10n + 5 g(n) = n

f(n) = 0 (g(n)) f(n) ≤ c.n

=> True 10n+5 ≤ 15.n.

2) f(n) = 10n + 5 g(n) = n^2

f(n) = 0 (g (n)) => True

3) f(n) = 10n + 5 g(n) = $\log_{10}n$

f(n) = 0 (g(n)) => false

10n + 5 ≤ 10000 .$\log_{10}n$ not true for all n values

Decreasing functions :-

f(n) = $\frac{1}{n}$, $\frac{c}{n^2}$, $\frac{n}{2^n}$, $\frac{\log n}{n}$

$$\left\{ \frac{n}{2} < \frac{c}{n^2} < \frac{1}{n} < \frac{\log n}{n} \right\}$$

- **Constant Functions**

f2(n)=2 :constant

- **Log Functions**

$f3\ (n) = \log n, (\log n)^{10}, \log \log n, (\log \log n)^{10}$

$\{\log \log n < (\log \log n)^{10} < \log n < (\log n)^{10}\}$

- **Polynomial Functions :**

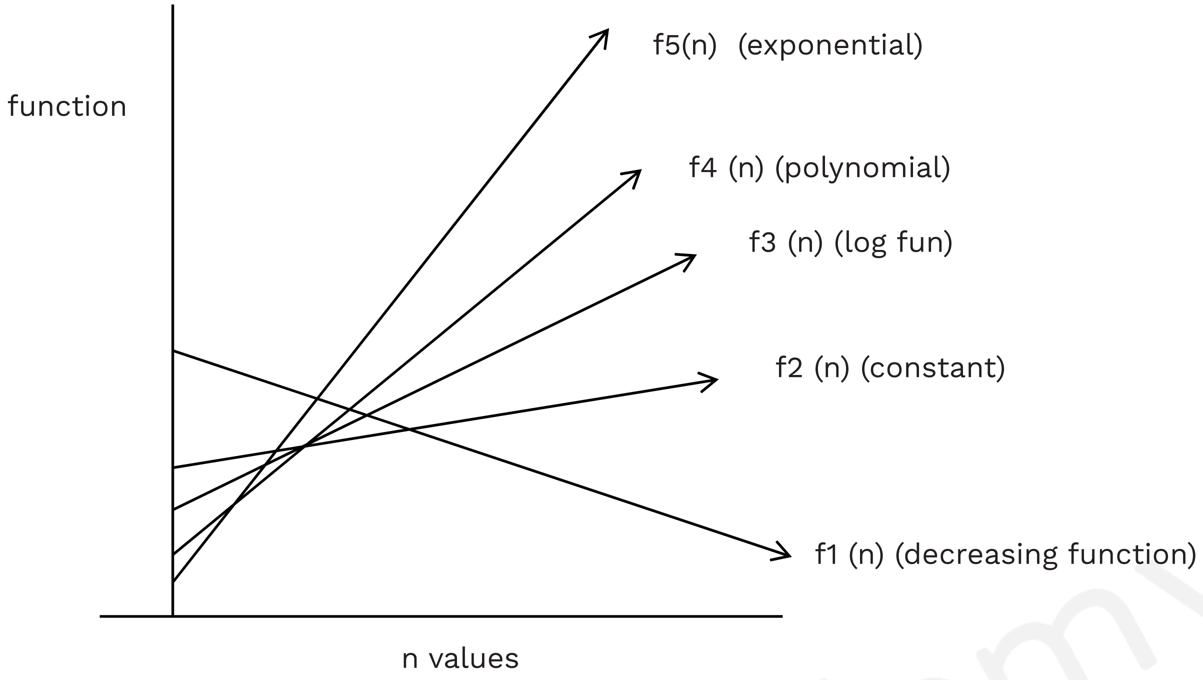
$f4\ (n) = \{ n^{0.1}, n^{0.5}, n^2, n^3, n^{10}, n \}$

$\{ n^{0.1} < n^{0.5} < n < n^2 < n^3 < n^{10} \}$

- **Exponential Function :**

$$f5\ (n) = \left\{ \begin{array}{l} a^n, a < 1 \text{ (decreasing function)} \\ a = 1 \\ \text{(constant fun)} \\ \quad a > 1 \\ \text{(exponential)} \end{array} \right\}$$

`



decreasing < constant < logarithmic < Polynomial < exponential

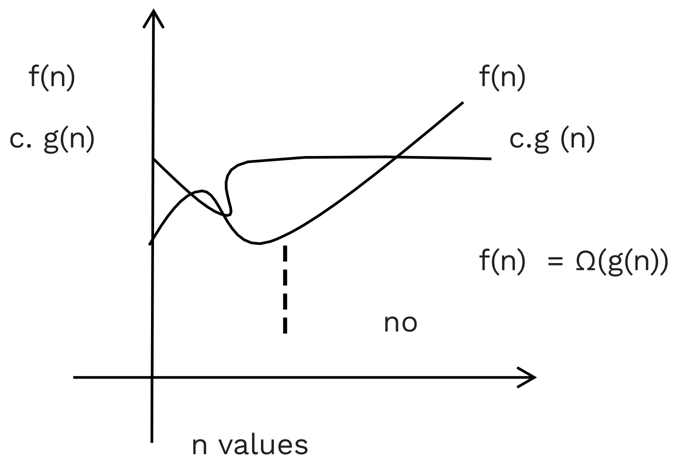
	$f(n) = n^{0.0001}$	$g(n) = \log_{10} n$
$n = 10^{10}$	$10^{1/1000}$	$\log_{10} 10^{10} = 10$
$n = 10^{10000}$	10,	$\log_{10} 10^{10000} = 10000$
$n = 10^{100000}$	10^{10}	10^{5}

- $\log_{10} n < n^{0.0001}$

Omega Notation (Ω) :-

f(n) & g(n) non negative fun f(n) = Ω g(n) iff

f(n) ≥ c g (n) for all n values where n ≥ no [c & n_0 are constant]



- f(n) = Ω g(n) iff g(n) Asymptotically smaller or equal to f(n)

$n! = \Omega(n^2)$

$n^n = \Omega(n!)$

$2^n = \Omega(n^2)$

$n^2+10 = \Omega(n^2)$

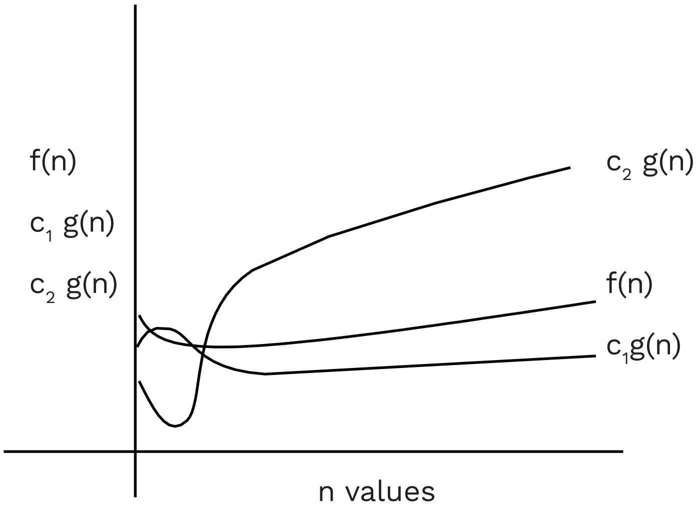
- f(n) = Ω g(n) iff g(n) = O(f(n))

Theta Notation (θ) :-

f(n) & g(n) non negative fun f(n) = θ (g(n)) iff

$$\overset{\Omega}{c_1 \cdot g(n) \leq f(n)} \overset{O}{\leq c_2 \cdot g(n)}$$

(for all n values where $n \geq n_0$, c_1, c_2 are constant)



$f(n) = \theta(g(n))$ iff $g(n)$ Asymptotically equal to $f(n)$

- $f(n) = \theta(g(n))$ iff $f(n) = 0(g(n))$ && $g(n) = \Omega(f(n))$

Little Omega (w):-

- $f(n)$ & $g(n)$ non negative functions

$f(n) = w(g(n))$ iff $\lim\limits_{x \to \infty} \frac{-f(n)}{g(n)} = \infty$

- $f(n) = w(g(n))$ iff $g(n)$ Asymptotically smaller than $f(n)$
- $f(n) = w(g(n))$ iff $g(n) = 0(f(n))$

Properties :

Reflexivity : [but not little o and little w]

$f(n) = 0(f(n))$

$f(n) = \Omega(f(n))$

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

Transitivity:

→ if $f(n) = O(g(n))$ & $g(n) = O(h(n))$

then $f(n) = O(h(n))$

Similarly Ω, θ, o, w, also true

Asymmetric :

1. $f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$
2. $f(n) = 0(g(n))$ iff $g(n) = w(f(n))$

- $f(n) + g(n) = 0(\max(g(n), g(n)))$ True

$n^2 + 2^n = 0(2^n)$

$n^2 + n^2 = 0(n^2)$

- $f(n) + g(n) = \Omega$ (max $(f(n), g(n)$) True

  $n^2 + 2^n = \Omega(2^n)$

  $n^2 + n^2 = \Omega(n^2)$

- $f(n) + g(n) = \theta$ (max $(f(n), g(n)$) True
- $f(n) + g(n) = \theta$ (max $(f(n), g(n))$ false for little 0 and little w.
- $\frac{f(n)}{c} = f(n) * c = 0(f(n))$ True

  $$\left.\begin{array}{l}\frac{f(c)}{c} = f(n) * c = \Omega(f(n)) \text{ True}\end{array}\right\} \text{also true for } \theta$$

Binomial Expansion:

$f(n) = (n + a)^x$ where a && x constant

- $(n + a)^x = \theta(n^x)$

$$\underbrace{[\, a_x n^x + a_{x-1} n^{x-1} + \ldots\ldots\ldots\ldots + a_1 n^1 + a_0]}_{f(n)} \geq \underbrace{[a_x]}_{c1}.\ \underbrace{n^x}_{g(n)}$$

$$\underbrace{[\, a_x n^x + a_{x-1} n^{x-1} + \ldots.. + a_1 n + a_0 \,]}_{f(n)} \leq \underset{c_2}{[a_x + a_{x-1} + \ldots a_0]}\ \underbrace{n^x}_{g(n)}$$

$(n + a)^x = \theta(n^x)$ a && x constant

Test whether given statement is true / false

a. $f(n) = 0((f(n))^2)$ false

eg. $f(n) = 1/n$

$1/n = 0(1/n^2)$ false

b. if $f(n) = 0(g(n))$ $f(n) \leq g(n)$

then $2^{f(n)} = 0[2^{g(n)}]$ false

$= 2^{f(n)} < 2^{g(n)}$ $2^{2n} \neq 0(2^n)$ but $2n=O(n)$

Solving the Recurrence Relation

i. Substitution Method

ii. Recursive tree Method

iii. Master Theorem

1) Substitution Method

1) $$T(n) = \begin{cases} 5\,T(n-1) + n, & \text{for } n>1 \\ 1 & \text{for } n<=1 \end{cases}$$

$T(n) = 5\,T(n-1) + n$

$T(n-1) = 5T(n-2) + n - 1$

$T(n) = 5\,[5T(n-2) + n-1\,] + n$

$T(n) = 5^2\,T(n-2) + 5\,(n-1) + n$

$T(n-2) = 5T(n-3) + n - 2$.

$T(n) = 5^2\,[5T(n-3) + n-2\,] + 5\,(n-1) + n$

$T(n) = 5^3\,T(n-3) + 5^2\,(n-2) + 5\,(n-1) + n$

K Time

$= 5^k\,T(n-k) + 5^{k-1}\,(n-(k-1)]\quad ......+5^2\,(n-2) + 5\,(n-1) + n$

$n - k = 1$

$k = n-1$

$T(n) = 5^{n-1} + 2*\,5^{n-2} + 3*\,5^{n-3} + ...(n-1)*5 + n$

$5*T(n)=5 + 2*5^{n-1} + 3*5^{n-2} + ...+5^2(n-1) + 5n$

$4*T(n)=5^n - 5^{n-1} - 5^{n-2}\,............-5(n-1) - n$

$$T(n) = 5*\frac{(5^n - 1)}{16} - n/4$$

$$T(n) = \theta(5^n)$$

2) $T(n) = \begin{cases} T(n-1) + 1/n, & \text{for } n>1 \\ 1 & \text{for } n<=1 \end{cases}$

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

$$\left(T(n-1) = T(n-2) + \frac{1}{n-1}\right)$$

$$= T(n-2) + \frac{1}{n-1} + \frac{1}{n}$$

$$\left(T(n-2) = T(n-3) + \frac{1}{n-1}\right)$$

$$= T(n-3) + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}$$

⋮ k Time

$$= T(n-k) + \frac{1}{n-(K-1)} + \frac{1}{n-(k-2)} + \ldots\ldots + \frac{1}{n-2} + \frac{1}{n-1} + 1/n$$

= where n- k = 1

K = n-1

$$= T(1) + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots\ldots + \frac{1}{n-1} + \frac{1}{n}$$

$$= 1 + 1/2 + 1/3 + 1/3 \ldots\ldots\ldots + \frac{1}{n}$$

$$= \log_e n = \theta\ (\log n)$$

$$\log_2 n = \log_e n = \underset{\displaystyle \frac{\log_2 n}{\log_2 10}}{\underset{\downarrow}{\log_{10} n}} = \theta\ (\log n)$$

$$\log_2 n$$

3. $$T(n) = \begin{cases} 2T(n/2) + n \log_2 n, & \text{for } n > 1 \\ 1 & \text{for } n <= 1 \end{cases}$$

$$T(n) = 2T\ (n/2) + n \log_2 n$$

$$\left(T(n/2) = 2T \left[\frac{n}{2^2}\right] + n/2 \log_2\ n/2 \right)$$

$$= 2 \left(2T \left[\frac{n}{2^2}\right] + \frac{n}{2} \log_2\ n/2 \right) + n \log_2 n$$

$$= 2^2\ T \left[\frac{n}{2^2}\right] + 2^*\ n/2^*\ \log_2 n/2 + n\log_2 n$$

$$T(n/2^2) = 2T \left[\frac{n}{2^3}\right] + n/2^2\ (n/2^2) \log_2 n/2^2$$

$$= 2^2 \left(2T \left[\frac{n}{2^3}\right] + \frac{n}{2^2} \log_2\ n/2^2 \right) + 2.\ n/2\ \log_2 n/2 + n \log_2 n$$

$$= 2^3\ T \left[\frac{n}{2^3}\right] + 2^2\ n/2^2\ \log_2\ (n/2^2) + 2.\ n/2\ \log_2\ (n/2) + n \log_2 n$$

$\vdots$ K Time

$$= 2^k \; T\left(\frac{n}{2^k}\right) + n \log_2 (n/2^{k-1}) + n \log_2 (n/2^{k-2}) + \dots\dots\dots + n \log_2 n/2^2 + n \log_2 n/2 + n \log_2 n$$

$$= 2^{\log_2 n} \; T(1) + n\left(\log n + \log n/2 + \log n/2^2 + \dots\dots + \log \frac{n}{2^{\log_2 n-1}}.\right)$$

$$= n^{\log_2 2} .1 + n\left(\log n + \log n/2 + \log n/2^2 + \dots\dots\dots + \log \frac{2n}{n}\right)$$

$$= n + n\left(\log_2 2 + \log_2 2^2 + \dots\dots\dots\dots + \log n\right)$$

$$= n + n \; [1 + 2 + 3 \dots\dots\dots\dots \log n]$$

$$= n + n \; \frac{\log n \; (\log n + 1)}{2} = \theta \; (n \log^2 n)$$

4) $T(n) = \begin{cases} 2T(n/2) + n/\log_2 n, & \text{for } n>1 \\ 1 & \text{for } n<=1 \end{cases}$

$$T(n) = 2T(n/2) + \frac{n}{\log_2 n}$$

$$\left(T(n/2) = 2T(n/2^2) + \frac{n/2}{\log_2 (n/2)}\right)$$

$$= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n/2}{\log(n/2)}\right) + \frac{n}{\log n}$$

$$= 2^2 \; T\left(\frac{n}{2^2}\right) + \frac{2}{2} \; \frac{n}{\log(n/2)} + \frac{n}{\log n}$$

$$= T(n/2^2) = 2T\left(\frac{n}{2^3}\right) + \frac{n/2^2}{\log n/2^2}$$

$$= 2^2\left(2T\left[\frac{n}{2^3}\right] + \frac{n/2^2}{\log\left[\frac{n}{2^2}\right]}\right) + \frac{n}{\log\left[\frac{n}{2^2}\right]} + \frac{n}{\log n}$$

$$= 2^3\, T\left[\frac{n}{2^3}\right] + \frac{n}{\log(n/2^2)} + \frac{n}{\log(n/2)} + \frac{n}{\log n}$$

K Time

$$= 2^k\ T\left[\frac{n}{2^k}\right] + \frac{n}{\log(n/2^{k-1})} + \frac{n}{\log\left[\frac{n}{2^{k-2}}\right]} \ \dots\dots + \frac{n}{\log(n/2^2)} + \frac{n}{\log(n/2)} + \frac{n}{\log n}$$

$$= \frac{n}{2^k} = 1$$

$$K = \log_2 n$$

$$= n + n\left(1 + 1/2 + 1/3 + \dots\dots \frac{1}{\log n}\right)$$

$$= n + n\log_e(\log n) = \theta(n\log\log n)$$

Q. $T(n) = \begin{cases} 8T(n/2) + n^2, & \text{for } n>1 \\ 1, & \text{for } n<=1 \end{cases}$

$$T(n) = 8T\ (n/2) + n^2$$

$$[\,T(n/2) = 8T(n/2^2) + (n/2)^2\,]$$

$$= 8\left(8T\left[\frac{n}{2^2}\right] + \left[\frac{n}{2}\right]^2\right) + n^2$$

$$= 8^2\ T\left[\frac{n}{2^2}\right] + 8\left[\frac{n}{2}\right]^2 + n^2$$

$$T\left((n/2^2) = 8T\left[\frac{n}{2^3}\right] + \left[\frac{n}{2^2}\right]^2\right)$$

$$= 8^2 \left(8T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right)^2 \right) + 8\left(\frac{n}{2}\right)^2 + n^2$$

$$= 8^3 \; T\left(\frac{n}{2^3}\right) + 8^2 \left(\frac{n}{2^2}\right)^2 + 8\left(\frac{n}{2}\right)^2 + n^2$$

$$\vdots \quad \text{K Time}$$

$$= 8^k \; T\left(\frac{n}{2^k}\right) + 8^{k-1} \left(\frac{n}{2^{k-1}}\right)^2 + 8^{k-2} \left(\frac{n}{2^{k-2}}\right)^2 + \ldots\ldots\ 8\left(\frac{n}{2}\right)^2 + n^2$$

$$=8^k \; T\left(\frac{n}{2^k}\right) + \left(2^k(n^2/2 + n^2/2^2 + n^2/2^3 \ldots\ldots\ldots n^2/2^k) \right)$$

$$\left(\frac{n}{2^k} = 1 \;\Rightarrow\; n = 2^k \quad k = \log_2 n \right)$$

$$2^k * n^2 * \frac{(1/2)}{1-(1/2)}$$

$$n^2 * 2^{\log_2 n}$$

$$= \theta\,(n^3)$$

5) $T(n) = \begin{cases} 7T(n/2) + n^2, & \text{for } n>1 \\ 1 \quad , & \text{for } n<=1 \end{cases}$

$$T(n) = 7T(n/2) + n^2$$

$$\left(T[(n/2)] = 7T\left(\frac{n}{2^2}\right) + \left(\frac{n}{2}\right)^2 \right)$$

$$= 7\left(7T\ (n/2^2) + \left(\frac{n}{2}\right)^2\right) + n^2$$

$$= 7^2\, T\left(\frac{n}{2^2}\right) + 7\left(\frac{n}{2}\right)^2 + n^2$$

$$\left(T\left(\frac{n}{2^2}\right) = 7T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right)^2 \right)$$

$$= 7^2\left(7T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right)^2\right) + 7\left(\frac{n}{2}\right)^2 + n^2$$

$$= 7^3\ T\ \frac{n}{2^3} + 7^2\left(\frac{n}{2^2}\right)^2 + 7\left(\frac{n}{2^2}\right)^2 + n^2$$

$\vdots$ K Time

$$= 7^k\, T\left(\frac{n}{2^k}\right) + 7^{k-1}\left(\frac{n}{2^{k-1}}\right)^2 + 7^{k-2}\left(\frac{n}{2^{k-2}}\right)^2 + \ldots\ldots + 7^2\left(\frac{n}{2^2}\right) + 7\left(\frac{n}{2}\right)^2 + n^2$$

$$= 7^k\ T\left(\frac{n}{2^k}\right) + n^2\left(\left(\frac{7}{4}\right)^{k-1} + \left(\frac{7}{4}\right)^{k-2} + \ldots\ldots + \left(\frac{7}{4}\right)^2 + \frac{7}{4} + 1\right)$$

$$= \frac{n}{2^k} = 1 \qquad k = \log_2 n$$

$$= 7^{\log_2 n}.1 + n^2 \left(\frac{1\left(\left(\frac{7}{4}\right)^K - 1\right)}{7/4\ -1}\right)$$

$$= 7^{\log_2 n} + n^2 \left[\frac{4}{3} \left(\frac{7}{4}\right)^{\log_2 n} - 1 \right]$$

$$= 7^{\log_2 n} + n^2 \{ (4/3)^* (n^{\log_2 (7/4)} - 1)$$

$$= n^{\log_2 7} + n^* (4/3)^*(n^{0.80} - 1)$$

$$= \theta(n^{\log_2 7})$$

Practice Problems:-

Q.1 Consider following function f

```
    void f(int n) {
    If (n≤ 0) return;
        Else {
    print (n);
    f(n-2);
    print (n);
    f(n-1); }
}
```

Let R(n) be recurrence relation which computes sum of values printed by f(n).

Then R(n) is

a) R(n-1) + R(n-2)

b) R(n-1) + R(n-2)+ n

c) R(n-1) + R(n-2) +2n

d) None of these

Common Data for Q.2 and Q.3

Consider the following C function;

```
    double foo (int n) {
    int i;
    double sum;
    if (n = = 0)
    return 1.0;
    else {
    sum = 0.0;
    for (i = 0; i< n; i ++)
    sum + = foo (i);
    return sum; }
 }
```

Q.2 The space complexity of the above functionis-

a) O(1)

b) O(n)

c) O(n!)

d) $O(n^n)$

[GATE-2005]

Q3 Suppose we modify the above function foo() and store the values of foo (i), 0 < = I < n, as and when they are computed. With this modification, the Time Complexity for function foo() is significantly reduced. The space complexity of the modified function would be:

a) O(1)

b) O(n)

c) $O(n^2)$

d) O(n !)

[GATE-2005]

Common Data Questions Q.4 and Q.5

Consider the following C functions:

```
    int f1(int n) {
    if (n = = 0|| n = = 1) return n;
    else
    return (2* f1(n-1) + 3 * f1(n-2)); }
    int f2(int n)
    {
    int i;
    int X[N], Y[N], Z[N];
    X[1] =1; Y[1] = 2; Z[1] =3;
    for (i = 2, i < = n; i++) {
    X(i) = Y [i-1] + Z [i-2];
    Y(i) = 2 * X(i),
    Z(i) = 3 * X[i]; }
    Return X[n];
 }
```

Q.4 The running Time of f1(n) and f2(n) are

a) $\Theta(n)$ and $\Theta(n)$

b) $\Theta(2^n)$ and $\Theta(n)$

c) $\Theta(n)$ and $\Theta(2^n)$

d) $\Theta(2^n)$ and $\Theta(2^n)$

[GATE-2008]

Q.5 f1(8) and f2(8) return the values

a) 1661 and 1640

b) 59 and 59

c) 1640 and 1640

d) 1640 and 1661

[GATE-2008]

Q.6 Consider the following functions:

$f(n) = 3n^{\sqrt{n}}$

$g(n) = {}^{\log_2 n}$

$h(n) = n!$

Which of the following is true?

a) h(n) is O (f(n))

b) h(n) is O (g(n))

c) g(n) is not O(f(n))

d) f(n) is O (g(n))

[GATE-2000]

Q.7 What does the following Algorithm approximate?

(Assume m > 1, E >0).

```
    x = m;
    y = 1;
while (x – y >E)
    {
    x=(x+y)/2;
    y=m/x;
    }
    print (x);
```

a) log m   b) m^2

c) $m^{1/2}$ d) $m^{1/3}$

[GATE-2004]

Q.8 Consider the following C-program fragment in which i, j and n are integer variables. For (i = n, j = 0; i > 0, i / = 2, j + = i); Let Val (j) denote the value stored in the variable j after termination of the for loop. Which one of the following is true?

a) val (j) = $\Theta(\log n)$

b) val j) = $\Theta(\sqrt{n})$

c) Val (j) = $\Theta(n)$

d) Val (j) = $\Theta(n \log n)$

[GATE-2006]

Q.9 Consider the following C code segment:

```
     int is Prime(n)
 {
      int i, n;
     for (i = 2; i < = sqrt (n); i ++)
  {
     if (n mod i ==0)
   {
     printf ("Not Prime ")
     return 0;
    }
     return 1;
   }
}
```

Let T (n) denote the number of Time the for loop is executed by the program on input n. Which of the following is TRUE?

a) $T(n) = O(\sqrt{n})$ and $T(n) = \Omega(\sqrt{n})$

b) $T(n) = O(\sqrt{n})$ and $T(n) = \Omega(1)$

c) $T(n) = O(n)$ and $T(n) = \Omega(\sqrt{n})$

d) None of the above

[GATE-2007]

Q.10 Consider the following function:

```
     Int unknown (int n)
 {
     Int i, j, k = 0;
     for (i = n/2; i <=n; i++)
     for (j = 2; j <=n; j= j*2)
     k = k + n/2;
     return (k);
 }
```

The return value of the function is

a) $\Theta(n^2)$

b) $\Theta(n^2 \log n)$

c) $\Theta(n^3)$

d) $\Theta(n^3 \log n)$

[GATE-2013]

Q.11 Consider the following C function:

```
   int fun 1 (int n)
{
   int i, j, k, p, q = 0;
   for (i = 1; i < n; ++i)
 {
   p =0;
   for (j =n; j > 1; j = j/2)
   ++p;
   for (k = 1; k < p; k = k*2)
   ++q;
}
   return q;
 }
```

Which one of the following most closely approximates the return value of the function fun 1?

a) n^3

b) $n (\log n)^2$

c) $n \log n$

d) $n \log (\log n)$

[GATE-2015]

Q.12 Let $f(n) = n^2 \log n$ and $g(n) = n(\log n)^{10}$ be two positive functions of n.

Which of the following statements is correct?

a) $f(n)= O(g(n)$ and $g(n) \neq O(f(n))$

b) $g(n)=O(f(n)$ and $f(n) \neq O(g(n))$

c) $f(n) \neq O(g(n))$ and $g(n) \neq O(f(n))$

d) $f(n) = O(g(n))$ and $g(n) = O(f(n))$

[GATE-2001]

Q.13 If $g = O(f)$ then find true statement from the following.

a) $f = O(g)$

b) $g = \Theta(f)$

c) $f + g = \Theta(g)$

d) $f + g = \Theta(f)$

Q.14 Consider the following three claim:

1. $(n + k)^m = \Theta(n^m)$ where k and m are constants
2. $2^{n+1} = O(2^n)$
3. $2^{2n+1} = O(2^n)$

Which of these claim are correct?

a) 1 and 2

b) 1 and 3

c) 2 and 3

d) 1, 2 and 3

[GATE-2003]

Q.15 Arrange the following functions I increasing asymptotic order:

A. $n^{1/3}$

B. e^n

C. $n^{7/4}$

D. $n \log^9 n$

E. 1.0000001^n

a) A,D, C, E, B

b) D, A, C, E, B

c) A, C, D, E, B

d) A, C, D, B, E

[GATE-2008]

Q.16 In the following C function, let n ≥ m.

```
int gcd(n,m)
{ if (n%m = = 0) return m;
n = n%m;
return gcd (m, n);
}
```

How many recursive calls are made by this function?

a) $\Theta(\log_2 n)$

b) $\Omega(n)$

c) $\Theta(\log_2 \log_2 n)$

d) $\Theta(\sqrt{n})$

[GATE-2007]

Q.17 If T_1= O(1), match List-I with List-II select the correct answer using the codes given below the lists;

List-I	**List-II**
A. $T_n = T_{n-1}+n$	1. $T_n=O(n)$
B. $T_n = T_{n/2}+ n$	2. $T_n=O(n^2)$
C. $T_n = T_{n/2}+ n \log n$	3. $T_n= O(n \log n)$
D. $T_n = T_{n-1}+\log n$	4. $T_n= O(n^3)$

Codes:

	A	B	C	D
(a)	2	1	3	3
(b)	3	1	4	2
(c)	2	3	4	1
(d)	3	1	2	4

[GATE-1999]

Q.18 The recurrence equation

$T(1) =1$

$T(n) = 2\,T(n-1) + n,\ n \geq 2$

Evaluates to

a) $2^{n+1} - n-2$

b) 2^n-n)

c) $2^{n+1}-2n-2$

d) $2^n + n$

[GATE-2004]

Q.19 Consider the following program:

```
    Int Bar (int n)
 {
    If (n< 2) return;
    Else
 {
    int sum = 0;
    int t, j;
    for (i = 1; i < = 4; i++) Bar (n/2);
    for (i = 1; i < = n; j++)
    for (j = 1; j < = i; j++)
    sum = sum + 1;  } }
```

Now consider the following statements:

S_1: The Time Complexity of Bar(n) is $\Theta(n^2 \log (n))$.

S_2: The Time Complexity of Bar(n) is $\Omega(n^2 \log (n^2))$.

S_3: The Time Complexity of Bar(n) is $O(n^3 \log (n^2))$.

The number of correct assertions are___

a) 0

b) 1

c) 2

d) 3

Q.20 Two alternative packages A and B are available for processing a database having 10^k records. Package A required 0.0001 n^2 Time units and package B requires 10 n $\log_{10} n$ Time units to process n records. What is the smallest value of k for which package B will be preferred over A?

a) 12

b) 10

c) 6

d) 5

[GATE-2010]

Q.21 Let W(n) and A(n) denote respectively, the worst case and average case running Time of an Algorithm executed on an input of size n. Which of the following is ALWAYS TRUE?

a) $A(n) = \Omega(W(n))$

b) $A(n) = \Theta(W(n))$

c) $A(n) = O(W(n))$

d) $A(n) = o(W(n))$

[GATE-2012]

Q.22 Let f(n) = n and g(n) = $n^{(1+\text{sign } n)}$, where n is a positive integer. Which of the following statements is/are correct?

I. $f(n) = O(g(n))$

II. $f(n)=\Omega(g(n))$

a) Only I

b) Only II

c) Both I and II

d) Neither I nor II

[GATE-2015]

Q.23 Write the following statements True or False

a) if f(n) = O(g(n)) then g(n) = O(f(n))

b) $f(n) + O(f(n) = \Theta$ (min (f(n), g(n))

c) $f(n) + o(f(n)) = \Theta$ (f(n))

d) (logn)! And {loglogn}! are polynomial bounded

Q.24 Consider the following functions from positive integers to real numbers:

$10, \sqrt{n}, n, \log 2^{n}, \frac{100}{n}$

Tire CORRECT arrangement of the above functions in increasing order of asymptotic complexity is:

a) $\text{Log}_2 n, \frac{100}{5}$ 10, $\sqrt{n}$, n

b) $\frac{100}{n}$, 10, $\log_2 n$, $\sqrt{n}$, n

c) 10, $\frac{100}{n}$,$\sqrt{n}$, log2 n, n

d) $\frac{100}{n}$, $\log_2 n$, 10, $\sqrt{n}$, n

[GATE-2017]

P.25 Let T(n) be the function definedby

$$T(n) = \begin{cases} T(1) \text{ for } n=1 \\ 7T(n|2) + 18n^2 \text{ for } n \geq 2 \end{cases}$$

Now which of the following statement is true?

a) $T(n) = \Theta(\log 2n)$

b) $T(n) = \Theta(n^{\log_2 7})$

c) $T(n) = O(n^{\log_2 n})$

d) None of these

Q.26 Consider the following C function:

```
int fun {int n)
int i, j;
for (i = 1; i <= n; i++)
for (j =1; j < n; j + = i)
printf {"%d %d", i, j);
```

Time Complexity of fun in terms of Θ notation is

a) $\Theta(n\sqrt{n})$

b) $\Theta(n^2)$

c) $\Theta(n\log n)$

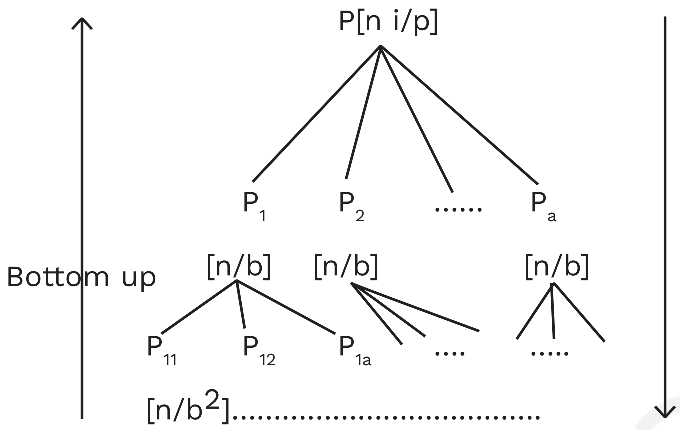
d) $\Theta(n^2 \log n)$

[GATE-2017]

Divide and Conquer

Divide and Conquer Methods

Problem (P) with n i/p's P is not small problem. Divide "P" with n i/p's into "a" sub-problems [P_1 P_2....Pa] with n/b inputs each && continue divide until each sub-problem become small.



until each sub-problem become small problem && return solution of small problems

Return solution of small problems && combine solutions in bottom up approach.

- Sub-problems must be independent.

Control Abstraction of Divide and conquer:-

Algorithm DAC (n)

```
 // Problem P with n i/p's
if(n=1)                    :small problem
return (solution(P))
Else
   || Divide
Divide P into "a" sub-problems $P_1$ $P_2$....... $P_a$ with n/b i/p each
```

Conquer $\left(\underbrace{\text{DAC} \left(\frac{n}{b}\right)}_{T(n/b)}, \underbrace{\text{DAC} \left(\frac{n}{b}\right)}_{T(n/b)}, \ldots\ldots\ldots \underbrace{\text{DAC}\ (n/b)}_{T(n/b)} \right)$

Recurrence Resolution of DAC.

T(n) : Time Complexity of DAC with n i/p's

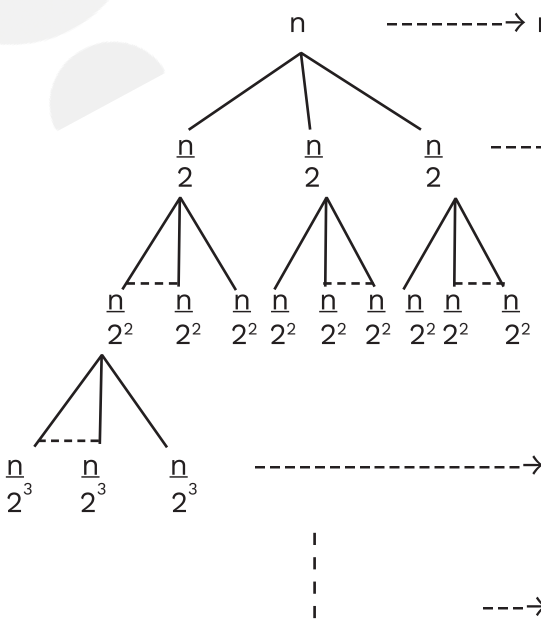
g(n) : Time Complexity to solve small problem

f (n) : Time Complexity to Divide && combine

$$T(n) = \begin{cases} g(n)\ ;\ n=1 \text{ (small problem)} \\ aT\left(\frac{n}{b}\right) + f(n)\ \ n>1 \end{cases}$$

Example:-

1. $T(n) = \begin{cases} 8T(n/2) + n\ , \text{ for } n>1 \\ 1\ \ \ ,\ \ \ \text{for } n<=1 \end{cases}$



$\dashrightarrow n^2$

$\dashrightarrow 8\left(\frac{n}{2}\right)^2 = 2n^2$

$\dashrightarrow 8^2 \left(\frac{n}{2^2}\right)^2 = 2^2 n^2$

$\dashrightarrow 8^3 \left(\frac{n}{2^3}\right)^2 = 2^3 n^2$

$\dashrightarrow 8^k \left(\frac{n}{2^k}\right)^2 = 2^{k-1} n^2$

$$T(n) = 8^K T\left(\frac{n}{2^k}\right) + n^2 [1 + 2 + 2^2 \ldots\ldots\ldots\ldots + 2^{k-1}]$$

$$= 8^K T\left(\frac{n}{2^k}\right) + n^2 [2^K - 1]$$

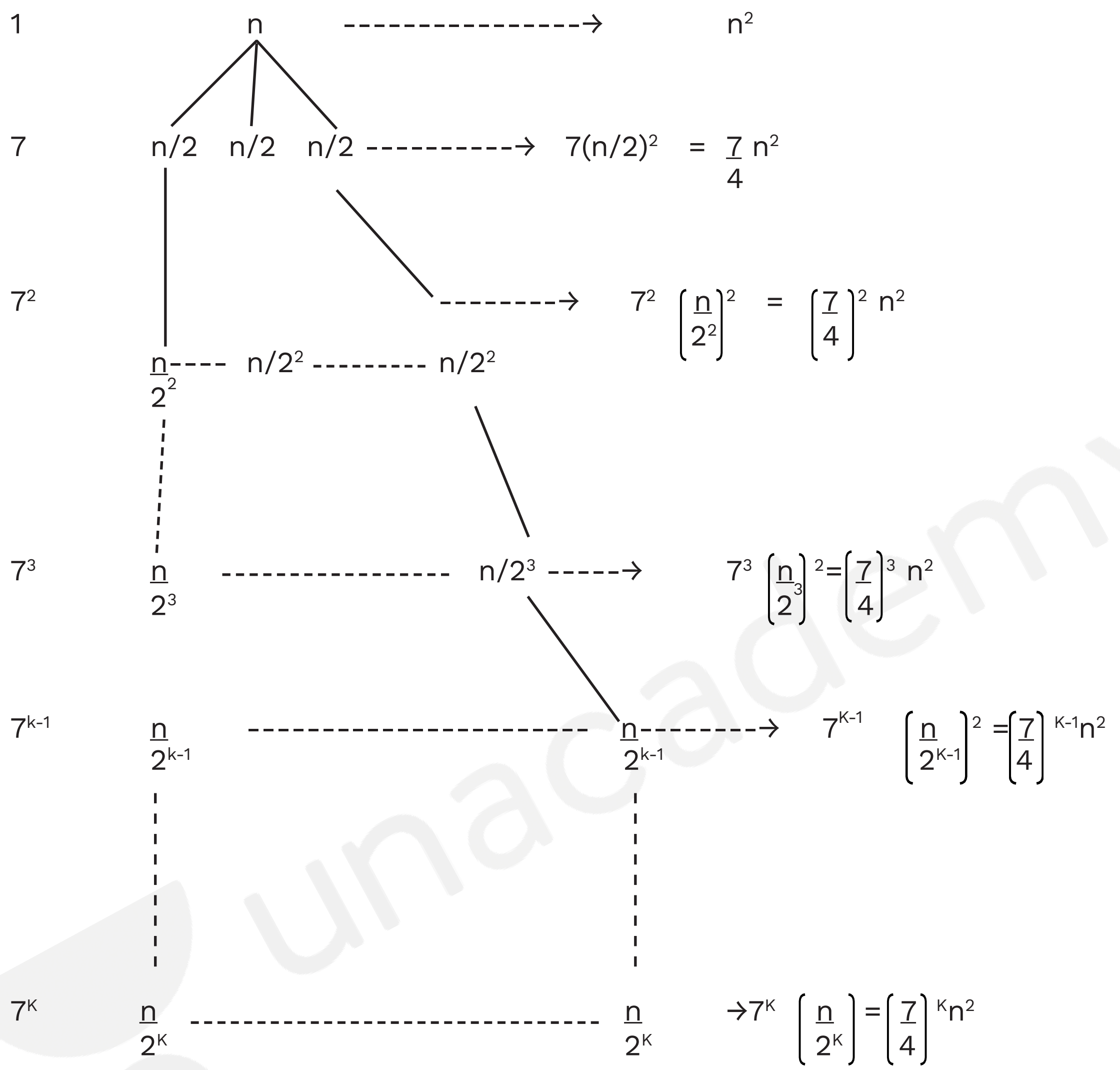
$$\left[\frac{n}{2^k} = 1, K = \log_2 n\right]$$

$$= 8^{\log_2 n} T(1) + n^2 [n-1]$$

$$= n^3 + n^2 [n - 1]$$

$$T(n) = 2n^3 - n^2 = \theta (n^3)$$

2. $T(n)= \begin{cases} 7T\left(\frac{n}{2}\right) + n^2 \text{ , for } n>1 \\ 1 \text{ , } \quad \text{for } n<=1 \end{cases}$


1
n
n²
7
n/2 n/2 n/2
7(n/2)² = 7/4 n²
7²
7² (n/2²)² = (7/4)² n²
n/2² ---- n/2² --------- n/2²
7³
n/2³ ----------------- n/2³
7³ (n/2³)² = (7/4)³ n²
7^{k-1}
n/2^{k-1} ---------------------- n/2^{k-1}
7^{K-1} (n/2^{K-1})² = (7/4)^{K-1} n²
7^K
n/2^K ------------------------------ n/2^K
7^K (n/2^K) = (7/4)^K n²

$$T(n) = 7^K T\left(\frac{n}{2^K}\right) + n^2 \left[1 + \frac{7}{4} + \left(\frac{7}{4}\right)^2 + \left(\frac{7}{4}\right)^3 \cdots\cdots \left(\frac{7}{4}\right)^{K-1}\right]$$

$$\left[.. \ \frac{n}{2^K} = 1, \qquad K = \log_2 n\right]$$

$$= 7^{\log_2 n} \, T(1) \ + \ n^2 \left[\frac{\left(\left(\frac{7}{4}\right)^K - 1\right)}{\frac{7}{4} - 1} \right]$$

⇨ $7^{\log_2 n} \quad + \quad n^2 \left[\frac{4}{3}\left[\left(\frac{7}{4}\right)^{\log_2 n} -1\right]\right]$

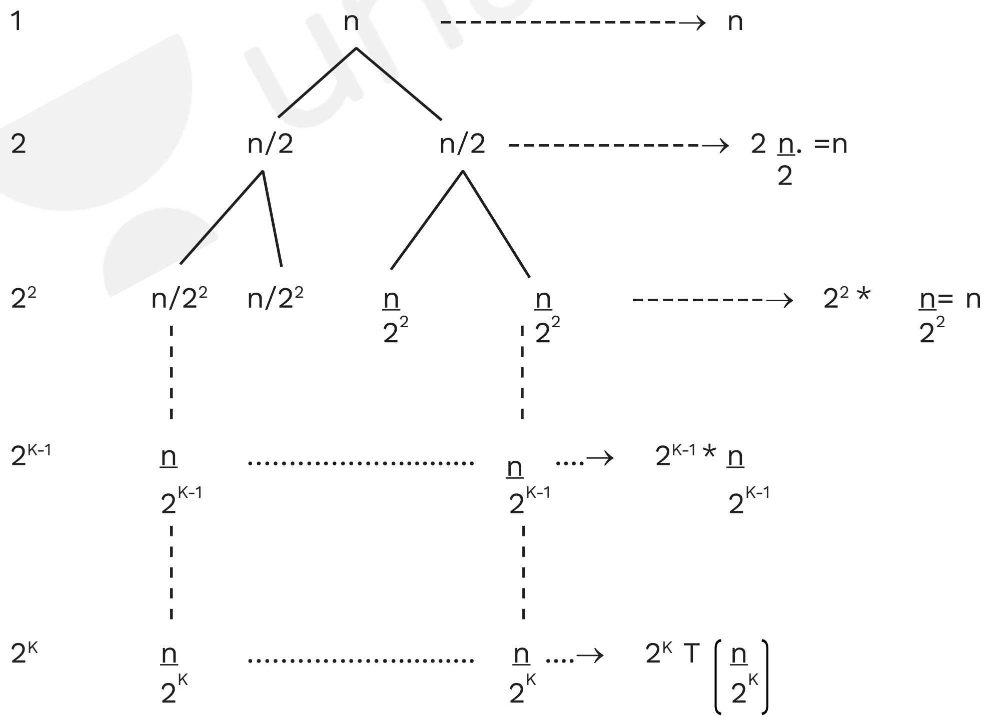
⇨ $7^{\log_2 n} \quad + \frac{4}{3} \quad n^2 \quad \left[\left(\frac{7}{4}\right)^{\log_2 n} -1\right]$

⇨ $7^{\log_2 n} + n^2 \{ (4/3)* (n^{\log_2 (7/4)} \; -1)$

⇨ $n^{\log_2 7} + n^2 * (4/3)*(n^{0.8} -1)$

⇨ $\approx \theta(n^{2.81})$

3. $T(n) = \begin{cases} 2T \;\; (n/2+n \,, & \text{for } n>1 \\ \quad\quad 1 & \text{for } n<=1 \end{cases}$



$T(n) = 2^K T\left(\frac{n}{2^K}\right) + n\ [1 + 1 + \ldots\ldots\ldots\ldots\ldots K - 1]$

$\frac{n}{2^K} = 1 \qquad K = \log_2 n$

$= 2^{\log_2 n} . 1 + n.k.$

$= n + n . \log_2 n = \theta (n \log n)$

Generalization of Master Theorems:

T(n) = aT(n/b) + f(n) where a ≥ 1 & b > 0 constants

Case 1: if $n^{\log_b a}$ polynomial bigger than f(n)

Then $T(n) = \theta (n^{\log_b a})$

Case 2: if $n^{\log_b a}$ & f(n) Asymptotically equal

Then, $T(n) = \theta (n^{\log_b a} * \log_b n)$ depth of recursion

Case 3: if f(n) polynomial bigger than $n^{\log_b a}$

then, $T(n) = \theta (f(n))$

		f(n)	$n^{\log_b a}$
Q1:	$T(n) = T(n/2) + \sqrt{n}$	$n^{0.5} > n^{\log_2 1} = n^0$	
	$= \theta(\sqrt{n})$		

Q2: $T(n) = 2T(n/2) + \sqrt{n} \quad n^{0.5} < n^{\log_2 2} = n^1$

$= \theta(n^{\log_2 2})$

$= \theta(n)$

Q3: $T(n) = T(n/3) + c \qquad \underbrace{C \qquad n^{\log_3 1} = n^0 = 1}_{\text{asympotically equal}}$

$= \theta(\log_3 n)$

Q4: $T(n) = 9T(n/3) + n^2 \qquad n^2 \qquad n^2 = n^{\log_b a}$

$= \theta(n^2 \log_3 n)$

Q5: $T(n) = 26T(n/3) + n^3 \qquad n^3 \qquad n^{\log_3 26} = n^{2.9}$

$T(n) = \theta(n^3)$

Q6: $T(n) = 27\,T(n/3) + n^3 \qquad n^3 \qquad n^{\log_3 27} = n^3$

$T(n) = \theta(n^3 \log_3 n$

Q7: $T(n) = 28\,T(n/3) + n^3 \qquad n^3 \qquad n^{\log_3 28} = n^{3.1}$

$T(n) = \theta(n^{\log_3 28})$

Examples failed by master theorem

$T(n) = 2T(n/2) + n\log n$

$n^{\log_b a}$ & (n) are logarithmic equal but polynomial not equal.

Master theorem failed

Change of Variables

$T(n) = a\,T(b\sqrt{n}) + f(n)$

Change of variable

$\downarrow \quad [n = b^{n}\ \sqrt[b]{n} = b^{n/b}]$

$T(b^m) = aT(b^{m/b}) + g(m)$

$T(b^m)\ s(m) \Rightarrow T(b^{m/b}) = s(m/b)$

$\downarrow \quad s(m) = a\,s(m/b) + g(m)$

Apply master theorem to solve

Q. Algorithm rec(n)

```
{
    if ($n \le 2$)
        return
      else
          return( rec ($\sqrt{n}$) + rec ($\sqrt{n}$) + n )
}
```

Solution: T (n) = Time Complexity of rec (n)

$$T(n) = \begin{cases} 1, & \text{for } n <= 2 \\ 2T(\sqrt{n}) + 1, & \text{for } n > 2 \end{cases}$$

$\{ n = 2^m \Rightarrow \sqrt{n} = 2^{m/2} \}$

$T(2^m) = 2T(2^{m/2}) + 1$

.. $T(2^m) = s(m) \Rightarrow T(2^{m/2}) = s(m/2)$

$s(m) = 2\,s(m/2) + 1$

Apply Master Methods

$S(m) = \theta(m)$

$S(m) = T(2^m) = T(n) = \theta(m)$

$T(n) = \theta(\log_2 n)$

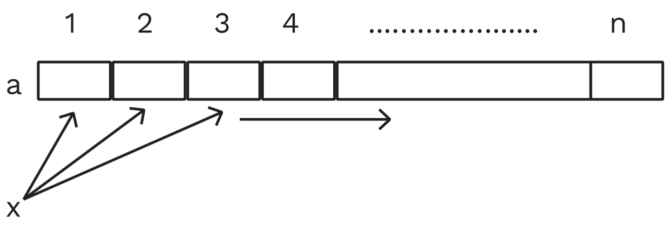
Examples of DAC

- → Binary search
- → Quick Sort
- → Merge Sort
- → Max – Min Algorithm
- → Strassens's Integer Multiplication
- → Strassens's Matrix Multiplication
- → Fast exponential Algorithm

Binary Search;

1) Linear Search :

a [1 ....n] array of n elements x searching element



Algorithm linear search (a, n, x)

```
|| a [1...n] element & x is searching element
   for (i = 1 ; i ≤ n ;i++)
    {
       if (x = = a [i] )
            return (i) ;
     }
          return (-1) || unsuccessful search
  }
```

Best case Time Complexity:- [min CPU Time required to terminate Algorithm]

if x present 1st position of array

number of comp = 1

Time Complexity = θ (1)

Worst case Time Complexity:- [Max CPU Time required to terminate Algorithm]

if x present last position / x not in array

number of comp = n

Time Complexity= θ (n)

Average Case Time Complexity:-[Average CPU Time required to terminate Algorithm]

$$\text{number of comparisons} = \frac{1 + 2 + 3 + \ldots\ldots\ldots\ldots + n}{n}$$

$$= \frac{n\,(n+1\,)}{2*n} = \frac{n+1}{2}$$

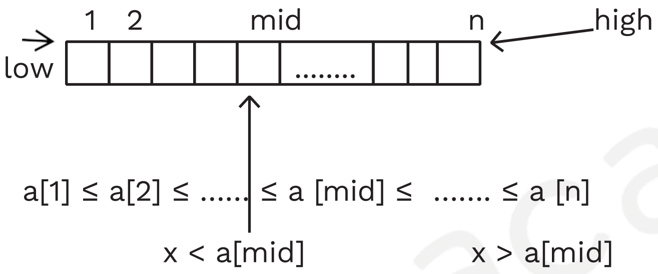
Time Complexity = θ (n)

Space complexity = θ(1) || excluding i/p space

Binary Search Algorithm:

a[1....n] array elements must be in sorted order

X : Searching element

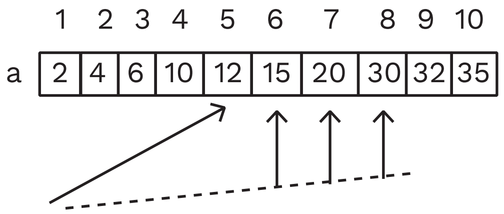


a[1] ≤ a[2] ≤ ...... ≤ a [mid] ≤ .... ≤ a [n]

x < a[mid]   x > a[mid]

```
Algorithm Binary Search (a, n, x)
  {
      low = 1 ; high = n
      while (low ≤ high)
  {
      mid = $\frac{low + high}{2}$
      if (x < a [mid] )
            high = mid – 1 ;
      else if ( x > a [mid] )
```

```
                low = mid + 1 ;
            else
            return (mid) ;
        }
            return (-1)
        }
```



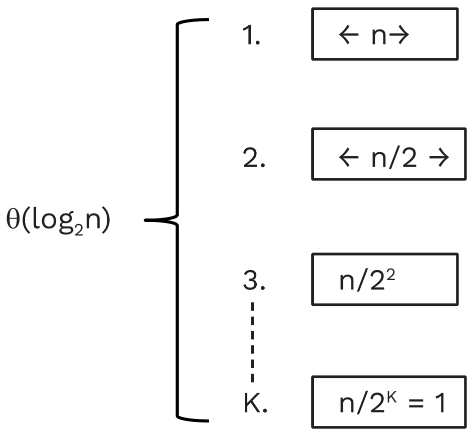
x = 25 =>	1) low =	1 ,	high = 10	=>	mid = 5
	2)	6	10	=>	8
	3)	6	7	=>	6
	4)	7	7	=>	7
		[8	7]		return(-1)
		low	high		// not found

Best case Time Complexity:- θ (1)

Min number of while loop to terminate Algorithm = 1. if “x” present in middle of array

Best Case Time Complexity of Binary Search = θ (1)

Worst Case Time Complexity:- θ (1)



Average case Time Complexity:-θ(logn)

$$\frac{1 + 2 + 3 + \ldots\ldots\ldots\ldots + \log n}{\log n} = \frac{1 + \log n}{2} = \theta(\log n)$$

Space Complexity of Binary Search : θ(1) || excluding i/p array space

Recursive Binary Search :

```
Algorithm RBS (low, high)

    {
            || a [ low ..... high ] is an array of n sorted elements & x is searching element
          ┌ if ( low ≥ high) || small problem
O(1)      │     {
         ─┤         if (x = = a [low] )
          │             return (low) ;
          └     }

              else

              {
           ┌   mid = (low + high )/2
           │   if (x < a [mid] )
           │       RBS (low, mid-1) ; => T(n/2)
T(n/2)+1   │         else if ( x > a [mid])
           │           RBS (mid + 1 , high) ; => T(n/2)
           │       else
           └           return (mid) ;
                   }
              }
```

Best Case Time Complexity:-

θ (1) || x is present middle of array

Worst Case Time Complexity. && Average case Time Complexity.

Time Complexity recurrence relation of Binary Search

$$T(n) = \begin{cases} 1 & , \quad \text{for } n<=1 \\ T(n/2)+1 , & \quad \text{for } n>1 \end{cases}$$

$= \theta(\log_2 n)$

Space Complexity of recursive Algorithm Binary Search :-

Depth of recursive tree : $\log_2 n$

$= O(\log_2 n)$ || stack space

- Overhead is much more in recursive
- Non Recursive binary search preferred than Recursive Bin search because of

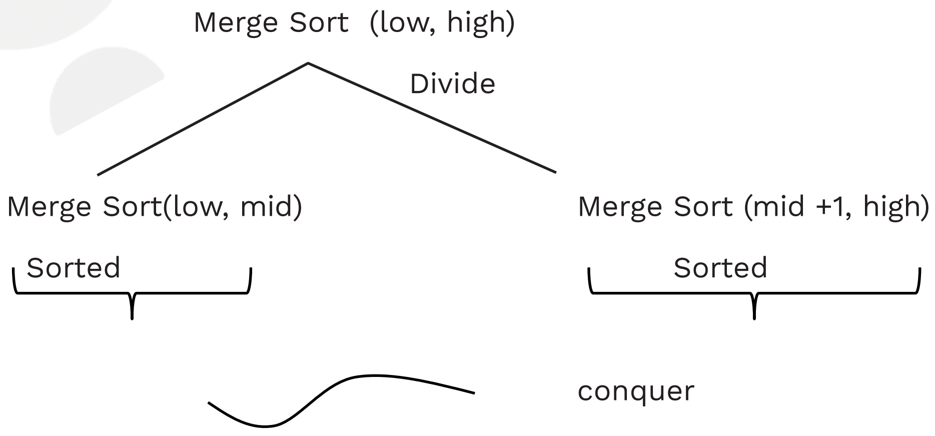
1] Space Complexity : $\theta(1)$

2] No overhead of function calls

Merge Sort Algorithm [fully DAC]

Given array of n elements

a [low .... high] array of n elements.

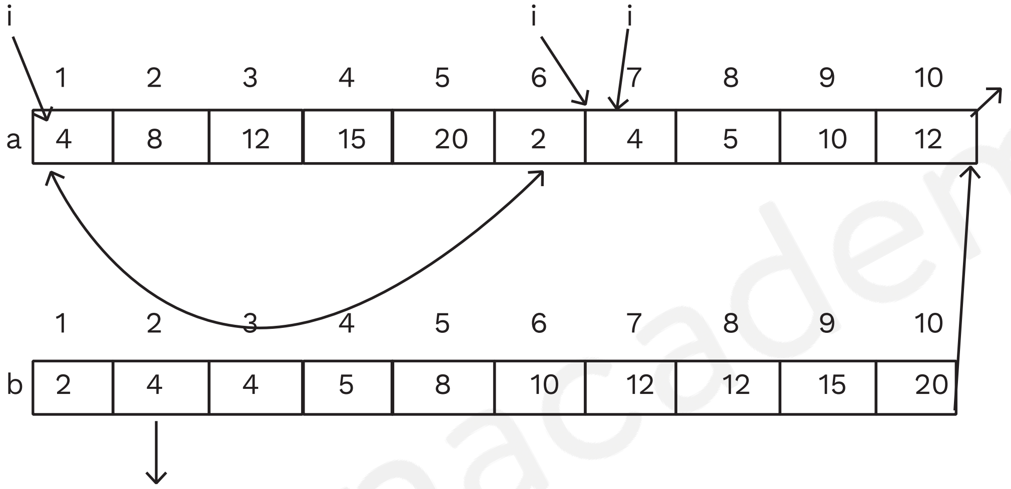


Merge (low, mid, high)

[Merges two sorted parts of array a [low...mid] && a[mid +1, high] into single sorted array]

```
                                    → T(n)
Algorithm Merge Sort (low, high)
   {
      || a[low, high] is array of n elements
        if (low < high) || more than one element
       mid = (low + high) /2
   }


   1.  Merge Sort (low, mid); => T(n/2)
   2.  Merge Sort (mid +1, high); => T(n/2)
       || conquer
   3.  Merge (low, mid, high) => cn

                                     → θ((n)
Algorithm Merge (low, mid, high)
   {                                       n/2          n/2
      || Merge two sorted arrays a [low...mid] && a[mid + 1, high] into single
       sorted array ||
   i = low ; j = mid + 1 ; k = low

      While (i ≤ mid && j ≤ high)
       {
         if (a[i] ≤ a[j] )}
       {
          b [k] = a [i];
         i++
        }
      else
       { b [k] = a [j] ;
             j++ ; }
         k++
      }
      if (i > mid )
        for ( x =j ; x ≤ high ;x++)
      { b[k] = a[x] ; k++ }
```

```
          else
         for (x=i ; x ≤ mid ; x++)
           {
                b [k] =  a [x] ;  k++}
       n { for (x = low ; x ≤ high ;x++)
          {
                a[x] = b[x] ;
          }
        }
```



	1	2	3	4	5	6	7	8	9	10
a	4	8	12	15	20	2	4	5	10	12

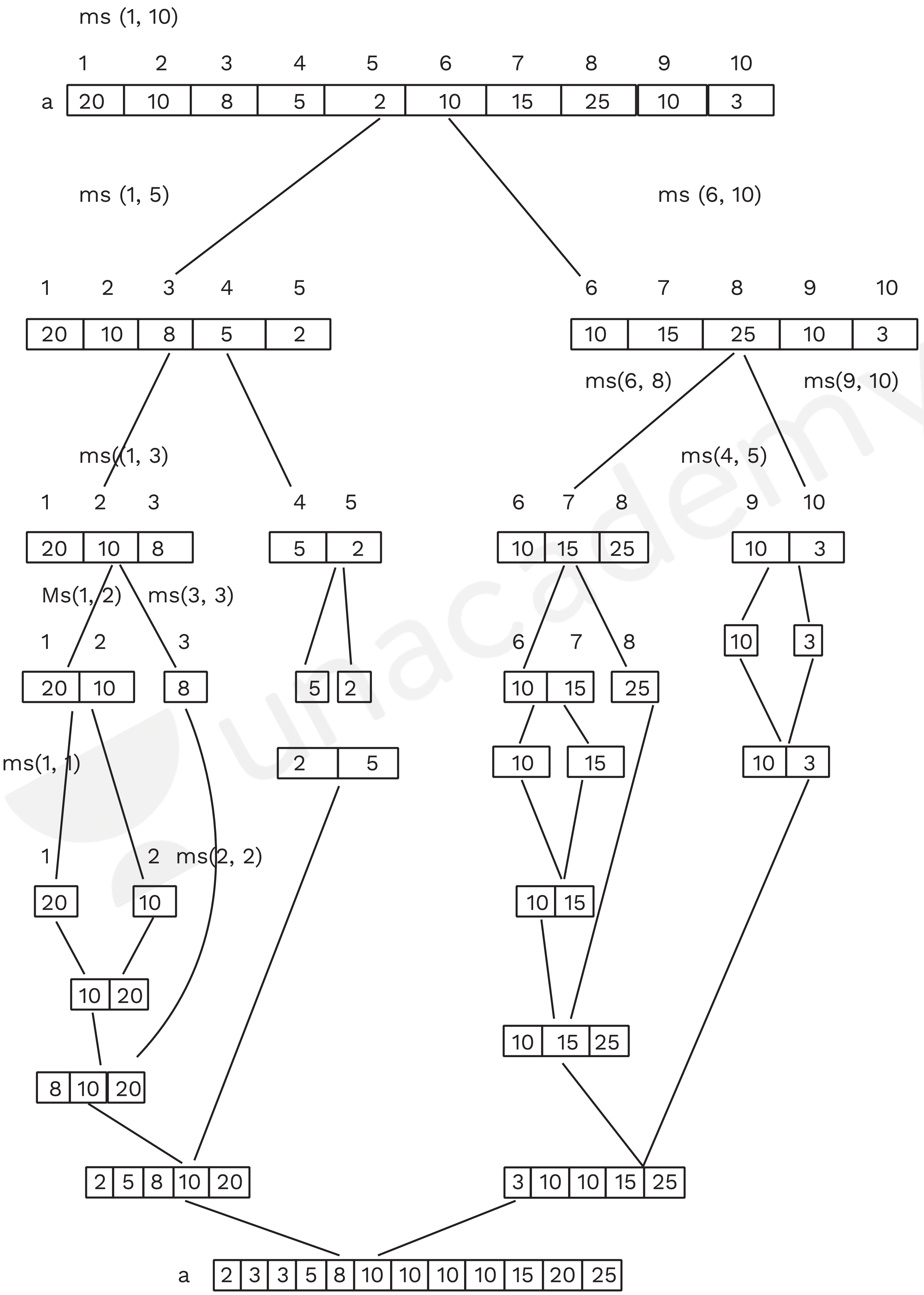
	1	2	3	4	5	6	7	8	9	10
b	2	4	4	5	8	10	12	12	15	20

|| Sorted

(i value written)

Whichever value will be less than will be written and increment its index

	1	2	3	4	5	6	7	8	9	10
a	20	10	8	5	2	10	15	25	10	3



ms (1,1)

ms (1,2)

ms (1,3)

ms (1,5)

ms (1,10)

Stack

Merge Sort time complexity recurrence rel

$$T(n) = \begin{cases} 2T(n/2) + cn, & \text{for } n > 1 \\ 1, & \text{for } n <= 1 \end{cases}$$

$= \theta(n \log_2 n)$ || (All cases BC, WC && AC)

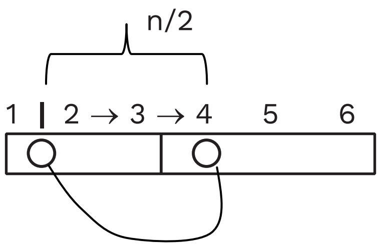
Depth of Recursion : $\theta(\log_2 n)$ || All cases of Merge sort

Space Complexity of Merge Sort

Auxiliary array ... b [1...n] = θ (n)

Stack Space.... $\log_2 n = \theta(\log n)$

SC of Merge Sort : θ (n)



If auxiliary array b[ ] not used merge cost to merge two sorted parts of array n/2 element each into single sorted array O(n^2) (worth case)

To place one element 2nd list into correct position required n/2 array swift operations

Total Merge Cost $\frac{n}{2} * n/2 = O(n^2)$

Stable Sorting Algorithm:-

Identical elements of i/p array must occupy same order in result of sorting Algorithm.

	1	2	3	4	5	6	7	
Input a	4	2a	0	2b	5	2	3	Sorting Algorithm

↓

	1	2	3	4	5	6	7	
Output a	0	2a	2b	2c	3	4	5	Result of Stable sorting Algorithm

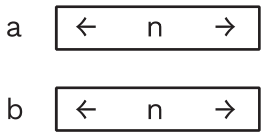
	1	2	3	4	5	6	7	
Output a	0	2b	2c	2a	3	4	5	Unstable Sorting Algorithm

Inplace Sorting Algorithm

Given array of n elements a [1...n] if sorting Algorithm uses same i/p array to sort list then "inplace sorting Algorithm"

+ stack [Recursive calls]

If sorting Algorithm uses extra array other than i/p array to sort the list then Not inplace sorting algorithm.



+ Stack (Recursive call)

Merge sort is stable sorting Algorithm but not inplace sorting Algorithm.

Problems:-

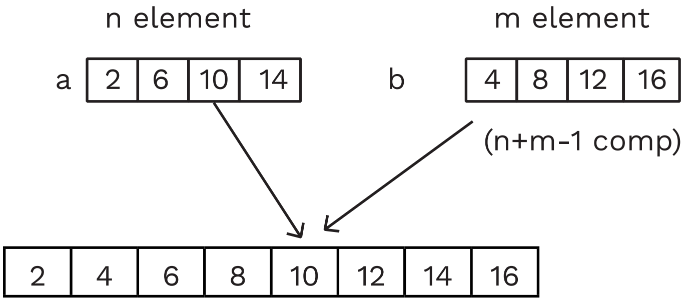
Q. How many max element comparisons required to merge two sorted array with n & m elements each into single sorted array.

a) n - m

b) n + m

c) n + m -1

d) n*m



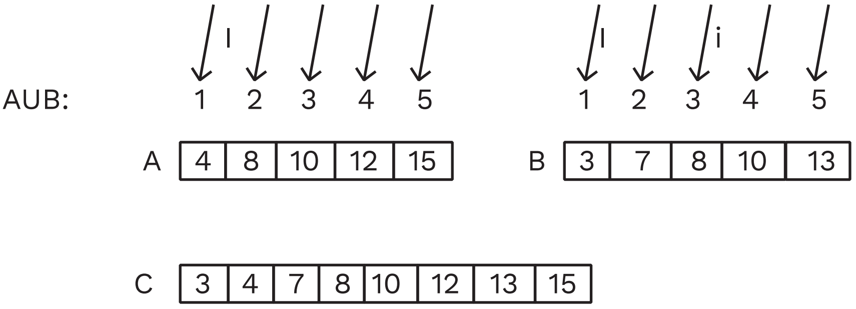
TC = θ(n +m)

Q. Given two sorted arrays “A”, “B” with n elements & m elements each what is TC to perform.

i) AUB

ii) A∩B

iii) A –B



A and B should contain distinct elements Algorithm union (A[n], B[m]) TC : θ(n+m)

```
A, B sorted arrays
i = 1 ; j = 1 ; k = 1
while ( i ≤ n && j ≤ m)
 {
      if (a[i] < b [j] )
        c[k] = a[i] ; i++}
else if ( a[i] >  b[j] )
{c[k]  = b[j]  ;  j++}


else
     {c[k] = a[i] ; i++ ; j++}
     K++
 }
if (i > n) for (x =j ; x ≤ m ;x++)
          { c[k] = b[x] ; k++;}
else
     {for (x =i ; x ≤ n ; x++}
        {c[k] = b[x] ; k++; }
```

AUB : Time Complexity = θ(n +m)}

2. A∩B = Time Complexity : θ (n+m)

Algorithm intersect (A[n], B[m])

Time Complexity = θ (n +m)

```
     {   i = 1 ; j =1 ;k=1
         while ( i ≤ n && j ≤ m)
        {
             i++;
          }
     else if (a [i] > b[j] )
         {
             j++ ; }
     else { c[k] = a[i] ; i++ ; j++ }
             k++
         }
```

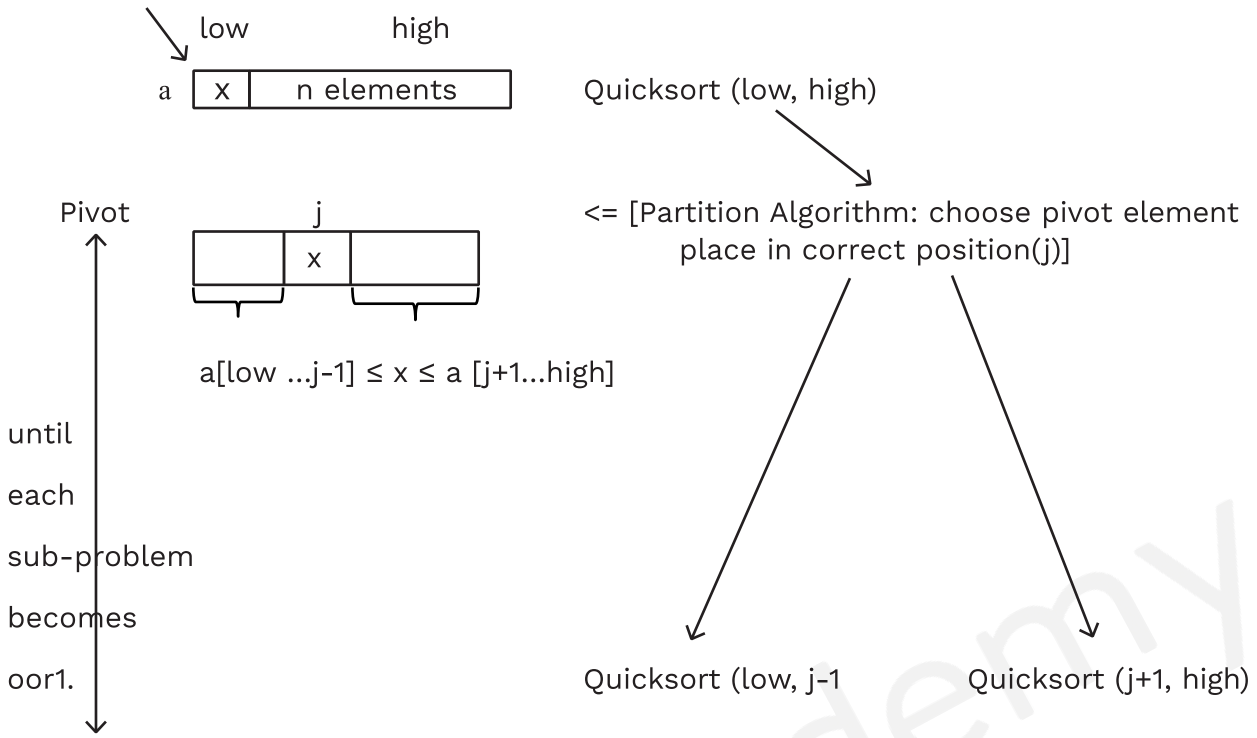
3. A-B : Time Complexity = θ (n + m)

```
       while ( i ≤ n && j ≤m)
         {
            if (a[i] < b[j]
        {
             c[k] = a[i] ; i++; }
     else if (a[i] > b[j] )
        {  j++;}
     else
           { i++ ; j++ }
        k++
        }
if (i ≤ n)
     for (x = i ; x ≤ n ;x++)
        {
             c[k] = a[x] ; k++ }
     }
```

Quick Sort Algorithm:-[Places pivot element into correct position]

- Widely used sorting algorithm
- Inplace sorting but not stable sorting
- Tony gorge (Turing Award: (Pi development))

Given array of n elements



number of elements (high - low +1)

```
Algorithm Quicksort (low, high)
 {
|| a[low...high]  array of n elements
        if (low < high) || more than one element}
  {
        j = partition (a, low, high( ;
           Quicksort-(low, j-1) ;
           Quicksort (j +1, high) :
     }
   }
```

Partition Algorithm

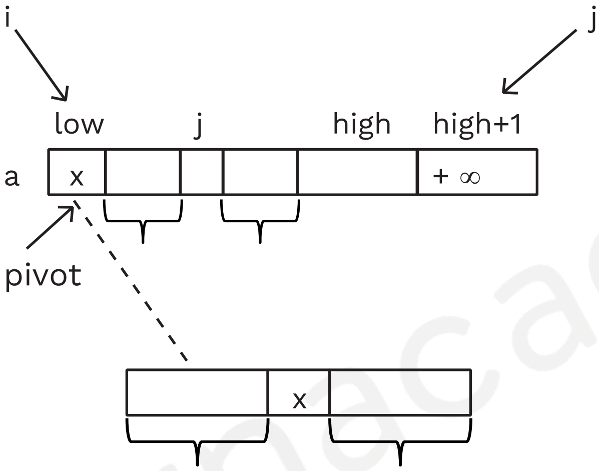
Time Complexity : θ (n).

Procedure :-

i = low high + 1= j ; a[high + 1] = + ∞ ;

x = a [low] ;

1. Increment i until a[i] > x
2. Decrement until a[j] < x
3. if (i < j) swap (a[i], a[j]
4. Repeat ①②③ until i ≥ j
5. Swap (a [low], a[j])
6. Return j

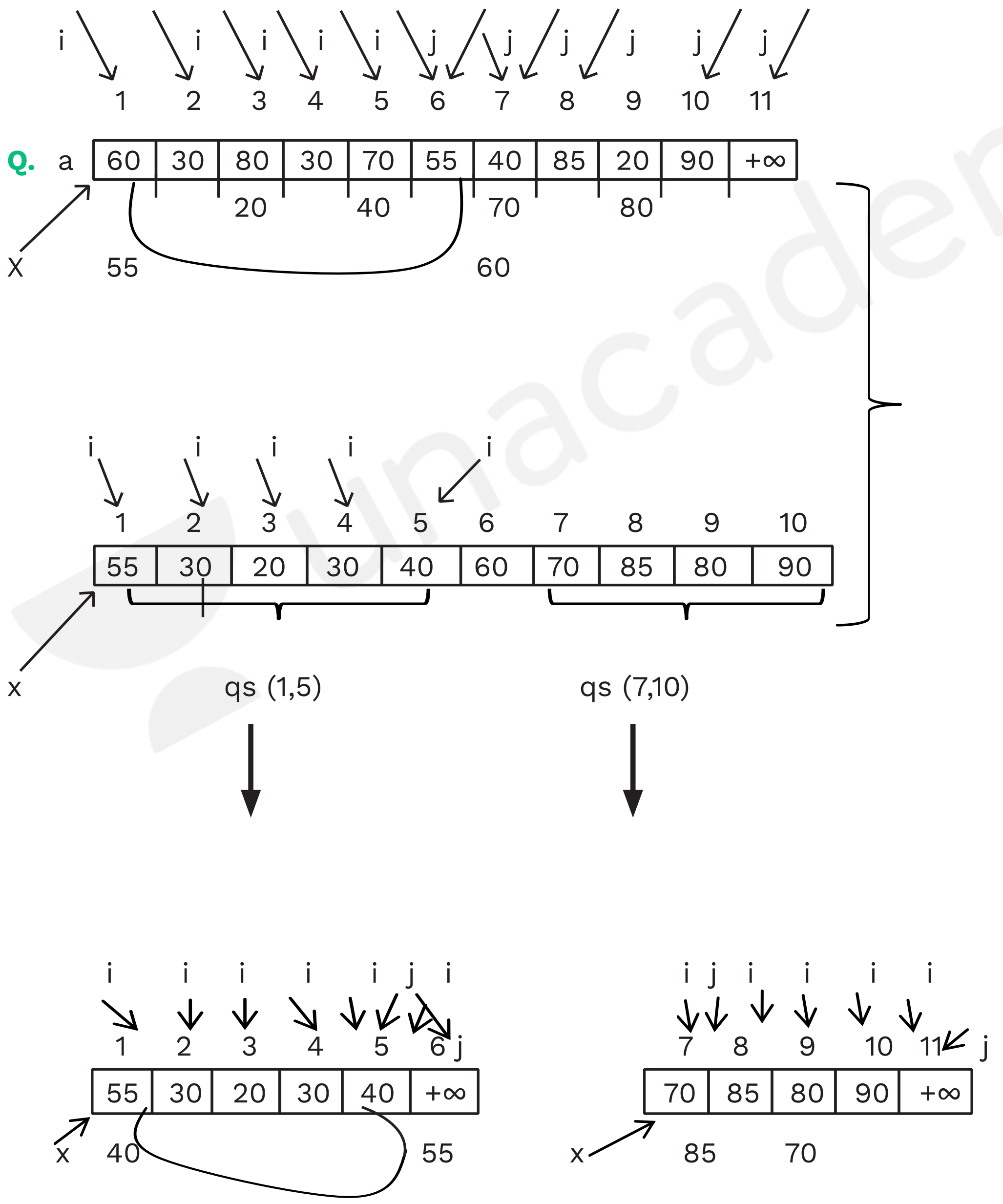


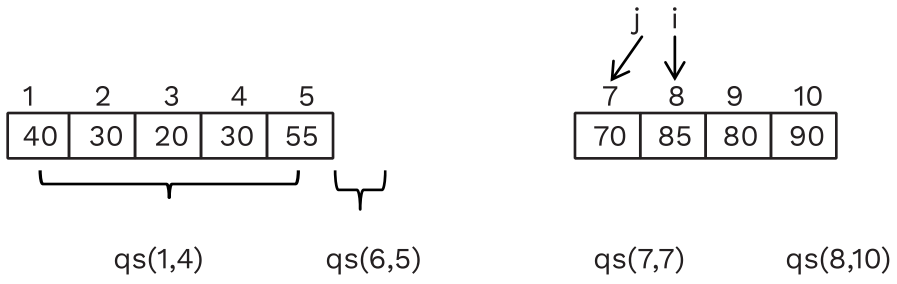
Algorithm partition (a, low, high) => θ(n)

```
{
   i = low ; j=high+ 1;          a[high +1] = +∞
    x =a[low];                          ↓
       ↘
        pivot                  To avoid array out of bound access.
        do
        {
          1.  do
              { i = i +1 ;
              } while  (a[i]< x)  →        if (we use i < high here instead +∞ no.
                                           of computation cost increases)
```

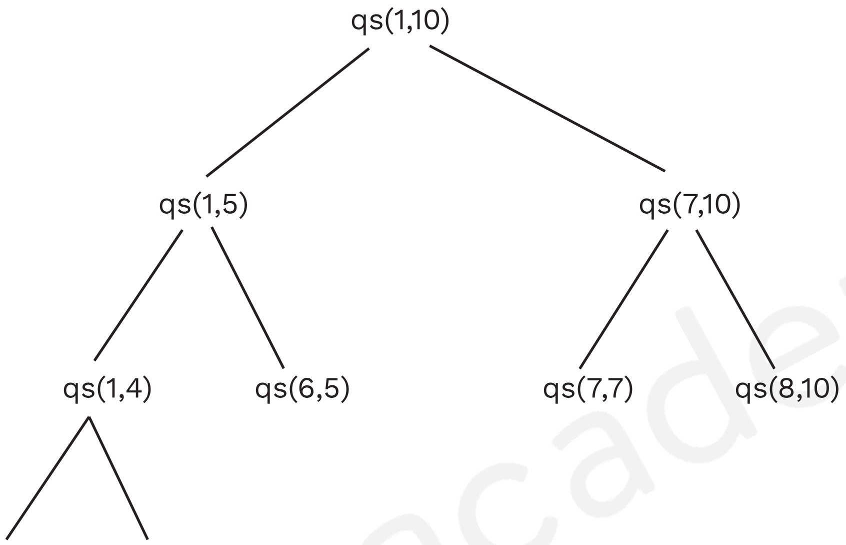
```
2.  do
      { j =j-1 ;
      } while (a [j] > x)

3.  if {(i < j) swap (a[i] , a[j] ) ;
    }  while ( i < j)
     swap (a [low] , a[j] ;
     return (j);
    }
```

Q.





[list “O” element]

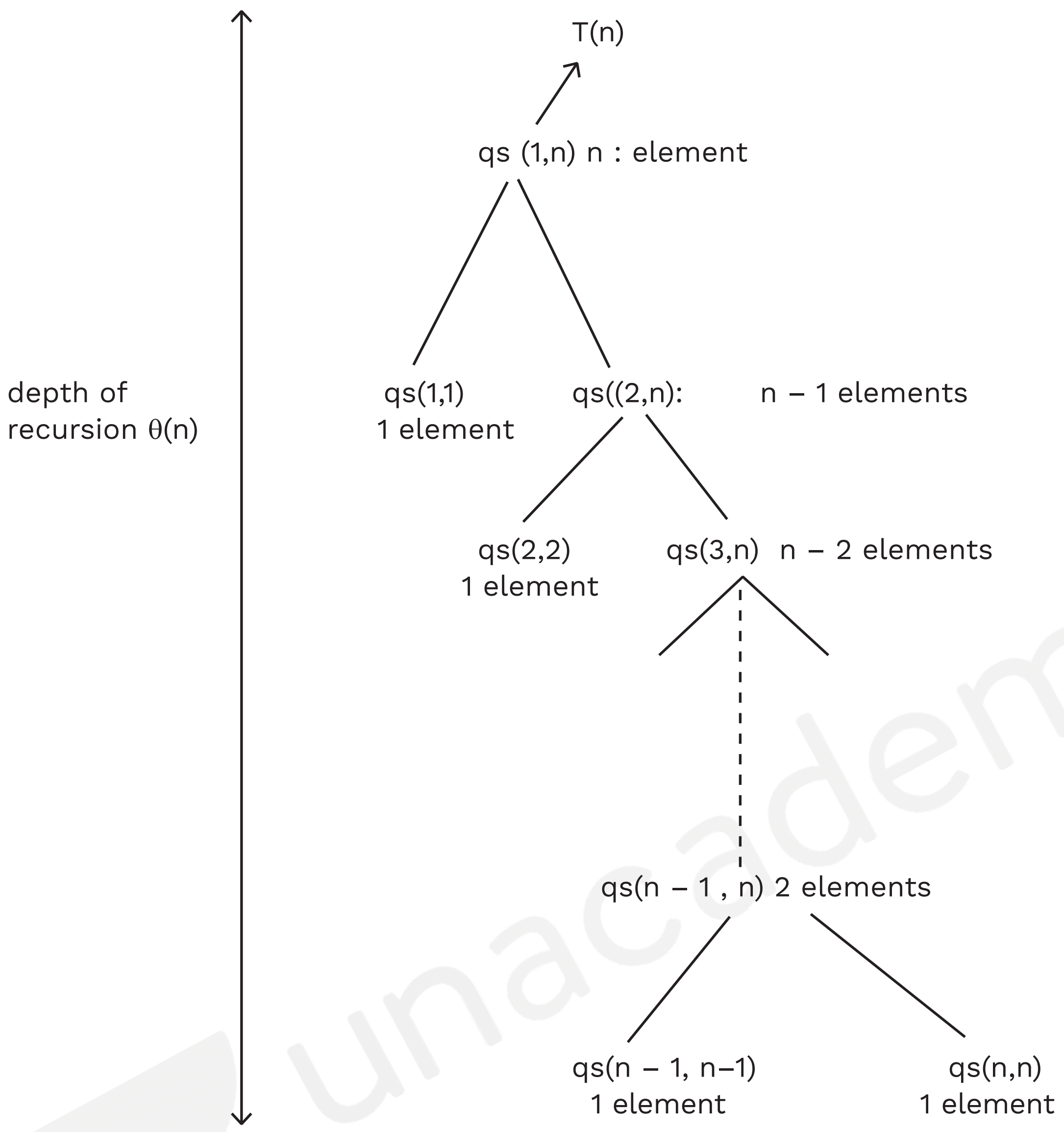


Time Complexity of Quicksort-

1. Worst Case Time Complexity

Quick sort behaves Worst Case if input array already in sorted order.

	1	2	3	4	5	n
a	2	4	6	8	10	



Worst Case of Quicksort recurrence relation:-

$$T(n)= \begin{cases} T(n-1) + cn\ , & \text{for } n > 1 \\ 1\ , & \text{for } n <= 1 \end{cases}$$

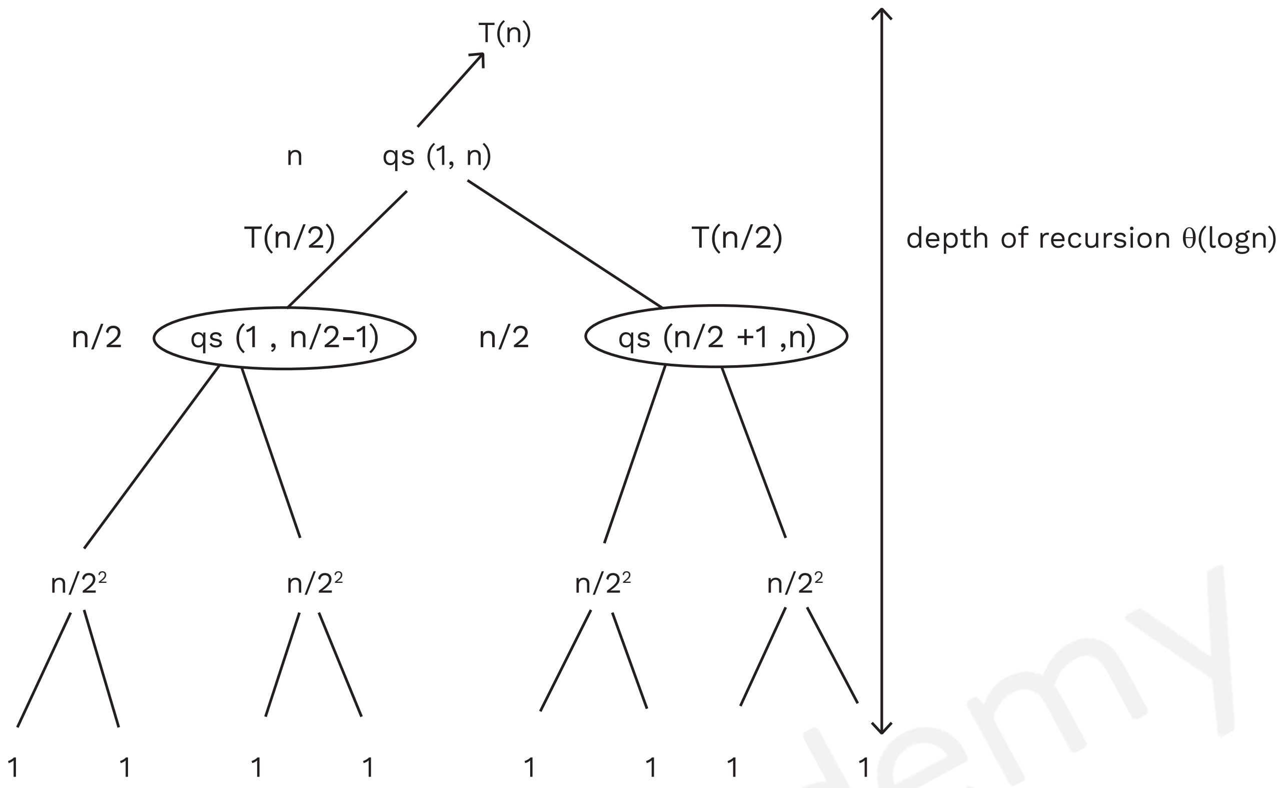
$T(n) = \theta\ (n^2)$

Max depth of recursion of quicksort : θ(n)

2. Best Case Time Complexity of Quick Sort:-

If pivot element places middle of list [partition Algorithm divides list into equal parts].

a [1.....n]

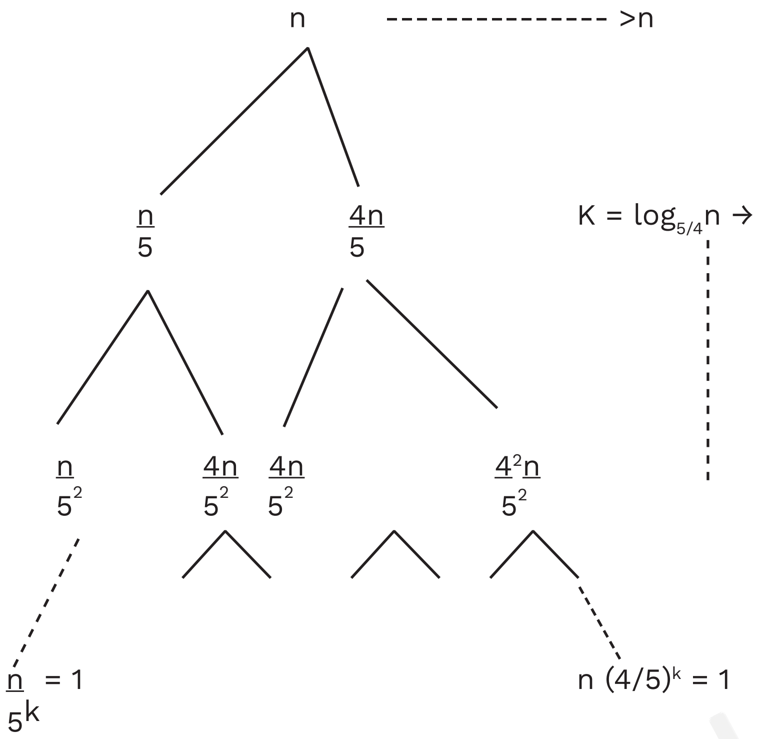


$$T(n) = \begin{cases} 2T(n/2) + cn & \text{, for } n > 1 \\ 1 & \text{, for } n <= 1 \end{cases}$$

$= \theta(n \log_2 n)$ Time Complexity of quick sort in Best Case

Average Case Time Complexity of Quick Sort:-

Array of n elements each time partition algorithm divides list into two sub-problem with $\frac{n}{5}$ && $\frac{4n}{5}$ elements



$$T(n) = \begin{cases} T(n/5) + \underline{T(4n/5)+n}, & \text{for } n > 1 \\ 1 & \text{for } n <= 1 \end{cases}$$

$$= \theta(n \log_{5/4} n) = \theta\ (n\log n)$$

Average Case Time Complexity Relation:-

$$T(n) = \begin{cases} T(\alpha\ n) + T\ (1-\ \alpha) \quad n, & \text{for } n > 1 \\ 1 & , \text{for } n <= 1 \end{cases}$$

Where $0 < \alpha < 1$

$T(n) = \theta\ (n.\log_{1/1-\alpha} n) = \theta(n\log\ n)$

$$\left.\begin{array}{l} T(n) = T\left(\frac{n}{100}\right) + T\left(\frac{99n}{100}\right) + n \\[2ex] T(n) = T\left(\frac{n}{55}\right) + T\left(\frac{54n}{55}\right) + n \\[2ex] T(n) = T(n/8) + T\left(\frac{7n}{8}\right) + n \end{array}\right\} \theta\,(n\log n)$$

Pivot Selection Methods:-

1] First or last element is pivot :-

Worst case Time Complexity qs is $\theta(n^2)$ if i/p array is in sorted array.

2] Average of min && max element of array:-

Example-

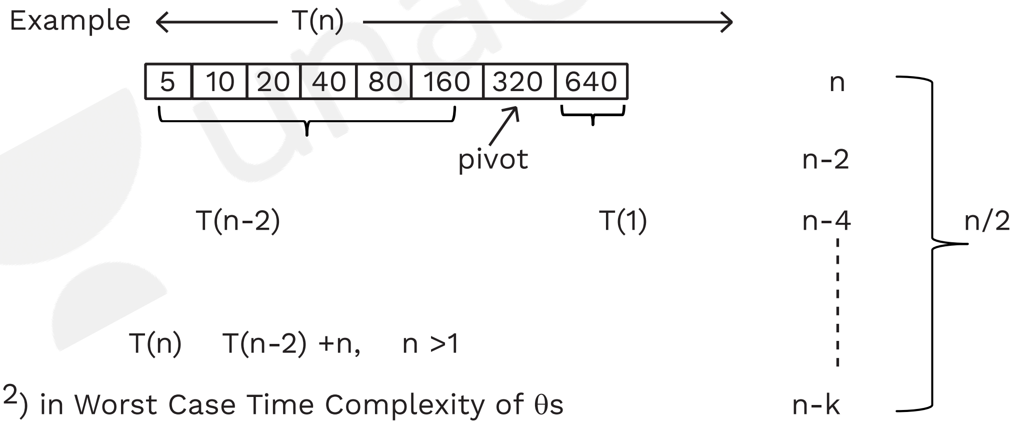
2	4	6	7	10	15	18

pivot

min = 2

max = 25

Average = 13



T(n) T(n-2) +n, n >1

$\theta(n^2)$ in Worst Case Time Complexity of θs

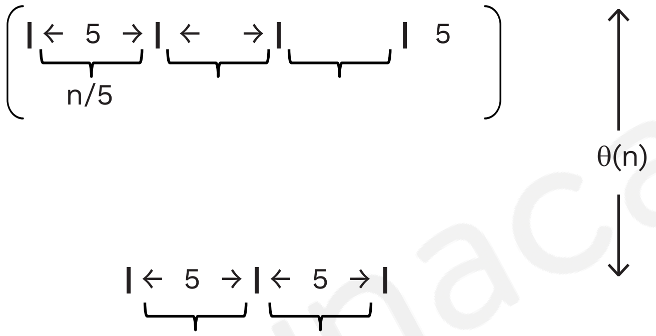
3] Median element of array is pivot element:-

```
Algorithm qs (i , h)
 {
   if (i < h)
    {
     K = median (a, i , h) - - θ(n) | O(n²) | O(nlogn)
```

```
        swap (a[i] , a [K] )
        j = partition (a, i, h) - - n
        qs (i , j-1) - - - T(n/2)
        qs (j + 1, h) - - - T(n/2)
      }
  }
        qs Time Complexity: Ω (nlogn) // All Cases
```

if median element of array can identify in O(n) time then what is worst case of qs. O(nlogn) or Ω (nlogn)

Median



$$c\left(\frac{n}{5}+\frac{n}{5^2}+\ldots\frac{n}{5^K}\right)=\theta(n)$$

(Best Pivot Selection)

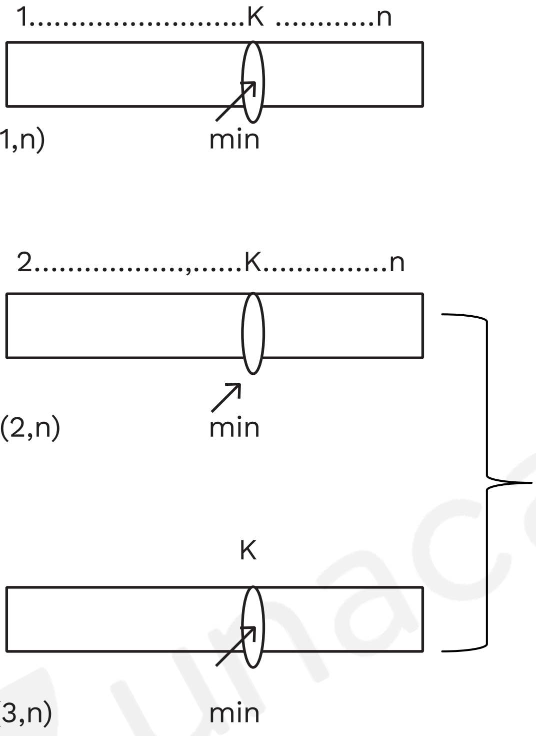
4] Random Pivot Selection Quick Sort:-

```
      Algorithm qs (i , h)
        {
            if (i < h)}
          {
          K = Random ( i, h) - - - θ(1)
```

```
            swap (a [i] , a[k] )
            j = partition (a, i, h) - - - O(n)
           qs (i, j – 1)
           qs (j + 1, h)}
          }
      }
```

qs (1,n) a

K =Random(1,n)



1..........................K ....n
min
2...................,......K...............n
min
K
min

qs (2,n)

K = Random(2,n)

qs (3,n)

K = random(3,n)

[Worst Case Time Complexity of quick sort using Random Pivot selection is $O(n^2)$ time]

Q. Assume of $\frac{n}{5}$ smallest can identify O(n) time

what is Worst Case Time Complexity of qs ?

a) $O(n^2)$

b) O(nlogn)

c) (n)

d) None

```
Algorithm    qs
   {
      if ( i < h)
    {
      K = pivot (a, i, h) O(n) : [given]
      j = Partition ( ) : $\theta(n)$
      qs (i, j -1 ) : T(n/5)
      qs (j+1, n) : T(4n/5)
   }         All casesO(nlogn)
 }
```

Q. Array $_1$[1, 2, 6, 8, 10]

Array $_2$[6, 8, 1, 2, 10]

If t_1 & t_2 comparisons required to sort array$_1$ & array$_2$ in ascending order using Quick sort which is tree

a) $t_1 < t_2$

b) $t_1 > t_2$

c) $t_1 = t_2$

d) None

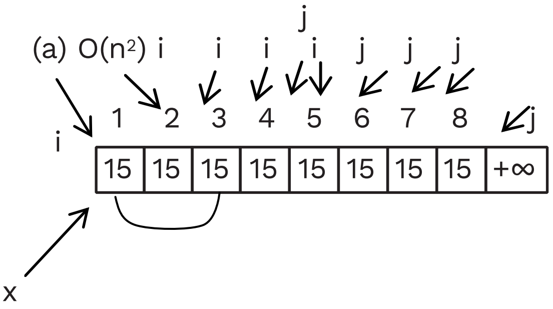
Q. List 1 : [1, 2, 3 ....n]

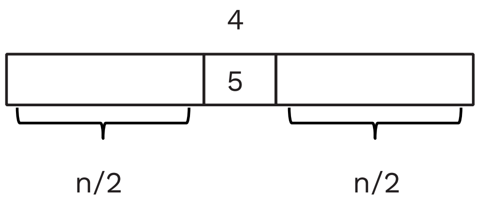
List 2 : [n, n-1, n-2, ..., 2, 1]

t_1 & t_2 comparison required to sort list$_1$ list$_2$in ascending order qs?

= T_1= T_2

Q. Assume all elements of array are identical what is Time Complexity of qs?





= θ (nlogn)

```
do{
      do{
           i++
           }(while(a[i]<=x));
       do{
            j--
           }(while(a[i]>x));
     }
```

$\theta(n^2)$

Answer: depends partition Algorithm implementation

Strassen's Integer Multiplication:-

- Straight Integer multiplication => Time Complexity : $O(n^2)$
- DAC integer multiplication. Time Complexity : $O(n^2)$
- Strassen's DAC integer multiplication. Time Complexity : $O(n^{\log_2 3})$ $O(n^{1.67})$

Integer Addition

X and Y are two integers with n bits each O(1) time addition of two 1 bit's.

$X : X_{n-1}\ X_{n-2}\ldots\ldots\ldots\ldots X_2\ X_1\ X_0$

$Y \Rightarrow Y_{n-1}\ Y_{n-2}\ldots\ldots\ldots\ Y_2\ \ Y_1\ Y_0$

$C_{n-1}\ \ C_{n-2} \qquad\qquad C_2\ C_1\ C_0$

$X + Y \Rightarrow$

$Z_n\ Z_{n-1}\ Z_{n-2} \qquad\qquad Z_2\ Z_1\ Z_0$

Time Complexity to add two "n" bit number : θ(n) || bit addition

Straight Integer Multiplication:-

X and Y are two n bit numbers.

X = 25 , Y = 15, Z = 0

1. X is odd , Z= z+y = 15

   $x=\left\lfloor \frac{x}{2} \right\rfloor$ =12   Y = 2.Y = 30

2. X is even (No Add)

   X= x/2 = 6   Y = 2.Y = 60

3. X is even (No Add)

   $X = \left\lfloor \frac{x}{2} \right\rfloor=3$ Y = 2Y = 120

4. X is odd Ƶ = Z + Y = 135

   X = x/2 = 1 , Y = 2Y = 240

5. X is odd Ƶ= Ƶ + Y= $\textcircled{375}$ return

   X = x/2 = 0   Y = 2Y = 480

X: $\boxed{0\ \ 1\ \ 1\ \ \overline{0\ 0}\ 1}$ 25

2x : $\boxed{1\ \ 1\ \ 0\ 0\ 1\ \ 0}$ 50 Left shift O(1)

X : $\boxed{0\ 1\ \ 1\ \ \overline{0\ \ \ 0}\ 1}$ (25)10

$\frac{X}{2}$: $\boxed{0\ 0\ 1\ \ 1\ \ 0\text{-}0}$ (12)10 => Right shift O(1)

```
            ƶ = 0 ;
            while (x > 0)
            {
              if (x. | . 2 = = 1)}
                {
n=                  ƶ= z + y}
log₂x       X = ⌊x/2⌋
            Y = 2Y
            return (ƶ) ;
          }
```

X = 25

$[\log_2 25] = 5$

5 bits required to represent x in binary

$X = 25 = (11001)_2$

X (11001) – then odd

(0 1 1 0 0) – even

(0 0 1 1 0)

5 right (~~0 0~~ 0 1 0)

shift (~~0 0 0~~ 0 1)

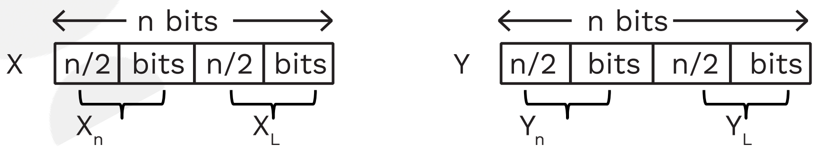
(~~0 0 0 0~~ 0)

Time Complexity two n bit number multiplication : $O(n^2)$

Best Case : O(n)

DAC Integer Multiplication:-

X and Y two n bit integers



$\left\lceil \frac{n}{2} \right\rceil$ bits
seal value

$X = X_n * 2^{\lceil n/2 \rceil} + Xe$

$Y = Y_n * 2^{\lceil n/2 \rceil} + Ye$

←n/2→

Eg. X (1011 | 0001) $2^7 + 2^5 + 2^4 + 2^0 = (177)_{10}$
$(11)_{10}$ (1)

$X = 11 * 2^4 + 1$

$X .Y = (X_n. 2^{n/2} + X_L) (Y_n * 2^{n/2} + Y_L)$

$X .Y = X_n * Y_n. 2^n + (X_n * Y_L + X_L * Y_n) 2^{n/2} + X_L * Y_L$

Cost of Multiplication of X, Y with n bits each equal to

- Cost of 4 sub integer multiplication of n/2 bits
- $\frac{3n}{2}$ left shift operation & 3 integer addition

```
Algorithm Integer Multiplication (X, Y, N)
  {
    if ( n = = 1)
      return (x.y)
    else
      }
      || Divide x, y divide into two sub integer each ( $X_n$, $X_L$, $Y_n$, $Y_L$)
  P = Integer Multiplication ($X_n$, $Y_n$, n/2 ) ;
  q = Integer  Multiplication   ($X_n$, $Y_L$, n/2);          4T(n/2)
  r = Integer Multiplication ($X_L$, $Y_n$, n/2 );
  s = Integer  Multiplication   (ϰ 1 , $Y_L$, n/2  ) ;
      || Conquer
      return ( P * $2^n$ + (q + r) $2^{n/2}$ + 5 )  cn
      }
}
```

$$T(n) = \begin{cases} 4T(n/2) + cn & ; n > 1 \\ 1 & , n \leq 1 \end{cases}$$

[DAC integer multiplication recurrence Relation]

Time Complexity : θ (n^2)

3] Strassen's DAC Algorithm:-

← n bits → ← n bits →

X $\boxed{\leftarrow n/2 \rightarrow \;|\leftarrow n/2 \rightarrow}$ Y $\boxed{\leftarrow n/2 \rightarrow | \leftarrow n/2 \rightarrow}$

X_n X_L Y_n Y_L

$\boxed{X = X_n .\ 2^{n/2} + X_L}$ $\boxed{Y = Y_n .\ 2^{n/2} + Y_L}$

$P = X_n * Y_n$

$q = X_L * Y_L$

$r = (X_n + X_L) * (Y_n + Y_L)$

$$\boxed{X.\ Y = P.\ 2^n + (r - p - q)\ 2^{n/2} + q}$$

To Multiply X, Y with n bits each.

Tn: 3 sub Integer multiplication of n/2 bits each = 3T (n/2)

$$\left.\begin{array}{l}\text{6 Integer addition / sub \&\&} \\ \frac{3n}{2} \text{ left shift operation}\end{array}\right\} c.n$$

$$T(n) = \begin{cases} 3T(n/2) + cn, & \text{for } n>1 \\ 1 & \text{for } n<=1 \end{cases}$$

Strassen's Integer Multiplication Recurrence Relation

Time Complexity = $\theta\ (n^{\log_2 3}) = \theta(n^{1.67})$

Strassen's Matrix Multiplication:-

I) Straight Matrix Multiplication Time Complexity : $\theta(n^3)$

II) D and C Matrix Multiplication Time Complexity : $\theta(n^3)$

III) Stressen's D and C Matrix Multiplication Time Complexity = $\theta\ (n^{2.81})$

1) Matrix Addition

$A_n x\ n + B_n x\ n = C_n x\ n$

$$\begin{array}{cc} & j \\ i \downarrow & \begin{pmatrix} a_{11}\dots\dots\dots a_1 n \\ \vdots \qquad\quad \vdots \\ an1\dots\dots\dots ann \end{pmatrix} \end{array} + \begin{pmatrix} b_{11}\ \dots\dots\dots b_1 n \\ \vdots \qquad\quad \vdots \\ bn1\dots\dots\dots bnn \end{pmatrix} =$$

```
    for ( i = 1 ;  i ≤n; i++)
  ┌{
  |       for ( j = 1 ; i ≤ n ;j++)
  |     ┌{
n |   n |  cij = aij + bij;
  |     └}
  └}
```

Time Complexity = $\theta(n^2)$

Straight Matrix Multiplication

Ap x q * Bq x r = Cp x r

$$\begin{pmatrix} a_{11}a_{12}..\ a_{1q} \\ a_{21} \\ ap1.....apq \end{pmatrix} * \begin{pmatrix} b_{11}b_{12}\dots\ bqr \\ \\ bq1 \qquad bqr \end{pmatrix} = \begin{array}{c} j \\ \begin{pmatrix} c_{11}\ .....\ .c,\ p \\ \\ cp1.....\ cpr \end{pmatrix} \end{array}$$

$C_{11} = a_{11} * b_{11} + a_{12} * b_{21} + ... + a_{1q} * b_{q1}$

Each Element of c [ ] [ ] required "q" element multiplication.

Total cost of Matrix Multiplication: Pxrxq element multiplication

Algorithm straight Matrix Multiplication (A[p] [q] , B [q] [r])

```
{
        for ( i = 1 ; i ≤ p ; i++)
        {
              for (j = 1 ; j ≤ r ;j++)
              {
 p                   c[i] [j] =0
                     for ( k = 1; k ≤ q ; k++)
      q              {
                         c [i] [j] = c [i] [j] + a[i] [k] * b [k] [j] ;
                   r }
              }
        }
  }
```

Time Complexity of Matrix Multiplication θ (P * r * q)

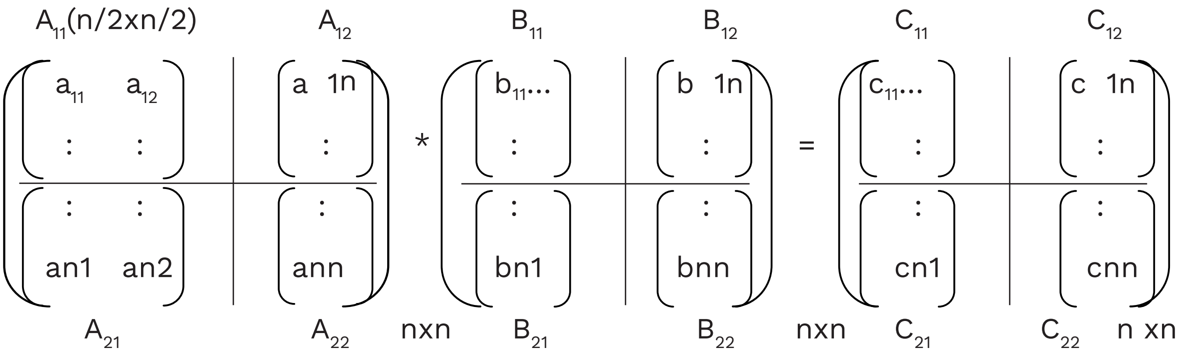
Anxn   Bnxn

Time Complexity of Matrix Multiplication $\theta(n^3)$

2. Divide and Conquer Matrix Multiplication

Anxn,   Bnxn

if n > 2 divide each matrix A, B with order nxn into 4 sub matrices each with n/2 x /n2



$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} * \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$C_{11} = A_{11} B_{11} + A_{12} * B_{21}$

$C_{12} = A_{11} B_{12} + A_{12} * B_{22}$

$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$

$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$

```
Algorithm D and C MM (A,B, n)
  {
    if ( $n \le 2$ )
        return (A * B)
      else
        {
           Divide A & B with order n x n into 4 sub matrices each with n/2 x n/2
p = D and C ($A_{11}$, $B_{11}$, n/2) ;
q = D and C  ($A_{12}$* $B_{21}$, n/2) ;
r = D and C ($A_{11}$* $B_{12}$, n/2);
s = D and C ($A_{12}$, $B_{22}$,n/2);                8T(n/2)
t = D and C ($A_{21}$, $B_{11}$, n/2);
u = D and C ($A_{22}$, $B_{21}$, n/2) ;
v = D and C ($A_{21}$, $B_{12}$, n/2) ;
w = D and C ($A_{22}$, $B_{22}$, n/2);

|| Conquer

$C_{11}$= P + q ;
$C_{12}$=  r +s;          4. $n^2 = \theta(n^2)$
$C_{21}$= t + u;                  ↑
$C_{22}$= v +w ;          Matrix Addition

return  $\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22}; \end{pmatrix}$
```

```
        }
    }
```

D and C Matrix Multiplication Recurrence Relation

$$T(n) = \begin{cases} 8T(n/2) + 4n^2, & \text{for } n>2 \\ a, & \text{for } n<=2 \end{cases}$$

$$= \theta(n^3)$$

Strassen's D and C Matrix Multiplication:-

A & B with orders n x n divided into 4 sub matrices each.

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right) = \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right)$$

$P_1 = A_{11} (B_{12}-B_{22})$

$P_2 = (A_{11} + A_{12}) B_{22}$

$P_3 = (A_{21} + A_{22}) B_{11}$

$P_4 = A_{22} (B_{21} + B_{11})$

$P_5 = (A_{11} + A_{22}) (B_{11} + B_{22})$

$P_6 = (A_{12} - A_{22}) (B_{21} + B_{22})$

$P_7 = (A_{11} - A_{21}) (B_{11} + B_{12})$

$P_5 + P_4 - P_2 + P_6 = C_{11}$

$P_1 + P_2 = C_{12}$

$P_3 + P_4 = C_{21}$

$P_5 + P_1 - P_3 - P_7 = C_{22}$

T(n) : Time complexity of multiply A & B matrices of order n x n equal to

$$\begin{cases} \text{7 sub matrix multiplication of order } n/2 \times n/2 \\ \text{18 matrix Add / sub} \end{cases}$$

$$T(n) = \begin{cases} 7T\ (n/2)\ +18\ n^2 & \text{for } n>2 \\ a & \text{for } n<=2 \end{cases}$$

Time Complexity : $\theta\ (n^{2.81})$

Matrix Multiplication	**Time Complexity**	**Stack Space**
Straight	$\theta(n^3)$	$\theta(1)$
D and C	$\theta(n^3)$	$\theta(\log_2 n)$
Stressen's D and C	$\theta(n^{2.87})$	$\theta(\log_2 n)$

Maximum Minimum Algorithm:-

a[1....n] array of n elements find maximum minimum element of array.

Straight Maximum Minimum Algorithm:-

	1	2	3	4	5	6	7
a	5	6	2	4	9	8	6

Maximum = ~~5~~ ~~6~~ 9   Minimum = ~~5~~2

```
Algorithm maximum- minimum (a,n)
  {
      maximum = minimum = a[i] ;
    for (i = 2 ; i ≤ n ; i++ )
        {
          if (a[i] > maximum)
(n-1)          maximum = a[i];
          else if (a[i] < minimum)
               minimum = a[i] ;
        }
   }
```

Minimum element comparisons:

(n-1) comparisons to find maximum & minimum (when elements are in increasing order)

1	2	3	4	5
5	10	15	20	25

Maximum = 25

Minimum = 5

Maximum element comparisons:-

2 (n-1) = 2 n – 2 comparison to find maximum & minimum (when elements are in decreasing order)

1	2	3	4	5
30	25	20	15	10

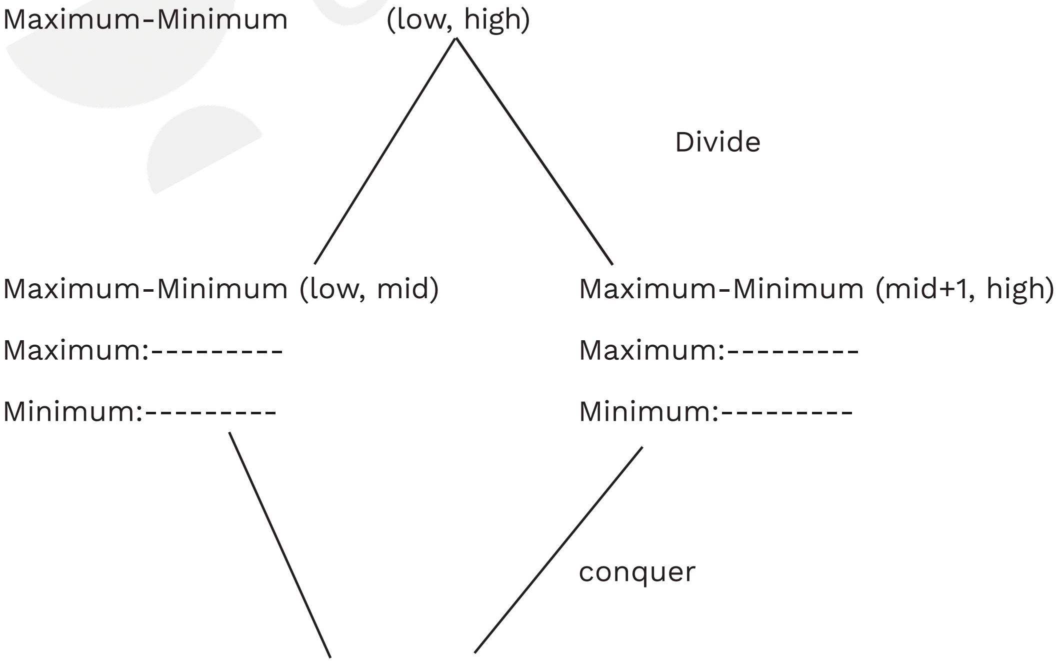
Aug element comparison:

$$= \frac{(n-1)}{2} * 1 + \frac{(n-1)}{2} * 2$$

$= \boxed{\frac{3n}{2} - 1.5}$ comparison to find maximum & minimum

D and C Maximum Minimum Algorithm:-

a [low, high] are given array of n elements



Maximum-Minimum (low, high)

Divide

Maximum-Minimum (low, mid)

Maximum:---------

Minimum:---------

Maximum-Minimum (mid+1, high)

Maximum:---------

Minimum:---------

conquer

if (maximum1 > maximum) then maximum = maximum 1

if (minimum1 < minimum) then minimum = minimum 1

```
Algorithm D and C maximum minimum (L, n, maximum , minimum)
          {
0               if (L = = n) || one element
                maximum = minimum = a [i];
          else if (L = = n – 1 ) || twoelements
                {
1                   if ( a[i] ≤ a[n])
                    { maximum = a[n] ; minimum = a[i] }
          else
                   { maximum = a[i] ; minimum = a[n]
           }
          else
           {

                mid = $\frac{(L + n)}{2}$
                D and C maximum minimum (L, mid, maximum, minimum)   →T(n/2)
                D and C maximum-minimum (mid+1, high, maximum1, minimum1) →T(n/2)
                ||conquer
2T(n/2)+2       if (maximum1 > maximum ) { maximum = maximum 1 ;}
                                                                      2
                if (minimum < minimum ) { minimum = minimum 10 ;}
                { return (maximum , minimum) ;
              }
```

T(n) = number of element comparisons to find maximum, minimum element.

$$T(n) = \begin{cases} 0, & n = 1 \\ 1, & n = 2 \\ 2T(n/2)+2, & n > 2 \end{cases}$$

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

$$\left[\ T(n/2) = 2T(n/2^2) + 2\ \right]$$

$$= \left(2T\left[\frac{n}{2^2}\right] + 2\right) + 2$$

$$= 2^2\ T\left[\frac{n}{2^2}\right] + 2^2 + 2 \qquad \left(T\left[\frac{n}{2^2}\right] = 2T\left[\frac{n}{2^3}\right] + 2\right)$$

$$= 2^2\left(2T\left[\frac{n}{2^3}\right] + 2\right) + 2^2 + 2$$

$$= 2^3T\left[\frac{n}{2^3}\right] + 2^3 + 2^2 + 2$$

$\vdots$ K times

$$= 2^KT\ \frac{n}{2^k} + 2^k + 2^{k-1} + ...+ 2^3 + 2^2 + 2^1$$

$$= 0 + 2*\left[2^K - 1\right]$$

$$\frac{n}{2^k} = 1$$

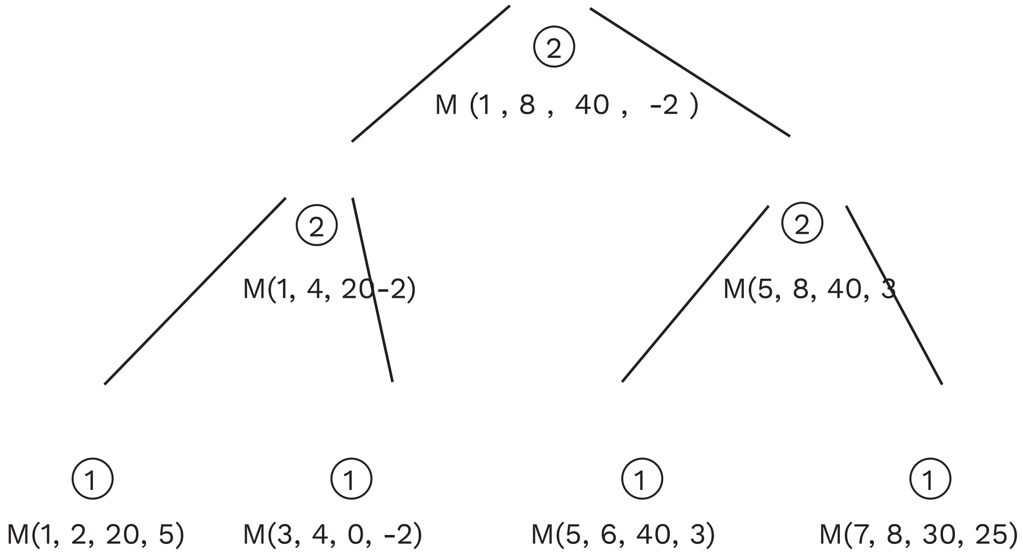
$$n = 2^k \qquad = 2\left[\ 2^{\log_2 n} - 1\ \right]$$

$$k = \log_2 n \qquad = 2\left[\ n - 1\ \right]$$

$$= \text{Assume}\left(\begin{array}{ll} \frac{n}{2^k} = 2 & n = 2^{k+1},\ \ 2^k = n/2 \\ & \log_2 n - 1 = k \end{array}\right)$$

$$= \frac{n}{2}.\ 1 + 2\left(\frac{n}{2} - 1\right) \quad = \frac{3n}{2} - 2$$

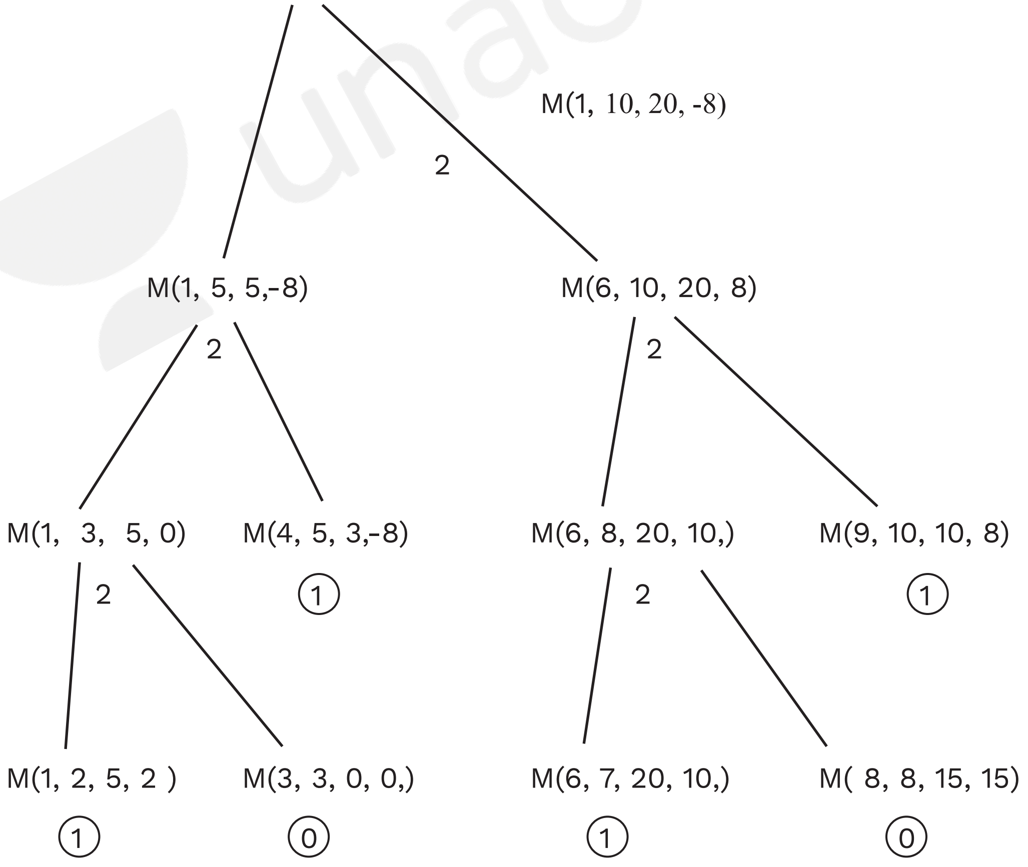
	1	2	3	4	5	6	7	8
a	20	5	0	-2	40	3	25	30



Number of element comparison = $\frac{3n}{2} - 2$ = 10 comparison

Q.

	1	2	3	4	5	6	7	8	9	10
a	5	2	0	3	-8	20	10	15	8	10



Number of comparison = 13.

Maximum Minimum Algorithm

Max Min Algo	Min Comp	Aug Comp	Max Comp	TC
Straight	n – 1	$\frac{3n}{2}$ -1.5	$\underline{2n}$ – 2	=>θ(n)
D and C	$\frac{3n}{2}$ – 2	$\frac{3n}{2}$ – 2	$\frac{3n}{2}$ – 2	=>θ(n)

Q. How many minimum comparison required find maximum & minimum element in array of n element

a) n – 1

b) 2n – 2

c) $\frac{3n}{2} - 2$

d) None

Q. How many maximum comparison required to find maximum minimum element in array of n elements.

a) n – 1

b) n – 2

c) $\frac{3n}{2} - 2$

d) None

Q. What is time complexity required to return element from array of n distinct elements which is neither maximum non minimum elements

a) θ(n)

b) θ(nlogn)

c) $\theta(n^2)$

d) θ(1)

	1	2	3 n
a			4...........

a[1] < a[2] < a[3]

(Elements must be distinct)

Q. What is the time comlexity to return element from array of n distinct elements which is neither 2^{nd} maximum nor 2^{nd} minimum element then 5 elements constant time sorting then comparison then θ (1)

Practice Problems:-

Q.1 Consider a list of recursive algorithms and a list of recurrence relations as shown below. Each recurrence relation corresponds to exactly one algorithm and is used to derive the time complexity of the algorithm.

List-I (Recursive Algorithm)

P. Binary search

Q. Merge sort

R. Quick sort

S. Tower of Hanoi

List-II (Recurrence Relation)

I. $T(n) = T(n-k) + T(k) + cn$

II. $T(n) = 2T(n-1) + 1$

III. $T(n) = 2T(n/2) + cn$

IV. $T(n) = T(n/2) + 1$

Which of the following is the correct match between the algorithms and their recurrence relations?

Codes:

	P	Q	R	S
(a)	II	III	IV	I
(b)	IV	III	I	II
(c)	III	II	IV	I
(d)	IV	II	I	III

[GATE-2004]

Q.2 You are given infinite array A in which the first n-cells contains integers in sorted order and the rest of the cells are filled with ∞. If you are not given the value of n, find time complexity of an algorithm that takes an integer X as input and find the position of element X in the given array A.

a) O(n)

b) O(log n)

c) O(n^2)

d) None of these

Linked Anser Q.3 and Q.4

Consider the following C program that attempts to locate an element x in an array Y[ ] using binary search.

The program is erroneous.

```
1. f(int Y[10], int x)
2. int i, j, k;
3. i = 0; j = 9;
4. do {
5. k = (i + j) | 2;
6. if (Y[K] < x) i = k; else j = k;
7. } while ((Y[k] ! = x) & (i < j));
8. if (Y[k] = = x)
     printf("x is in the arrary");
9. else printf("x is not in the array");
10. }
```

Q.3 On which of the following contents of Y and x does the program fail?

a) Y is [1 2 3 4 5 6 7 8 9 10] and x <10

b) Y is [1 3 5 7 9 11 13 15 17 19] and x < 1

c) Y is [2 2 2 2 2 2 2 2 2 2] and x > 2

d) Y is [2 4 6 8 10 12 14 16 18 20] and 2 < x < 20 and x is even

[GATE-2008]

Q.4 The correction needed in the program to make it work properly is

a) Change line 6 to : if (Y[k] < x) i = k + 1; else j = k – 1;

b) Change line 6 to : if (Y[k] < x) i = k -1; else j = k + 1;

c) Change line 6 to : if (Y[k] < x) i = k, else j = k;

d) Change line 7 to : while ((Y[k] = = x) & (i <j));

[GATE-2008]

Q.5 Given a sorted array of n-elements where other than one element x everyother element repeat two times. Then how much time will it take to find position of x.

Q.6 An array A[1...n] contains all the integers from 0 to n, except one element. How much time it will take to determine the missing integer?

a) O(n)

b) O(log n)

c) $O(n^2)$

d) None of these

Q.7 For merging two sorted lists of sizes m and n into a sorted list of size m + n, we required comparisons of

a) O(m)

b) O(n)

c) O(m + n)

d) O(log m + log n)

[GATE-1995]

Common Data for Q.8 & Q.9

In a permutation a_1... an of n distinct integers, an inversion is a pair (a_i, a_j) such that i < j and a_i> a_j

Q.8 If all permutations are equally likely, what is the expected number of inversions in a randomly chosen permutation of 1...n?

a) n(n-1)/2

b) n(n-1)/4

c) n(n + 1)/4

d) $2n[\log_2 n]$

[GATE-2003]

Q9 What would be the worst case time complexity of the insertion sort algorithm, if the inputs are restricted to permutations of 1...n with at most n inversions?

a) $\theta(n^2)$

b) $\theta(n \log n)$

c) $\theta(n^{1.5})$

d) $\theta(n)$

[GATE-2003]

Q.10 Let F_k denote the kth Fibonacci number with $F_k = F_{k-1} + F_{k-2}$ for $k \geq 2$, $F_0 = F_1 = 1$. Consider the following variation of an merge sort, which assumes that the number of elements in its list argument L is a Fibonacci number $F_{k.}$

```
{
Algorithm FibMergeSort (L)
L is a list of items from a totally ordered set, whose length is a Fibonacci number $F_k$
If L contains only 1 element, then return L;
Else {
Divide L into $L_1$(the first $F_{k-1}$ items) and $L_2$(the remaining $F_{k-2}$ items)
Sorted $L_1$:= FibmergeSorte$L_1$)
Sorted $L_2$:= FibmergeSorte$L_2$)
Sorted L: = Merge(sorted$L_1$, sorted$L_2$)
Return sorted; } }
```

Assuming that the “divide” step in FibMergeSort takes constant time no comparisons) and Merge behaves similar to the merge in the normal merge sort. Identify which of the following expressions most closely matches the total number of comparisons performed by FibmergeSort when initially given a liat of F_k elements?

a) O(k log k)

b) O (K^2)

c) O(K F_k)

d) O($F_k$log K)

Q.11 If one uses straight two-way merge sort algorithm to sort the following elementsin ascending order:

20, 47, 15, 8, 8, 9, 4, 40, 30, 12, 17

Then the order of these elements after second pass of the algorithm is

a) 8, 9, 15, 20, 47, 4, 12, 17, 30, 40

b) 8, 15, 20, 47, 4, 9, 30, 40, 12, 17

c) 15, 20, 47, 4, 8, 9, 12, 30, 40, 17

d) 4, 8, 9, 15, 20, 47, 12, 17, 30, 40

[GATE-1999]

Q.12 We are given a sequence of n-numbers a_1, a_2, a_3... a_n, we will assume that all the number are distinct. We say two indices i < j form an inversion if a_i a_j,

How much time it will take to find total number of inversions in the given array?

a) $O(n^2)$

b) O(n log n)

c) O(n)

d) None of these

Q.13 Suppose you have k-sorted arrays, each with n-elements and you want to combine those k-sorted arrays into a single sorted array of kn elements. How much time it will take?

a) O(kn)

b) O(kn log k)

c) $O(k^2)$

d) None of these

Q.14 Suppose there are 4 sorted lists of n/4 elements each. If we merge these lists into a single sorted list of n elements, for the n = 400 number of key comparisons in the worst case using an efficient algorithm is_____

Q.15 Assume that a mergesort algorithm in the worst case takes 30 seconds for an input of size 64. Which of the following most closely approximates the maximum input size of a problem that can be solved in 6 minutes?

Q.16 Which of the following is divide and conquer application?

a) Heap sort

b) Insertion sort

c) Bubble sort

d) Merge sort

[DRDO-2008]

Q.17 Assume an array A [1, ..., n] has n-elements, and every element of an array is positive and less than or equal to n. An element is said to be "majority element", if it is occurred in more than $\frac{n}{2}$ positions of an array. What is the time complexity to check

whether the majority element exist or not in the given array? [Best answer]

a) O(log n)

b) O(n)

c) O(n log n)

d) O(n^2)

Q.18 Binary search can e carried out on a set of ordered data items stores in a

(a) Array

(b) Stock

(c) Queue

(d) List

[DRDO-2008]

Q.19 Given 2-sorted arrays each of n-elements and distinct. How much time it will take to find middle element of the union sorted array?

a) O (1)

b) O(log n)

c) O(n)

d) None of these

Q.20 Which of the following statements is true?

I. As the number of entries in a hash table increases, the number of collisions increases.

II. Recursive programs are efficient

III. The worst case complexity for Quick sort is O(n^2)

IV. Binary search using a linear linked list is efficient.

(a) I and II

(b) II and III

(c) I and IV

(d) I and III

Q.21 Let s be a sorted array of n integers. Let t(n) denote the time taken for the most efficient algorithm to determined if there are two elements with sum less than 1000 in s. which of the following statements is true?

a) t(n) is 0(1)

b) $n \leq t(n) \leq n \log_2 n$

c) $n \log_2 n \leq t(n) < \frac{n}{2}$

d) $t(n) = \frac{n}{2}$

[GATE-2000]

Q.22 The cube root of a natural number n is defined as the larger natural number m such that $m^3 \leq n$. The complexity of computing the cube root of n (n is represented in binary notation) is

a) $O(n)$ but not $O(n^{0.5})$

b) $O(n^{0.5})$ but not $O((\log n)^k)$ for any constant $k > 0$

c) $O((\log n)^k)$ for some constant k $k > 0$, but not $O((\log\log n)^m)$ for any constant $m > 0$

d) $O((\log\log n)^k)$ for some constant $k > 0.5$, but not $O((\log\log n)^{0.5})$

[GATE-2003]

Q.23 Merging k-sorted lists each of size n|k into one sorted list of n-elements using heap sort will take how much time?

Ans: O(n log k)

Q.24 Consider a modification to merge sort in which m|k sublists each of length k are sorted using insertion sort and then merged using standard merge procedure. Then find total time complexity of modified merge sort.

Ans: mk + m log (m|k)

Q.25 Given array of n-distinct elements. What is the worst case running time to find ith smallest element ($1 \leq i \leq n$) from those n elements? (Select the best answer by assuming n is larger than 500)

a) O(logn)

b) O(n)

c) O(n|logn)

d) $O(n^2)$

Ans: (b)

Q.26 The worst case time complexity of Quick sort for n elements when the median is selected as the pivot is;

a) O(n)

b) O(n^2)

c) O(n log n)

d) O(n log n)

[DRDO-2008]

Q.27 Randomized quick sort is an extension of quick sort where the pivot is chosen randomly. What is the worst case complexity of sorting n numbers using randomized quick sort?

a) O(n)

b) O(n log n)

c) O(n^2)

d) O(n!)

[GATE-2001]

Q.28 Given two arrays of numbers a_1,...., a_n and b_1,...., b_n where each number is 0 or 1, the fastest algorithm to find the largest span (i, j) or report that there is no such span,

a) Takes O(3^n) and Ω(2^n) time if hashing is permitted

b) Takes O(n^3) and Ω($n^{2.5}$) time in the key comparison model

c) Takes θ(n) time and space

d) Takes O(√n) time only if the sum of the 2n elements is an even number

[GATE-2006]

Q.29 Suppose you are provided with the following function declaration in the C programming language.

int partition (int a[], int n);

The function treats the first element of a [ ] as a pivot, and rearranges the array so that all elements less than or equal to the pivot is in the left part of the array, and all elements greater than the pivot is in the right part. In addition, it moves the pivot so that the pivot is the last element of the left part. The return value is the number of elements in the left part.

The following partially given function in the C programming language is used to find the K^{th} smallest element in an array a[ ] of size n using the partition function. We assume k ≤ n.

```
int Kth_smallest (int a [ ] n, int K) {
int left_end = partition (a, n);
if (left_end + 1 = = k)
return a [left_end];
if (left_end + 1 > k)
return kth_smallest (______);
else
return Kth  _smallest(_____);
```

The missing argument lists are respectively

a) (a, left_end, k) and (a + left_end +1, n-left_end-1, k-left_end-1)

b) (a, left_end, k) and (a, n-left_end-1, k-left_end-1)

c) (a+left_end+1, n-left_end-1, k-left_end-1) and (a, left_end, k)

d) (a, n-left_end-1, k-left_end-1) and (a, left_end, k)

[GATE-2015]

Q.30 Consider an array consisting of the following elements in unsorted order (placed randomly), but 60 as first element.

60, 80, 15, 95, 7, 12, 35, 90, 55

Quick sort partition algorithm is applied by choosing first element as pivot element. How many total number of arrangements of array integers is possible preserving the effect of first pass of partition algorithm?

Q.31 The minimum number of comparisons required to sort 5 elements is

a) 4

b) 5

c) 6

d) 7

[DRDO-2008]

Q.32 An element in an array X is called a leader if it is greater than all elements to the right of it in X. The best algorithm to find all leaders in an array.

a) Solves it in linear time using a left to right pass of the array

b) Solves in linear time using a right to left pass

c) Solves it is using divide and conquer in time Θ (n log n)

d) Solves it in time Θ (n^2)

[GATE-2006]

Q.33 The number of comparisons required to find maximum and minimum in the given array of n-elements using divide and conquer is_____

a) $\left[\frac{3n}{2}\right]$

b) $\left[\frac{3n}{2}\right]$

c) $\left[\frac{3n}{2}\right] + 2$

d) $\left[\frac{3n}{2}\right] - 2$

Q34 An unordered list contains n distinct elements. The number of comparisons to find an element in this list that is neither 2^{nd} maximum nor 2^{nd} minimum is

a) Θ (n log n)

b) Θ (n)

c) Θ (log n)

d) Θ (1)

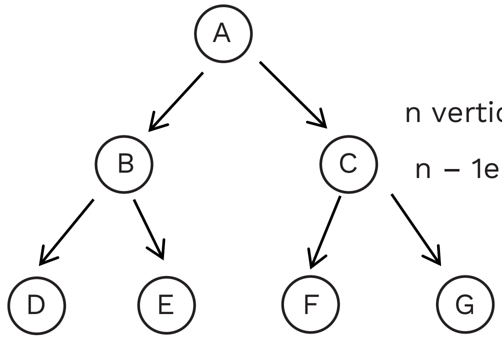
Greedy Techniques

Data Structures used in greedy Algorithm :

1. Tree and graph representation
2. Binary Trees
3. Binary Heaps

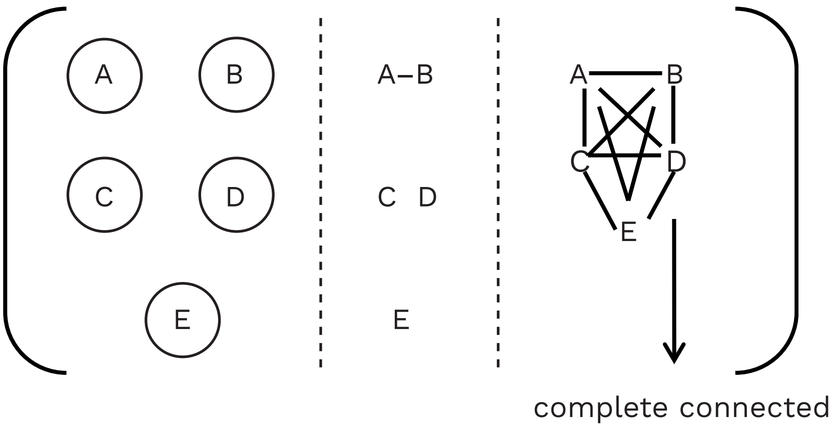
   [Max Heap && Min heap]
4. Set Algorithm of Disjoint sets union & find Algorithm

Tree	Graph
• Single Rooted All tree operations Start from Root node	• Any vertex of graph can use as starting point to perform to graph operation
• Tree always has directed edges && always acyclic	• graph may/may not have directed edges. May/May not be cyclic
• Always connected [for n vertex tree no. of edges(n-1)]	• may/may not connected [for n vertex simple graph n can be o edges to n $\underbrace{\frac{(n-1)}{2}}$ edges null graph complete connected graph

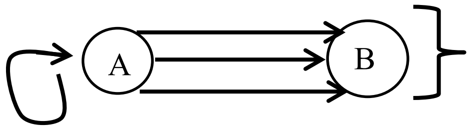


n vertices

n – 1edges



Simple Graph :- Graph with no self loops and no parallel edges

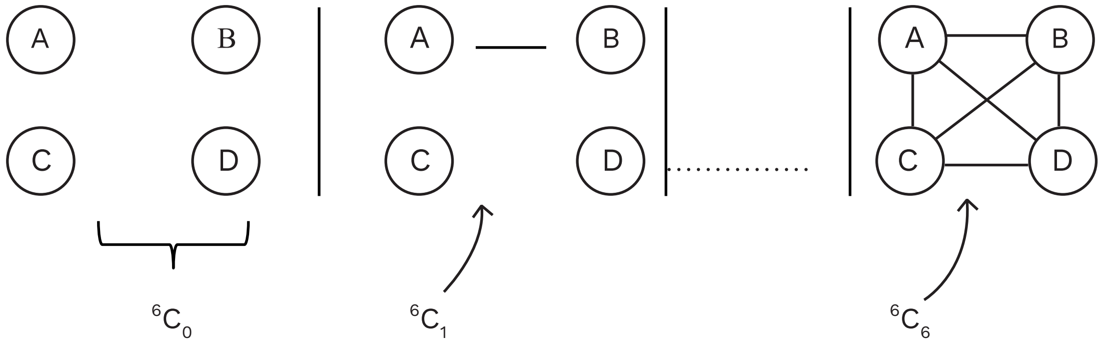


Not simple graph

Q. How many simple undirected graphs possible for given n vertices

Simple graph: $2^{\frac{n(n-1)}{2}}$

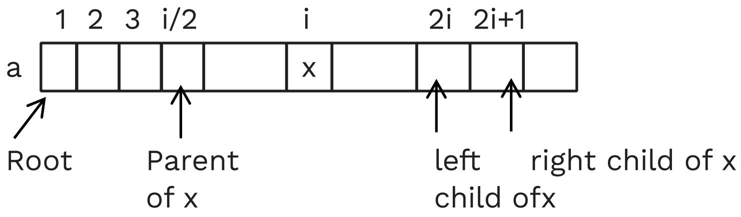
n =4, number of possible edges = $\frac{4 * 3}{2}$

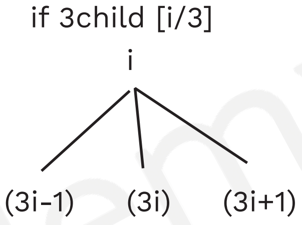


$^6C_0 + {}^6C_1 + {}^6C_2 + {}^6C_3 + {}^6C_4 + {}^6C_5 + {}^6C_6 = 2^6$

Tree Representations:-

1) Array Representation
2) Linked list representation
3) Array Representation of Binary tree :- (not always for Binary)



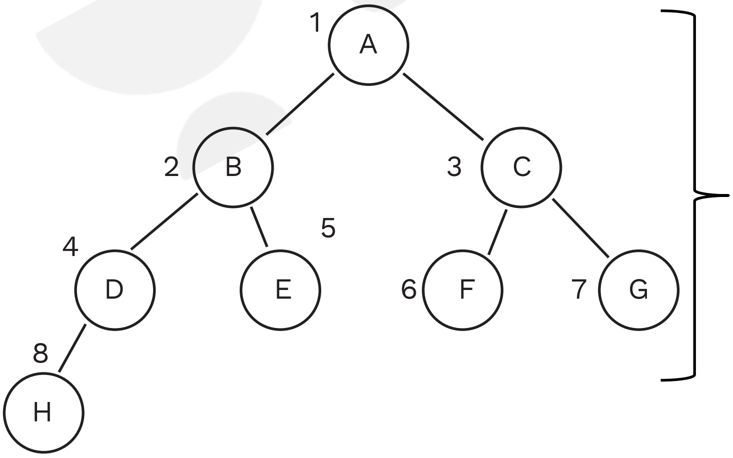


Root index => 1

If Node x at index i

then left child of x at 2.i.index right child of x at (2i+1) index parent of x at (i/2) index.

Example:

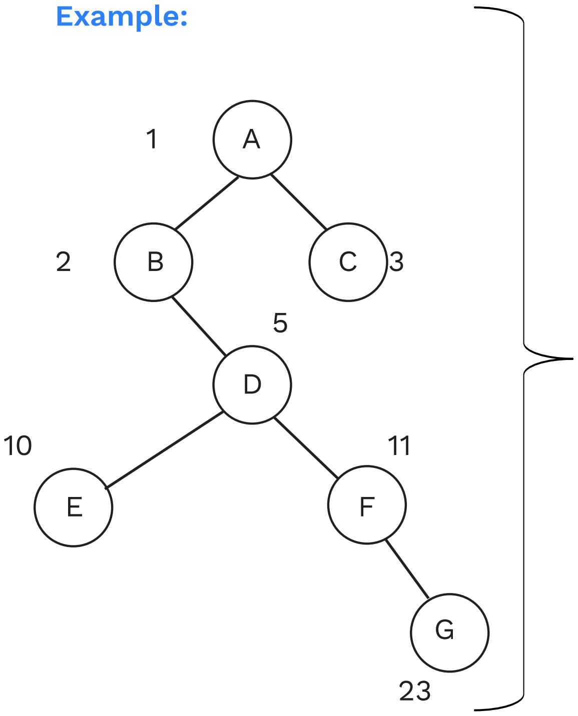


array representation prepared if node indexes of tree is continuous integers.

array

1	2	3	4	5	6	7	8
A	B	C	D	E	F	G	H

Example:

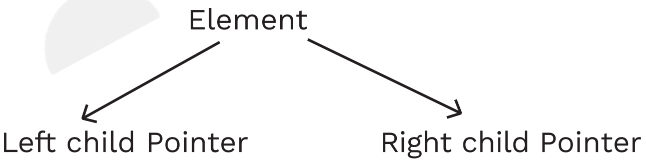


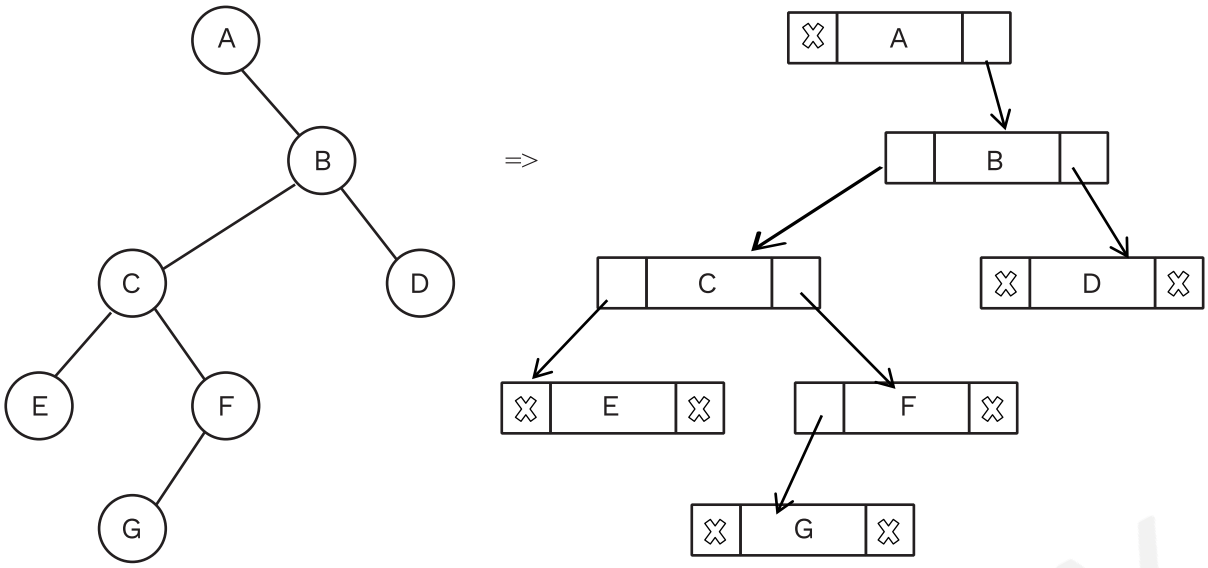
array representation not good if tree node indexes not continuous integers

|| wastage of memory space

	1	2	3	4	5	6..	10	11..........23		
array	A	B	C		D		E	F	....	G

2) Linked list Representation of Binary Tree.





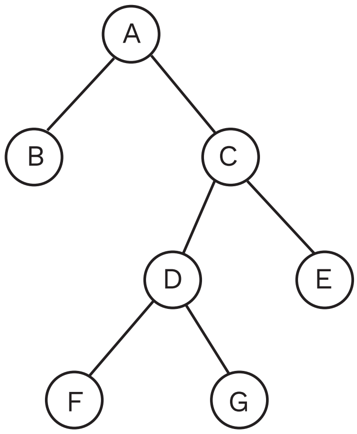
n Vertices Binary Tree 2n pointers

$$\left\{\begin{array}{l}\text{n + 1 null pointers}\\ \text{n - 1 are non null pointers}\end{array}\right\}$$

Binary Tree:-

- Strict Binary Tree
- Complete Binary Tree
- Full Binary Tree
- Strict Binary Tree:- each node must be either o or 2 Childs

Example:



[Linked list representation preferred]

Q.1 Complete K-ary [every node either-o or K child]. How many no. of leaf nodes with n internal nodes,

a) n. k

b) (n-1) K + 1

c) n(K-1) + 1

d) n(K-1)

Q.2 n-ary tree in which every node o or n child if x is number of internal nodes find number of leaf nodes?

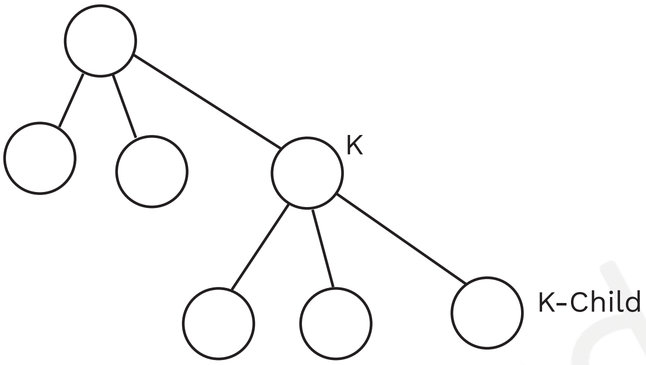
a) X (n -1) + 1

b) X.n – 1

c) X n + 1

d) X (n +1)

1. Given number of internal nodes [n]	find number of leaf nodes
0	1
1	K
2	2K-1
3	3K-2
:	
n	N(K-1) +1



2. X : number of internal nodes	find number of leaf nodes
0	1
1	n

Q.3 Number of leaf nodes in a tree of n nodes with each node having 0 or 3 Childs.

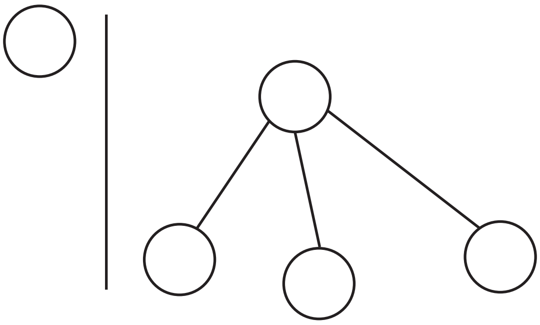
a) [n/2]

b) $\frac{[n - 1]}{2}$

c) $\frac{[n + 1]}{2}$

d) $\frac{[2n + 1]}{3}$

Given n nodes	find number of leaf nodes
1	1
4	3
7	5
10	7

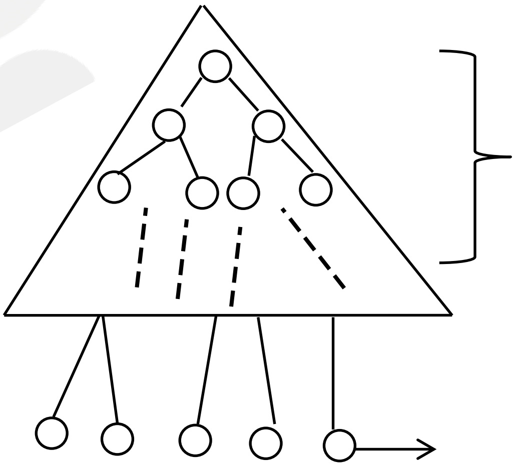


Complete Binary Tree

Complete Binary tree (CBT) of “L” levels is a binary tree in which all 1 to (L-1) levels must be fully occupied

&&

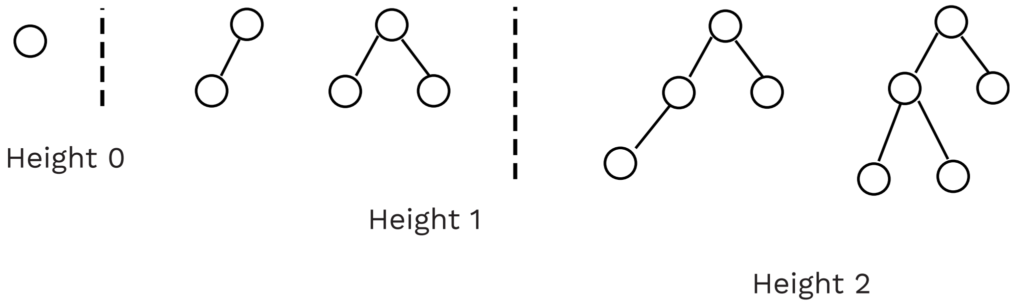
and Lth level nodes must filled from left to Right



(L-1) levels full tree

Nodes can insert left to right

CBT Examples :-



Height	**min number of nodes**	**Max number of nodes**
0	1	L
1	2	3
2	4	7
3	8	15
:	:	:
:	:	:
h	2^h	2^{h+1}-1

1. number of nodes in CBT of height n :

    { 2^n nodes to $2^{n+1}-1$}

2. Height of CBT with n nodes : θ ($\log_2^n$)

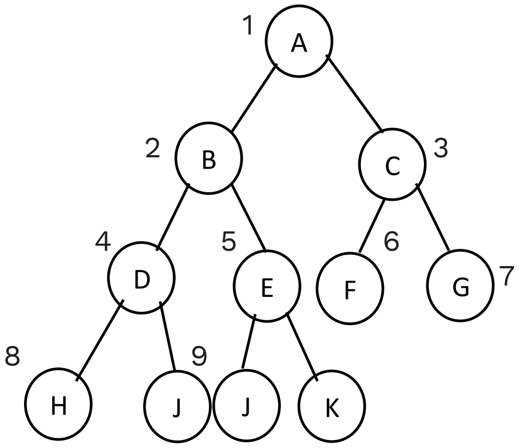
    $n = 2^h \Rightarrow h = \log_2 n = \theta(\log n)$

    $n = 2^{h+1}-1 \Rightarrow h = \log_2(n+1)-1 = \theta(\log_2 n)$

* Node indexes of CBT are continuous Integers

[Array representations preferred to store CBT].

$\searrow$ no overhead of pointers.



1	2	3	4	5	6	7	8	9	10	11
A	B	C	D	E	F	G	H	I	J	K

Internal nodes$\left[\frac{n}{2}\right]$ leaf nodes$\left[\frac{n}{2}\right]$

(only forCBT)

$1 \ldots\ldots\ldots\ldots \left[\frac{n}{2}\right] \qquad \left[\left[\frac{n}{2}\right] + 1\right] \ldots\ldots\ldots\ldots n$

$\longleftarrow$ n nodes $\longrightarrow$

Complete BT

$\left[\frac{n}{2}\right]$Internal nodes of CBT $\left[\frac{n}{2}\right]$are leaf nodes of CBT

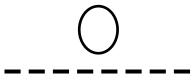
Full Binary Tree (FBT)

FBT is Binary tree with

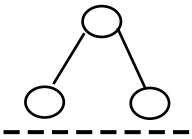
1) Each node 0 or 2 Childs

(and)

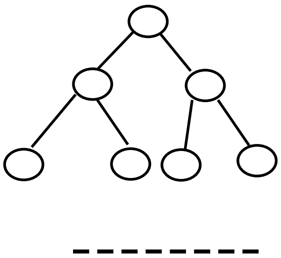
2) All leaf nodes much be at the same level.



Height :0



Height : 1



Height : 2

1. Number of nodes in FBT of height h, n: $2^{n+1}-1$ nodes
2. Height of FBT h, n = $2^{n+1}-1$

   => $h = \log_2(n+1) - 1 = \theta(\log_2 n)$

3. Every FBT is CBT.

Priority Queue Data structure:-

Data Structure which should perform

1. Repeated min/max element deletion

   and

2. Repeated Insertion of elements efficiently. [can consist duplicate elements].

Application:

2	8	4
6	2	4
10	9	7

Max value: high priority

=> delete max repeatedly

List processes => Insertion of element

=> can have duplicate value.

2	8	4
6	2	4
10	9	7

Min value:

- Delete min repeatedly
- Insertion of element
- Can have duplicate element

- Binary heap Data Structure are best Data Structure for priority Queue implementation

1) Max heap

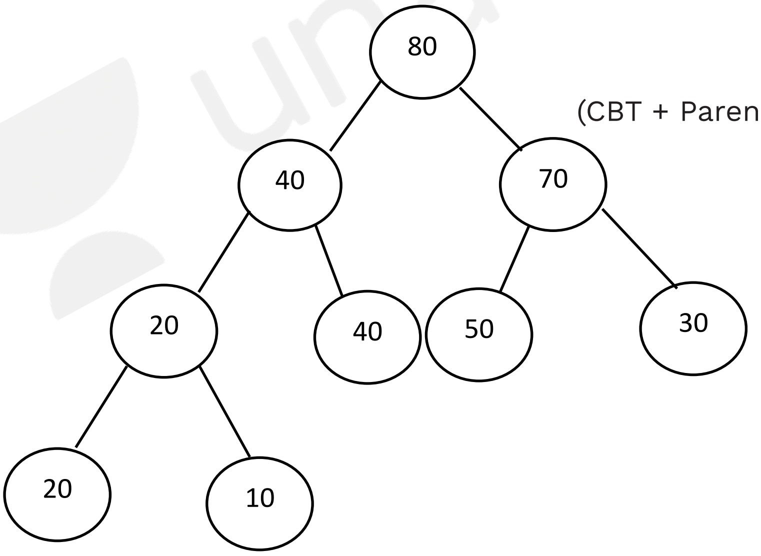
   [Best Data Structure for Repeated max element deletion]

2) Min heap

   [Best Data Structure for repeated min element deletion]

   - Max (Min) heap :-
   - Complete Binary tree with every parent Max (min) than all decedent nodes.

Max heap Example:-

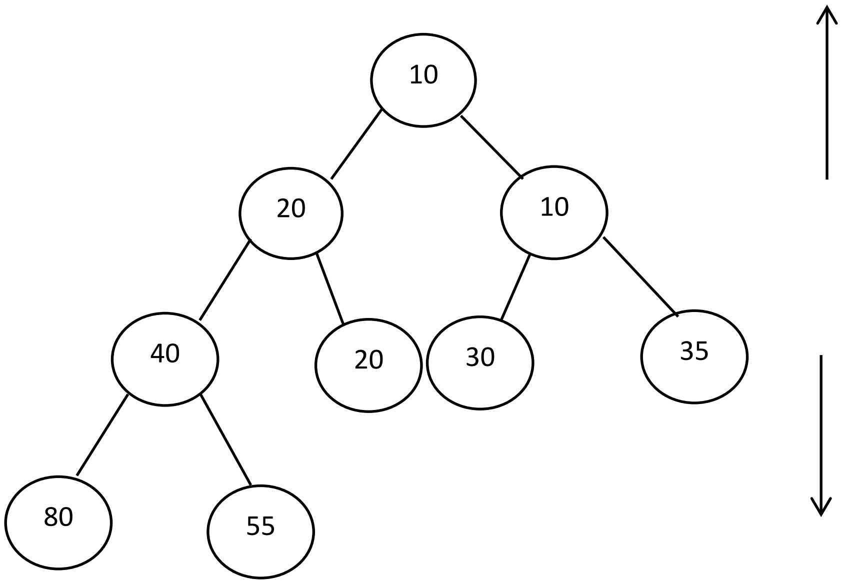


(CBT + Parent Max than decedents)

	1	2	3	4	5	6	7	8	9
a	80	40	70	20	40	50	30	20	10

(Max heap array)

Min heap example:-



CBT + parent min than than decedents

$\log_2 n$ Height

	1	2	3	4	5	6	7	8	9
array	10	20	10	40	20	30	35	80	55

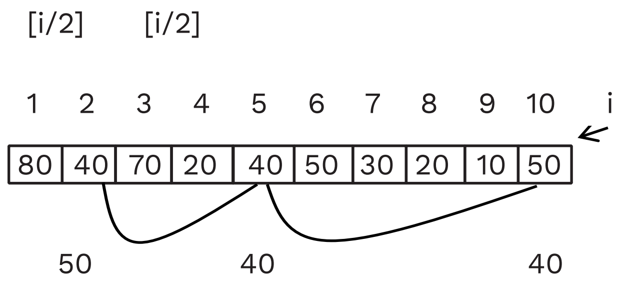
(Min heap array)

Insertion of element into Max heap

```
Algorithm Insert Max heap (a, n, x)
{
   || [ a [1....n-1]  elements of array are already in max heap.
         i = n ; item = x;
     while ( i > 1 && a [i/2] < item)
      {
         a [i] = a [i/2]] ;
            i = i/2;
     }
        a [i] = item ;
  }
```

Time Complexity to inset element into max heap of n element : θ ($\log_2 n$)

Best Case : θ(1) Average Case /Worst Case : θ($\log_2 n$)



1	2	3	4	5	6	7	8	9	10
80	40	70	20	40	50	30	20	10	50

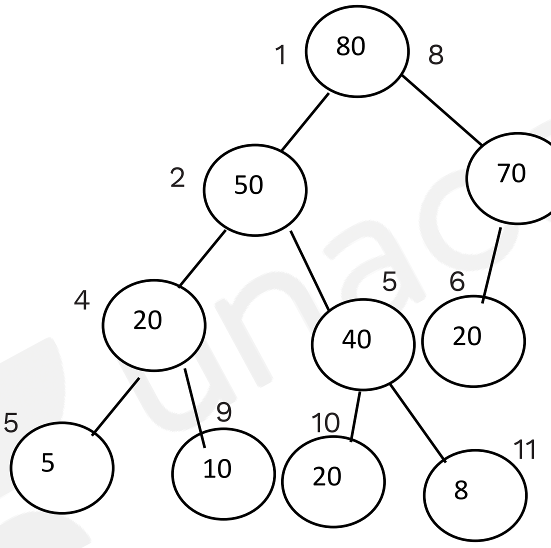
item = 50

i = 10

5

2

Delete Max from Max heap:-



Max element = a [1]

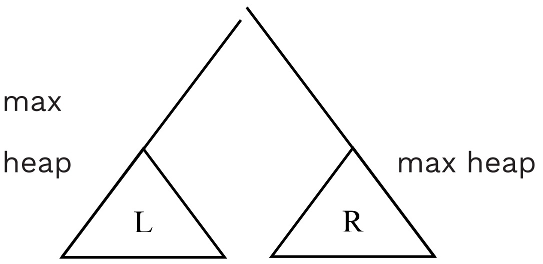
a [1] = a[n] call adjust

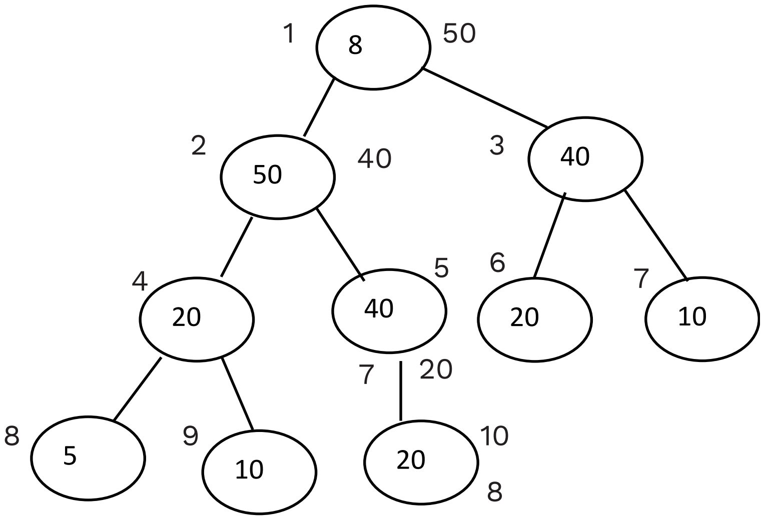
Max heap (a, 1, n-1) ;

Max heap

Adjust

K Not in max heap





item = 8

j = 2

4 5

10

20

a [i/2] = item

Greedy Methods Solves following problems:-

1] Single source shortest path Algorithm

a] Dijkstras Algorithm [Greedy]

b] Bellman fond Algorithm [Not greedy]

2] Minimal Spanning Tree Construction

a) Prim's Algorithm (greedy Algorithm)

b) Kruskal's Algorithm {greedy Algorithm}

3] Huffman Coding [Greedy Algorithm]

[Optimal Prefix Encoding].

4] Fraction knapsack Problem (greedy Algorithm)

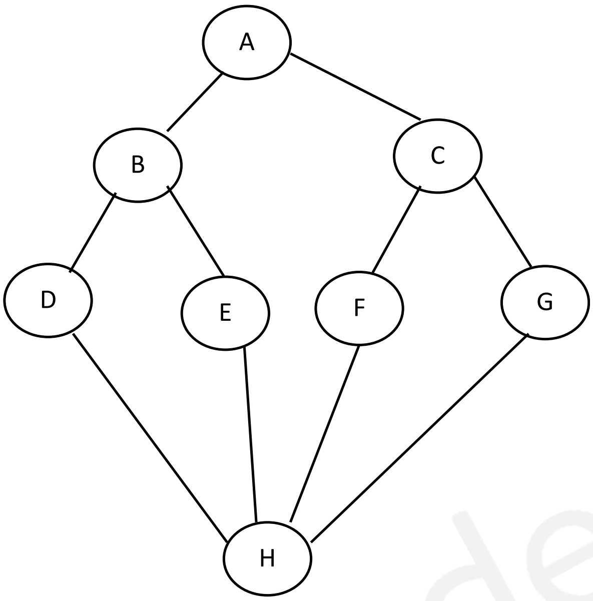
O/1 knapsack problem = (failed by greedy)

5] Job scheduling based on deadline Time (greedy)

6] Topological order (greedy)

Graph Representation

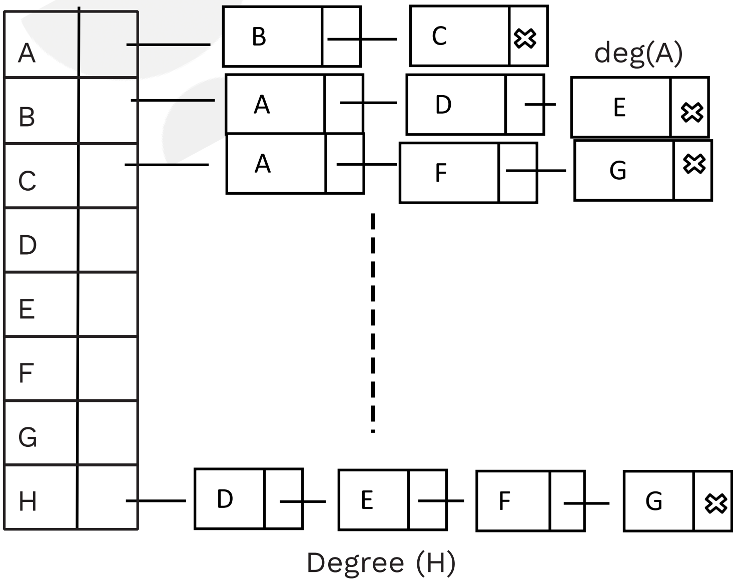
- Adjacency list representation
- Adjacency Matrix representation



Adjacency list Representation

Index Node

Lable Pointer



Adjacency Matrix Representation

Adj[1...n][1...n]

$$\text{Adj }(i, j) = \begin{cases} 1, & (i,j) \text{ is an edge} \\ 0, & (i, j) \text{ is not an edge} \end{cases}$$

- Preferred to store sparse graph

- Time Complexity to find Adj of given vertex (v)

θ (dev(v) :

{

Best case Ω (1)

Worst case O(n)

}

- Time Complexity to find Adj of all vertices

far (i = 1, i ≤ n ; i++)

{

find Adj (V_i) }

Time Complexity =

deg (v_1) + deg (v_2) + ...deg (vn)

= θ (n+ e) $\left[\begin{array}{ll} \Omega(n & \text{to}O(n^2) \\ \text{B.C} & \text{W.C} \end{array}\right.$

$$\sum_{i=1}^{n} \deg (V_i) = 2(e) = \theta(e)$$

$$\begin{array}{c|cccccccc} & A & B & C & D & E & F & G & H \\ \hline A & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ B & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ C & & & & & & & & \\ D & & & & & & & & \\ E & & & & & \ddots & & & \\ F & & & & & & & & \\ G & & & & & & & & \\ H & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{array} \quad n\text{x}n$$

- Time Complexity to find Adj of given vertex O(n) [all cases]

- Time Complexity to find Adj of vertices = θ (n^2) (All cases)

- Space Complexity to store graph of n vertices, edges

O(n^2) (All cases)

- Adj Matrix Representation preferred for dense graph (because no overhead of pointers)

- Space Complexity to store graph of n vertices && e edges in adj list representation

B.C W.C

$n + 2\,|e\,| = \Theta\,(n + e)$: $\Omega\,(n)$ to $O(n^2)$

Dense Graph: || More number of edges graph G with n vertices & e edges where edges

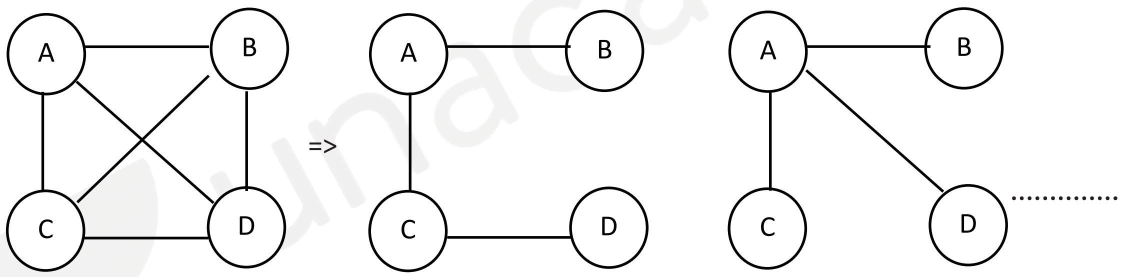
$(e) = O(n^2)$

Example: Complete Connected Graph

Minimal Spanning Tree Construction:-

Spanning Tree G:-

For given graph G (n,e) spanning Tree of given graph is G'(n,e) such that G' has all vetrices of G && G' must be connected && Acyclic

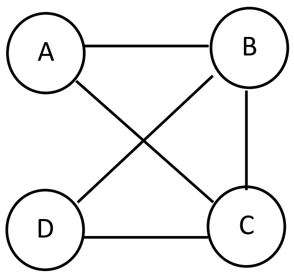


G' : Spanning Tree

- Number of spanning trees of complete connected graph with n vertices are n^{n-2}.

Kirchoff theorem :-

1. Represent graph in Adj Matrix format



$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 1 & 1 & 0 \\ B & 1 & 0 & 1 & 1 \\ C & 1 & 1 & 0 & 1 \\ D & 0 & 1 & 1 & 0 \end{array}$$

2. Multiply (-1) to Adj Matrix

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & -1 & -1 & 0 \\ B & -1 & 0 & -1 & -1 \\ C & -1 & -1 & 0 & -1 \\ D & 0 & -1 & -1 & 0 \end{array}$$

3. Replace diagonal by degree of vertex

Adj (i, i) = degree (v_i)

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 2 & -1 & -1 & 0 \\ B & -1 & 3 & -1 & -1 \\ C & -1 & -1 & 3 & -1 \\ D & 0 & -1 & -1 & 2 \end{array}$$

4. Cofactor of any diagonal element is number of Spanning Tree's of graph

(spanning tree)

$$\text{Cofactor } A_{i,j}=(-1)^{i+j} * \left| \text{det} \ \underset{n-1 \times n-1}{[\ ]} \right|$$

for element 2 = a_{11}

$$(-1)^2 \text{ det} \begin{vmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 2 \end{vmatrix}$$

= 3 (6 -1) + 1 (-2 – 1) – 1 (1 + 3)

= 15 – 3 – 4

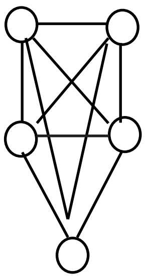
= 8. Spanning tree's.

Q. Given complete connected of n vertices graph. How many edge disjoint spanning tree possible?

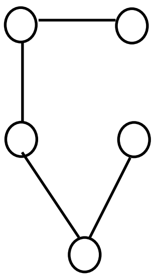
Answer. Number of edges $\frac{n(n-1)}{2}$

= $\frac{n}{2}$ number of edge disjoint Spanning Tree's

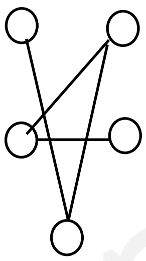
Number of edges =(n-1) in each spanning tree



10 edges



4 edges



4 edges

$\frac{5*4}{2}$= 10 edges

Minimal Spanning Tree Construction

MST: sum of edge costs of spanning tree must be minimum.

Greedy Algorithm used to construct MST

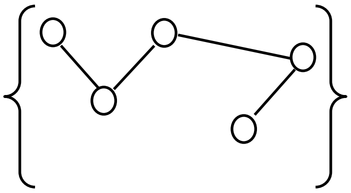
1. Prim's Algorithm
2. Kruskal's Algorithm

Prim's Algorithm

- Construct MST edge by edge
- "X" set of edges included into MST so far,

  All edges of "x" must be connected.

X :



Next including edge u,v into MST such that { x U (u,v)} must be connected && min increment of sum of edge costs.

Implementation

	1	2	3	4	5	6	7	8
dis	∞	∞	∞	∞	∞	∞	∞	∞

	1	2	3	4	5	6	7	8
prev	-1	-1	-1	-1	-1	-1	-1	-1

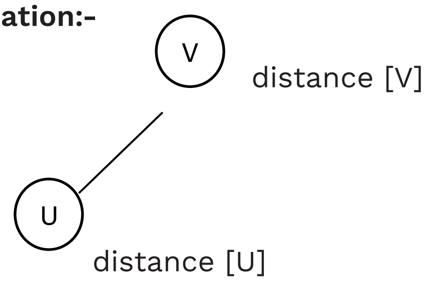
	1	2	3	4	5	6	7	8
Colour	w	w	w	w	w	w	w	w

list				
	1	2	3	4
	5	6	7	8

|| Vertex Include

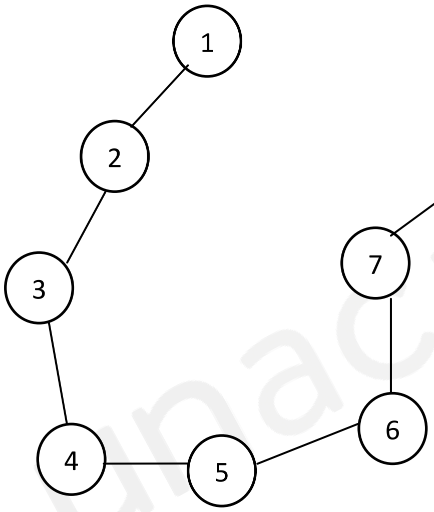
5 4 3 6 7 8 2 1

Relaxation:-



```
if (distance [V] > cost (u, v) && colour [V] = = w)
    {
        distance [V] = cost (u,v)
        prev [V] = u
      }
```

```
1.  Initialization
2.  for each vertex of list(ϕ)
    {U = delete minimum distance vertex from ϕ apply edge relaxation all adjacency of u
}
```

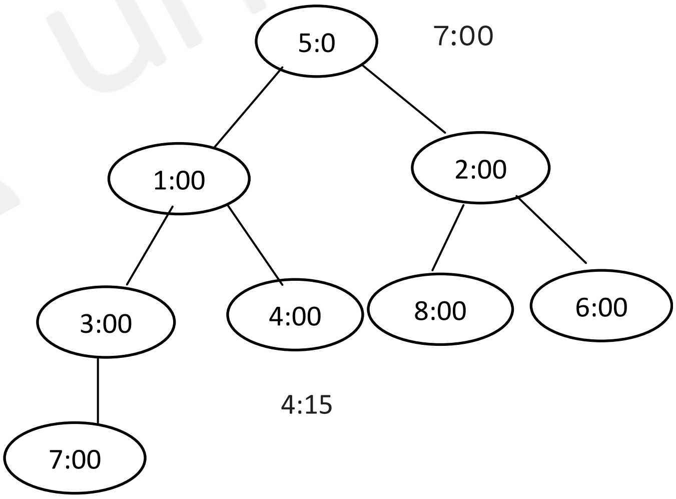


Algorithm prims (graph (G) , n, e , cost [ ] [ ])

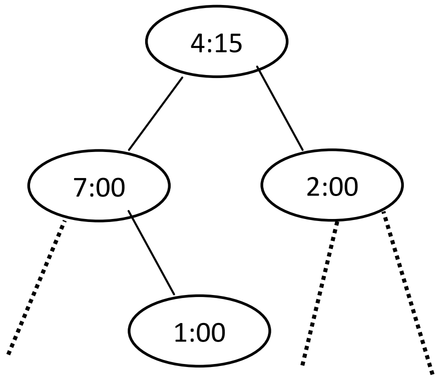
```
    {
        ┌ for (i = 1 ; i ≤ n ; i++)
        │     {
θ(n)    └ distance [Vi] = ∞ ; prev [Vi] = -1;
                      colour [Vi] = w }
          distance [s] = 0 ;
θ(n) [ build minimum heap (ϕ) for all vertices based distance values
          while (ϕ is not empty)
```

```
              {
              U = Delete Minimum distance vertex : logn from (Φ)
              colour [u] = "B" ; || u included into MST

              V= find Adj of vertex (u) : degree (u) or
              for (each vertex of v set) O(n)

              {
                  if (distance [v] > cost (u,v) && colour [v] = = w)
n                 {
times                 distance [v] = cost [u, v] ;
                      prev [v] = u;
        deg(u)        Key decrement (ϕ, v, distance [v] ; θ(logn)
                  }
              }

          }   return (distance [ ], prev [ ] );
}
```

Priority Queue (ϕ)



[Key decrement] θ(logn)



Prim's Algorithm Time Complexity:-

1] Using minimum heap & Adj list representation of graph

$$\underset{\substack{\uparrow\\ \text{initialization}}}{O(n)} + \underset{\substack{\uparrow\\ \text{Build}\\ \text{minimum heap}}}{O(n)} + n \; \underset{\substack{\uparrow\\ \text{delete}\\ \text{minimum}}}{\{\log n\}} + \underset{\substack{\uparrow\\ \text{find Adj(u)}}}{\text{degree }(u)} + \underset{\substack{\uparrow\\ \text{Key decrement}}}{\text{degree }(u).\log n}$$

$O(n) + O(n) + O(n\log n) + 2\,|\,e\,|\, + 2\,|\,e\,|\,.\log n$

$= \theta\,((n + e)\,.\,\log n\,)$

2] Using minimum heap && matrix graph representation:-

$$O(n) + O(n) + n\,\{\log n + \underset{\substack{\uparrow\\ \text{find Adj}}}{O(n)}\} + \text{degree}(u)\,.\,\log n$$

$\theta\,(n^2 + e\log n\,)$

Best Case: when e is (n-1) tree $\theta(n^2)$

Worst Case: when completed graph $e = \frac{n(n-1)}{2}$

$\theta\,(n^2\log n)$

3] Different Algorithm: Time Complexity of prim's Algorithm using arrays implementation is θ (n^2) in all cases

Note

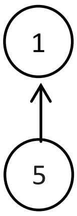
- Set Operations.

1. Find Algorithm
2. Union Algorithm

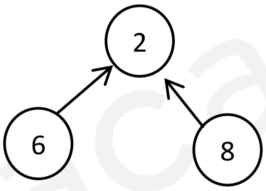
} Disjoint sets

(Used in kruskals MST implementation)

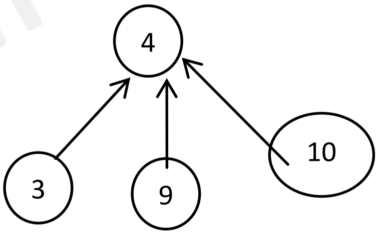
S_1= {1, 5 }



S_2= {2, 6, 8}

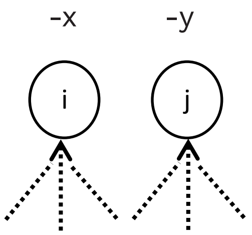


S_3= {3, 4, 9, 10 }



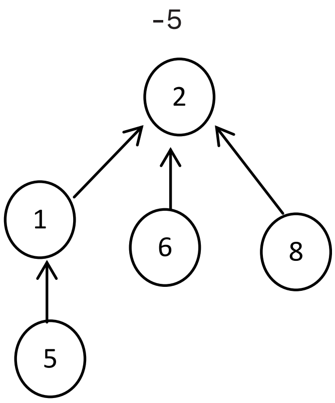
P[i]	-2	-3	4	-4	1	2	2	2	4	4
i	1	2	3	4	5	6	7	8	9	10

Weighted Union



-x < -y

Union (1,2)



find (5) = 2

Algorithm W union (i,j) Time comparison : θ(1)

```
{
    Ƶ= P[i] + P [j]
    if (P[i] < P[j] ) || set i has more element
   {
    P[j] = i ;}
    P [i] = z ;

      else {
        P[i]  =  j ;
        P [j] = z ;
    }
}
```

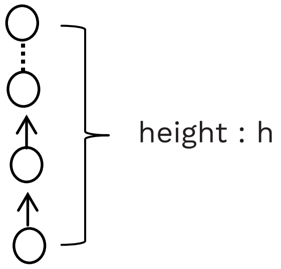
Find Algorithm

find (i)

return root of set of which element i is member.

4 = find (10)

4 = find (4)



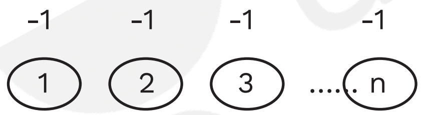
```
Algorithm find (i)
  {
     while ( P[i] > 0)
    {
      i = P[i]
      }
         return (i)
  }
```

Time Complexity find operation

: θ (height of set tree)

: O(logn)

Initially n sets each set one element



Initial sets

P[i]	-1	-1	-1	-1	-1	-1
i	1	2	3	4	.... n-1	n

a] If n sets combine into 1 set by using (n-1 union operations:

b] Minimum height of set tree ?

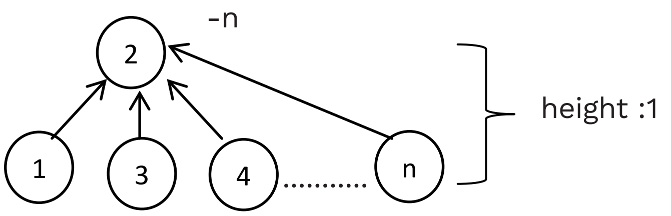
=> $\theta(1)$

Union (2, 1)

Union(2, 3)

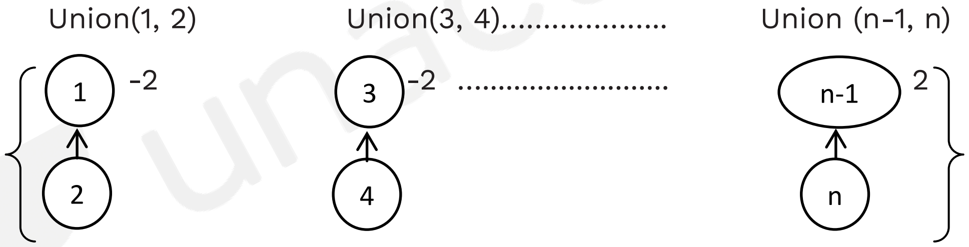
Union (2, 4)

Union (2, n)



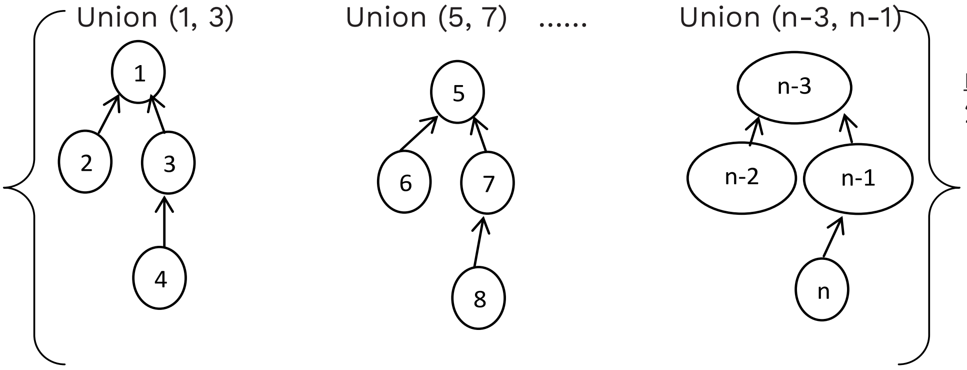
P[i]	2	-n	2	2	....	2
i	1	2	3	4	....	Ω

(11) Max height of set tree ?



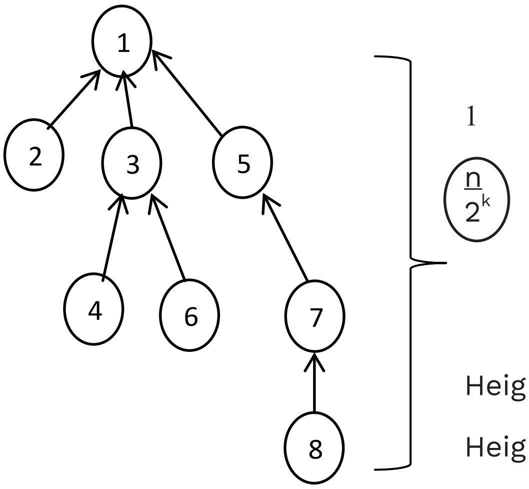
$\frac{n}{2}$ sets each set 2 element

Height : 1



$\frac{n}{2^2}$ sets 2^2 elements each

Height : 2



1

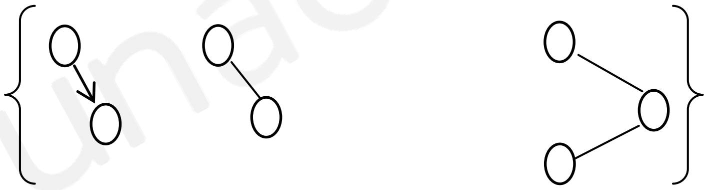
$\frac{n}{2^k}$ sets each sets 2^k elements

Height : K

Height : O(logn)

Krushkal’s Algorithm:-

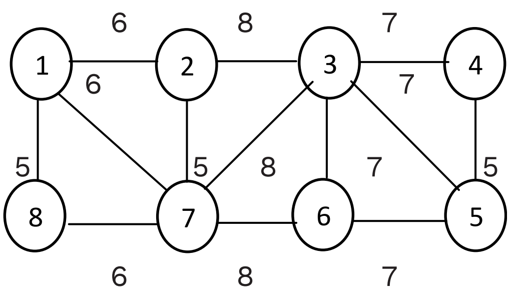
MST constructs edge by edge let “x” be set of edges included into MST so for these “x” edges may not connected.



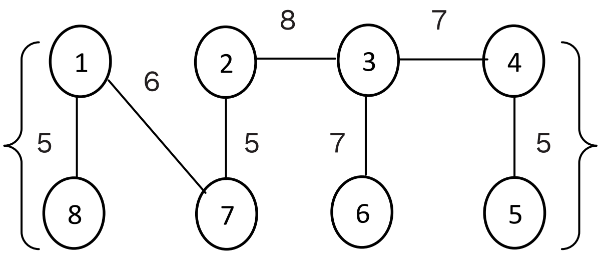
- Next including edge (u, v) into MST such that { x U (u,v)} is minimum increment of sum of edge costs && not forms cycle in MST.

Procedure of Krushkal’s

1. Delete Minimum cost edge (u, v) from graph G
2. Add (u, v into MST if (u, v) not forms cycle in MST. { set operations find && union algorithm used }
3. Repeat ① ② until MST becomes (n-1) edges or graph (G) becomes “0” edges.



MST



cost of MST : 43

Cycle Detection in MST construction:-

(u, v) is minimum cost edge deleted from graph

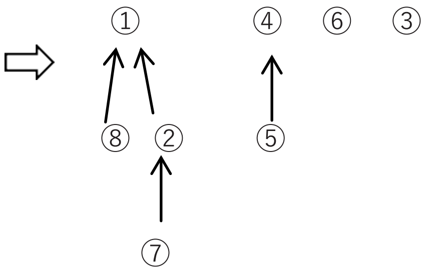
if (find (u) ≠ find (v))

then (u, v) not forms cycle in MST otherwise, (u,v) forms cycle.

{ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ } || initially

edges deleted from graph.

• (1,8)	Find (1) ≠ find (18)
	Union (1,8)
• (2,7)	Find (2) ≠find (7)
	Union (2,7)
• (4,15)	Find (4) ≠find (5)
	Union (4,5)
• (1,7)	Find (1) ≠find (7)
	Union (1,7)



X (1,2)	find (1)	= find(2)	|| forms cycle
X (8,7)	find(8)	= find(7)	|| forms cycle

Krushkal’s Algorithm implementation:-

- Build priority Queue (minimum heap) for all edges of graph using edge cost is priority
- Initialize set array for all vertices of graph

P[i]	-1	-1	-1	-1	....	-1
i	1	2	3	4		Ω

- Edges of MST stared in array t[1.....n-1 , 1...2]

	1	2
1	1	8
2	2	7
3	4	5
:		
:		
n-1		

Algorithm Krushkal’s (n, e cost [ ] [ ]) => θ(e loge)

{

O(e) { 1. Build minimum heap (ϕ) for all edges based on edge cost.

O(n) { 2. Initialize set array for all vertices
Minimum cost = 0; i =0 || number of edges in MST

```
3. While (i < N-1 && $\Phi$ is not empty)
     {
            (u,v) = deleted minimum cost edge ($\Phi$) → log e
            X = find (u) ; Y = find (v) →log n
e           if (x $\neq$ y) || (u,v) not forms cycle in MST
         {
            i = i + 1;
            t [i, 1] =u ; t[i, 2] = y
            minimum cost = minimum cost + cost (u,v)
            W union (X , Y) → $\theta$ (1)
         }
     }
     if (i < n-1) then G is disconnected no ST possible
     else
            return (t [ ] [ ] , minimum cost)
     }
     log e = log n
     log e = log $n^2$ = 2. Log n
```

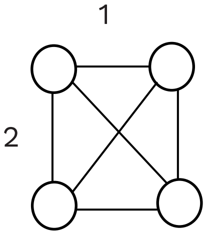
Time Complexity of krushkal's algorithm = θ (e log e) or θ (e log n)

- What is time complexity of krushkal's algorithm if edge costs are given in ascending order.

  θ (e log n)

  (u,v) = Deleted minimum cost edge (θ(1))

- edge cost { 1, 2, 3, 4, 5, 6, } for complete connected graph of 4 vertices
- What is maximum possible cost of minimal spanning tree ?

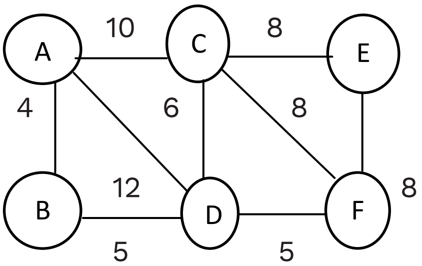


(1 + 2 + 4) =7

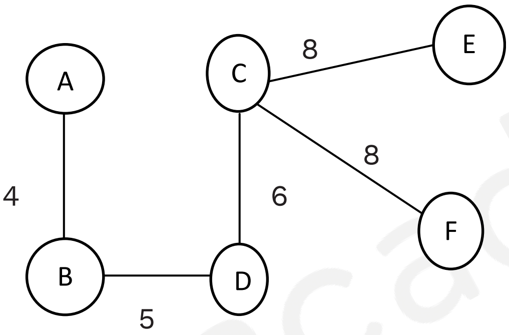
(1 + 2 + 3) =6

↓

assuming it forms cycle

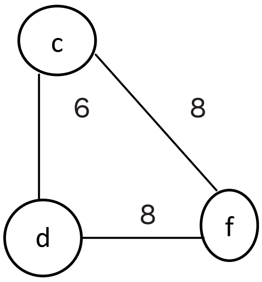


How many different MST’s possible



Total = 4_{C_2}-1 = 5 MST

6 selection (one selection forms cycle



1. CE && EF
2. CE && CF
3. CF && EF
4. DF && CE
5. DF && EF

Huffman coding

(compression technique)

[Optimal Encoding Method]

char to Binary : Encoding

Binary to char : Decoding

Encoding Methods

1) Fixed length encoding
2) Prefix encoding (Huffman coding]

1] fixed length encoding

Each char uses equal number of bits

$\underline{Xx}$=> 00 : a

2bits 01 : b

10 : c

11 : d

Xx....x => 2^n distinct char
nbits can be Encoding

[$\log_2 n$] bits => n distinct char Encode

Advantage : Easy to encode && decode

Disadvantage: Message length is large.

$$\left.\begin{array}{l}a\\b\\c\\z\\e\\f\\x\\g\end{array}\right\}$$

using fixed length encoding

[$\log_2 9$] = 4 bits

message length
[34] * 4 = 136 bits

2] Huffman Encoding :- (Optimal Prefix Encoding)

Most frequently used characters represented by less number of bits

- Least frequently used char represented by more number of bits.
- Proper prefix of any char code should not be the code of other char.

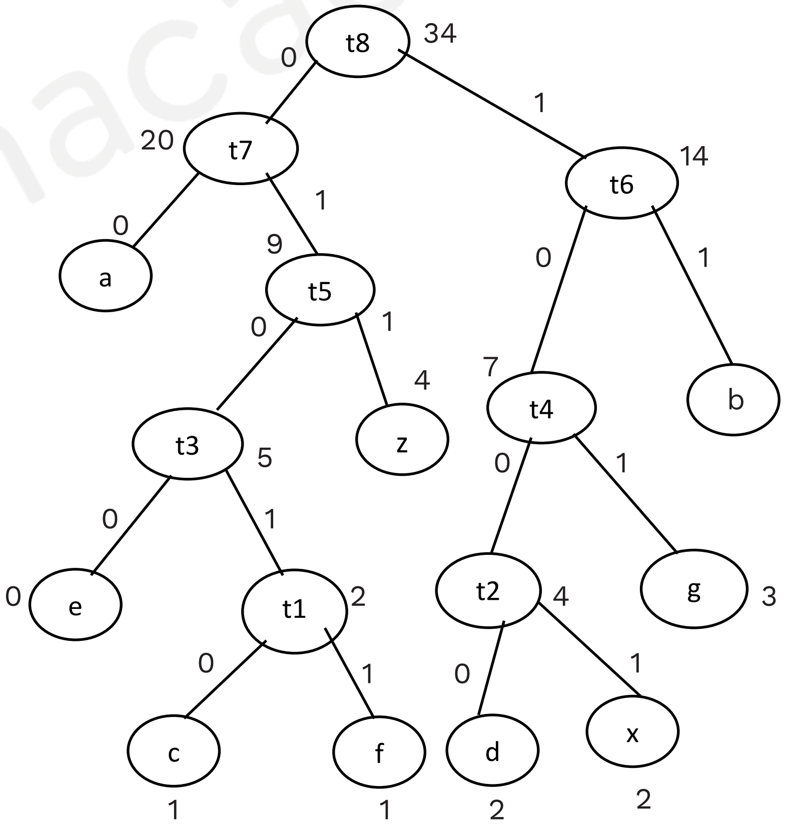
[110100] => a

{ 1
11
110 => b
1101 => c
11010 => d }

Decoding not possible (become Ambiguous violation (this is not in Huffman).

Char	frequency
a	11
b	7
c	1
d	2
e	3
f	1
x	2
z	4
g	3

2 way optimal Merge tree



Huffman coding

a => [00]

b => [11]

c => [01010]

d => [1000]

e => [0100]

f => [01011]

g => [101]

x => [1001]

z => [011]

sum of frequency count

2 * 11 + 7 * 2 + 1 *5 + 2 * 4 + 3 *4 + 1*5 + 3*3 + 2*4 + 4*3 = 95 bits

Aug bits per char = $\frac{95 \text{ bits}}{34}$

$$\boxed{\text{sum of frequency count } \sum_{i=1}^{n} Li * d_i = 2.79 \text{ bits/ char}}$$

2 way optimal Merge tree θ (nlogn)

|| n distinct char each char frequency count given.

1. Each char is one tree with height is frequency count of char.

   O(n) <= || build min heap for n distinct char.

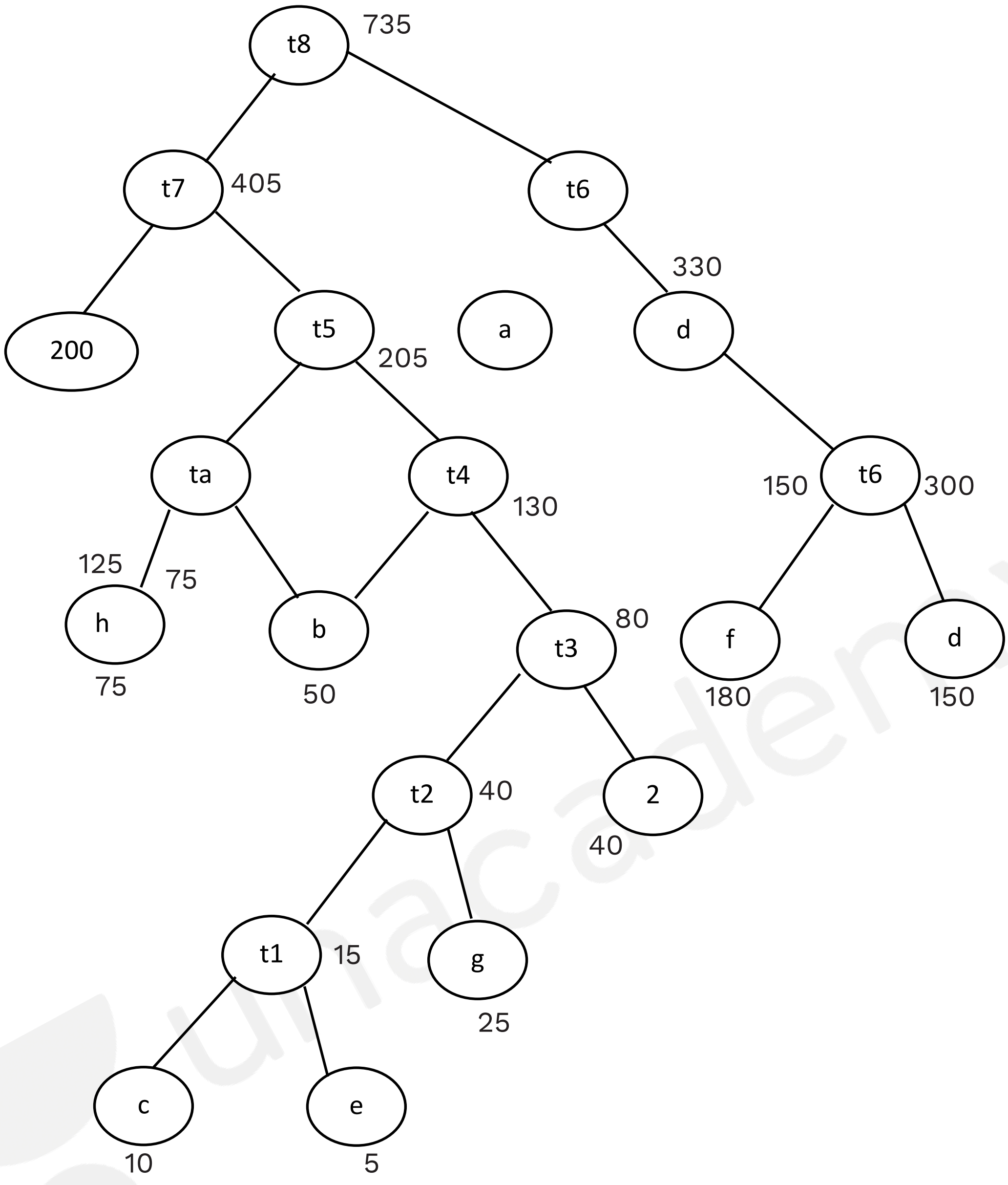
2. Delete "2" min cost trees && Merge into 2 min delete one tree with height sum of height logn + 1 insert of each tree.

3. Repeat (2) until or n trees become one tree [until min heap become empty]

Time Complexity to construct huffman tree : θ (nlogn)

{using heap Data Structure}

Q



Char:	a	b	c	d	e	f	g	h	2
Frequency:	200	50	10	150	5	180	25	75	40

sum of frequency count

= 2 x 200 + 4x50 + 6x10 + 2x150 + 6x5 + 2x180 + 6x25 + 3x75 + 5x40

= 400 + 200 + 60 + 300 + 30 + 360 + 150 + 225 + 225 + 200

= 900 + 105 + 510 + 425

= 1005 + 935

= 1935

number min bits required for any char : 2

I) Max bits for any char : 6

II) Avg bits per char = $\frac{1935}{735}$

Fixed length : 4bits [$\log_2 9$]

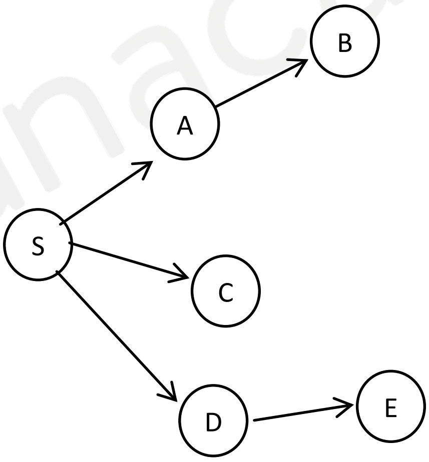
Total sum of frequency count

= 4 * 735 = 2940 bits

Single Source Shortest Path Algorithms:-

Graph G(n,e) && edge costs cost [ ] [ ] and source vertex (s)

Single source shortest path algorithm should determine shortest path distance from given source s to every vertex of graph



Source to every vertex shortest distance cost has to compute

- Dijkstras Algorithm (Greedy Algorithm)
- Bellman ford Algorithm (Not Greedy Algorithm)

Dijkstras Algorithm:

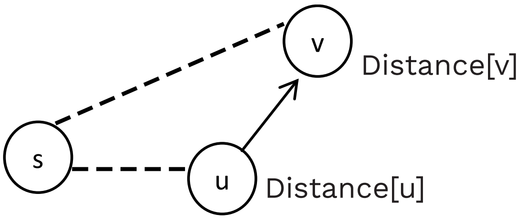
[Used as Routing Algorithm in CN]

1. Distance [ ] for all vertices initialize ∞.

   distance [s] =0

   prev [ ] for all vertices initialize "-1"

2. Choose vertex (u) which has minimum distance apply edge Relaxation for all adjacency vertices of u.

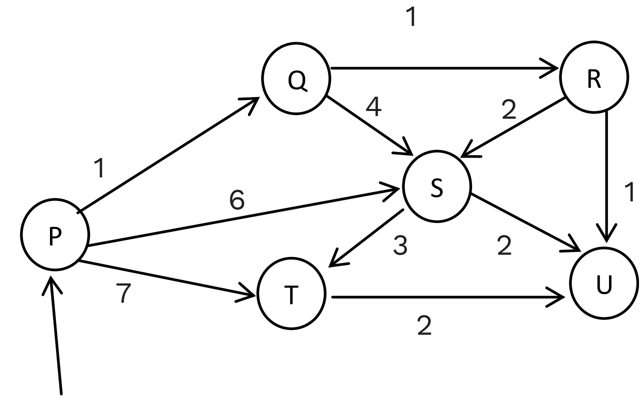


Distance[x] defined as shortest path distance from path (s) to vertex(x)

```
if ( distance[v] > distance[u] + cost (u,v) )
    {
        distance [v] = distance [u] + cost [u,v]
        prev [v] = u
    }
    || Vertex (u) computed Relaxation distance [u] is finalized
```

3. Repeat for all vertices

Q



Source

	P	Q	R	S	T	U
Distance	0	∞	∞	∞	∞	∞

	P	Q	R	S	T	U
Prev [Path]	-1	-1	-1	-1	-1	-1

Priority Queue || for all vertices priority (minimum heap) distance of vertex.

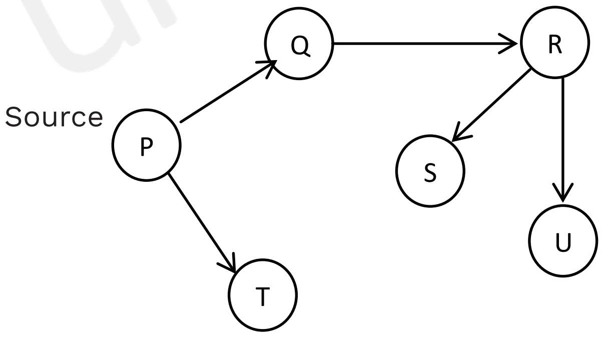
List- ~~PQR~~S T~~U~~

P Q R U S T →

- Order of vertices applied edge relaxation process [distance finalized]

  P Q R U ST

Dijkstra’s spanning tree



	List
P–Q	:1
P – Q –R	:2
P – Q – R–S	:4
P – Q – R–U	:2
P–T	:7

Shortest paths from source

```
Algorithm Dijkstras (G, n, e, cost [ ] [ ] , S)
  {
     for ( i = 1 ; i ≤ n ; i++)
       { distance [Vi] =+∞
θ(n)       prev [Vi] =-1
       }
           distance [S] = 0

  θ(n) [ Build minimum heap (Φ) for all vertices based on distance.
  While (Φ is not empty)

      {
           U = Deleted minimum distance vertex from log n minimum heap (Φ)
           V = find Adj vertices of u → degree [v] O(n)
           for (each vertex of V set)
           {
             if (distance [V] > distance [U] + cost (u,v)
ntimes       {
                distance [V] = distance [U] + cost (u,v) ;
      deg(u)    prev [V] = u ;
                  Key decrement (ϕ, V, distance [V] ; → log n
             }
           }
      }
                return (distance [ ], prev [ ] ) ;
     }
```

Time Complexity of Dijkstra’s Algorithm using

(1) Minimum heap && Adj list representation of graph.

$= \theta\ ((n + e) \log n\ )$

(2) Minimum heap && Adj matrix representation of graph

θ (n^2 + e log n)

(3) Dijkstra’s Algorithm implementation using arrays time complexity

: θ (n^2) in all cases

Q. What is time complexity of Dijkstra’s Algorithm using sorted linked list as priority Queue && Adj list representation of graph.

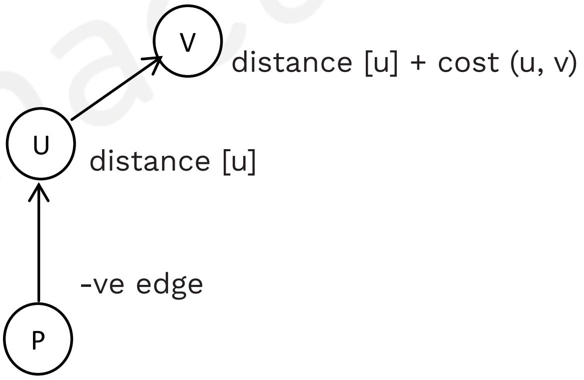
Time Complexity O(e. n)

Worst Case : O (n^3)

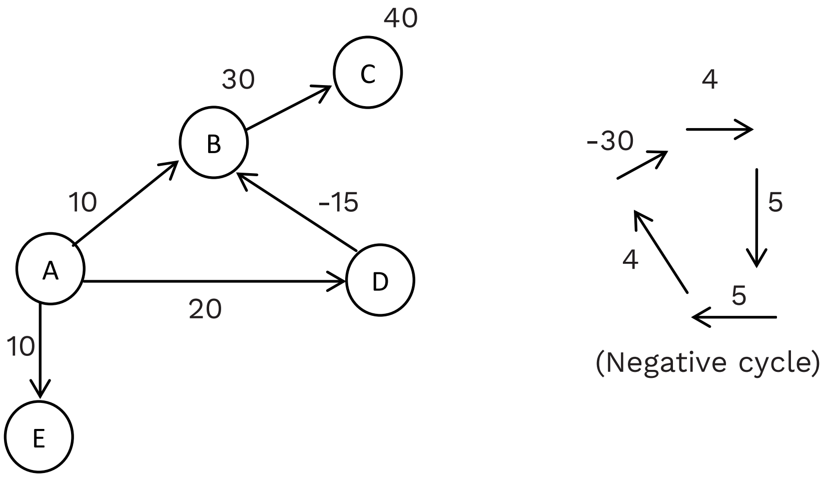
Unsorted array :- Hash is used to find the position for Key decrement.

Limitations of Dijkstra’s Algorithm:-

(1) Dijkstras Algorithm may failed to determine shortest path cost from source if negative edges in graph [Assume no negative cycle reachable from source]



Example:-



(Negative cycle)

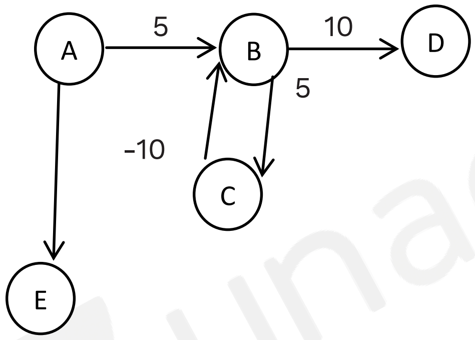
	A	B	C	D	F
distance	0	∞	∞	∞	∞

Result of Dijkstra’s

(2) Dijkstra’s Algorithm failed to “detect-negative cycle” reachable from source [if negative cycle reachable from source exists then not possible to find shortest path distance from source to vertices reachable through negative cycle]

Detection of negative cycle must require if reachable from source

Source



[A to D]

A – B – D	: 15
A – B – C – B – D	: 10
A – B – C – B – C – B – D	: 5

	A	B	C	D	E
Distance	0	∞	∞	∞	∞

=> Dijkstras o/p

Bellman Ford Algorithm;-

- Not Greedy Algorithm
- Can find shortest path cost for all vertices from source even if –ve edge with no negative cycle reachable from source exists.
- Detects negative cycle reachable from source if exists.
- Not detects every negative cycle if it is not reachable from source

→ Procedure:-

1. Initialize distance [Vi] = + ∞

   prev [Vi] = -1

   Distance [s] = 0
2. Apply edge relaxation for every edge of graph && Repeat (n-1) times.
3. If n^{th} edge relaxation distance value < $(n-1)^{th}$ time edge relaxation for any vertex. then –ve cycle reachable from source exists.

   Otherwise no negative cycle which is reachable from source is exists.

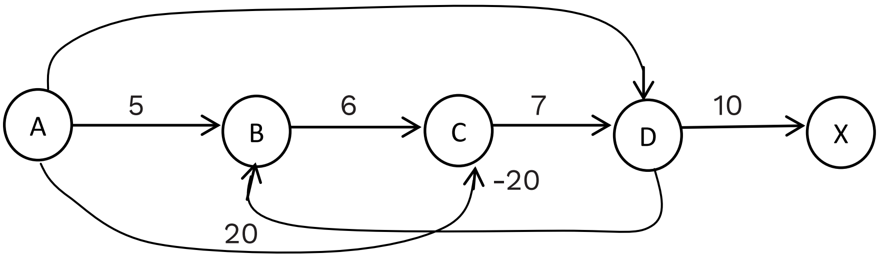
Ⓐ...............................Ⓧ => resulted distance source from s to x

1. Almost (n-1) after (n-1) times edges can be in edge relaxation shortest path from A to X.
2. n^{th} time edge Relaxation

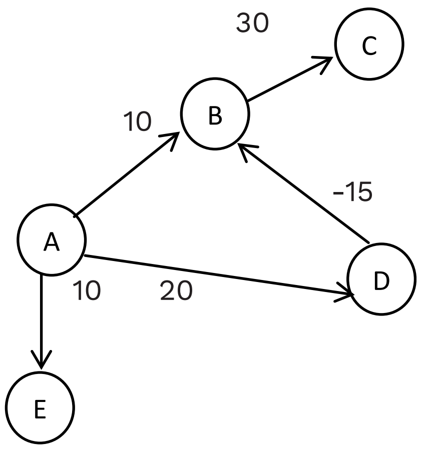
   distance [X] < distance [X]

   for n^{th} time edge relaxation — for $(n-1)^{th}$ time edge relaxation

"X" is going through negative cycle from source



Example:



Order can be different

B – C

D – B

A – B

A – D

A – E

(a) Bellman Ford Algorithm:

(b)

	A	B	C	D	E
distance	0	∞	∞	∞	∞

A	B	C	D	E
-1	-1	-1	-1	-1

n-1					
i= 1	0	10	∞	20	10
i =2	0	5	40	20	10
i =3	0	5	35	20	10
i = 4	0	5	35	20	10
i = 5	0	5	35	20	10

no negative cycle

[correct shortest path from source]

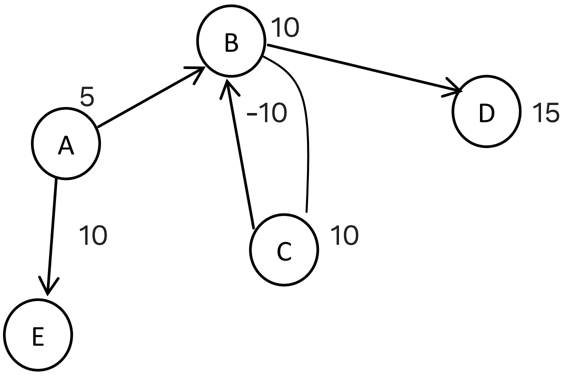
(c) Dijkstras Algorithm result:

	A	B	C	D	E
Distance	0	5	40	20	10

↓

Wrong

Example: 2



A – B

A – E

B – C

B – D

C – B

(a) Bellman ford Algorithm

	A	B	C	D	E
Distance	0	∞	∞	∞	∞

		A	B	C	D	E
	i = 1	0	0	10	15	10
(n-1))	i = 2	0	-5	5	10	10
times	i = 3	0	-10	0	5	10
	i = 4	0	-15	-5	0	10
	i = 5	0	-20	-10	-5	10

B, C, D going through

|| negative cycle from source

|| negative cycle reachable from source exists.

(b) Dijkstra's Algorithm.

	A	B	C	D	E
distance	0	0	10	15	10

```
Algorithm Bellman ford (n, e, cost [ ] [ ], edge [ ] [ ] S )
    {
        || Initialization
          for ( i = 1 ; i ≤ n ;i++)
          {
O(n)        distance [Vi] = + ∞ , prev [Vi] = -1 ; }
            distance [s] = 0;
    || Apply edge Relaxation for all edges and repeat (n-1) times
        for (i = 1 ; i ≤ n-1 ; i++)
          {
            for (each edge (u,v) of graph)
              {
                 if (distance [v] > distance [u] + cost (u,v) )
                     {
$\theta(n.e)^{n-1}$      distance [v] = distance [u] + cost(u,v)
              e          prev [v] = u
                  }
              }
          }

        || -ve cycle detection
            for (each edge (u,v) of graph )
          {
               if (distance [v] > distance [u] + cost (u,v)
                 { negative cycle reachable from source exists.
$\theta(e)$              return (false)}
              }
                     return (distance [ ] , pre [ ]
          }
```

Time of Bellman ford : $\theta(n.e)$

Worst Case Time Complexity of Bellman Ford $\theta(n^3)$

- **Fraction Knapsack problem:-]**

  Given n objects and m capacity knapsack each object i with weight wi and profit pi.

  if Xi fraction of object I included into knapsack with required Wi * ϰi weight && results Pi * xi profit where 0 ≤ ϰi ≤ 1

  In order to include object into Knapsack maximise profit.

  Find fraction values { X_1, X_2, X_3......Xn}

  S.t maximize $\sum_{i=1}^{n} Pi * Xi$

  Subjected to $\sum Wi * ϰi \leq n$

  $0 \leq ϰi \leq 1$

Practice Problems:-

Q.1 A complete binary min-heap is made by including each integer in [1, 1023] exactly once. The depth of a node in the heap is the length of the path from the root of the heap to that node. Thus, the root is at depth 0. The maximum depth at which integer 9 can appearis_____.

[GATE-2016]

Q.2 Given two inputs BST and Min heap tree with n-nodes. To get the sorted order which is better and how much time?

Ans: BST and O(n)

Q.3 Consider the following “Max-heapify” algorithm. Array has size atleast n and 1 ≤ i ≤ n. After applying the. Max-heapify rooted at A[i], the result will be subtree of A[1,....n] rooted at A[i] is max heap. [Assume that except root A[i], all its children satisfied heap property]

```
int , m;
p = i;
while(X)
```

```
{
    if (Y && Z)
    m = $2^p$ +1;
    else m = $2^p$;
    if (A[p] < A[m])
 {
    Swap (A[p], A[m];
    p = m;
 }
    else
    return
}
```

Find missing statements at X, Y and Z respectively to apply the heapify for subtree rooted at A[i].

a) p ≤ n, (2p +1) ≥ n, A [2p + 1] > A [2p]

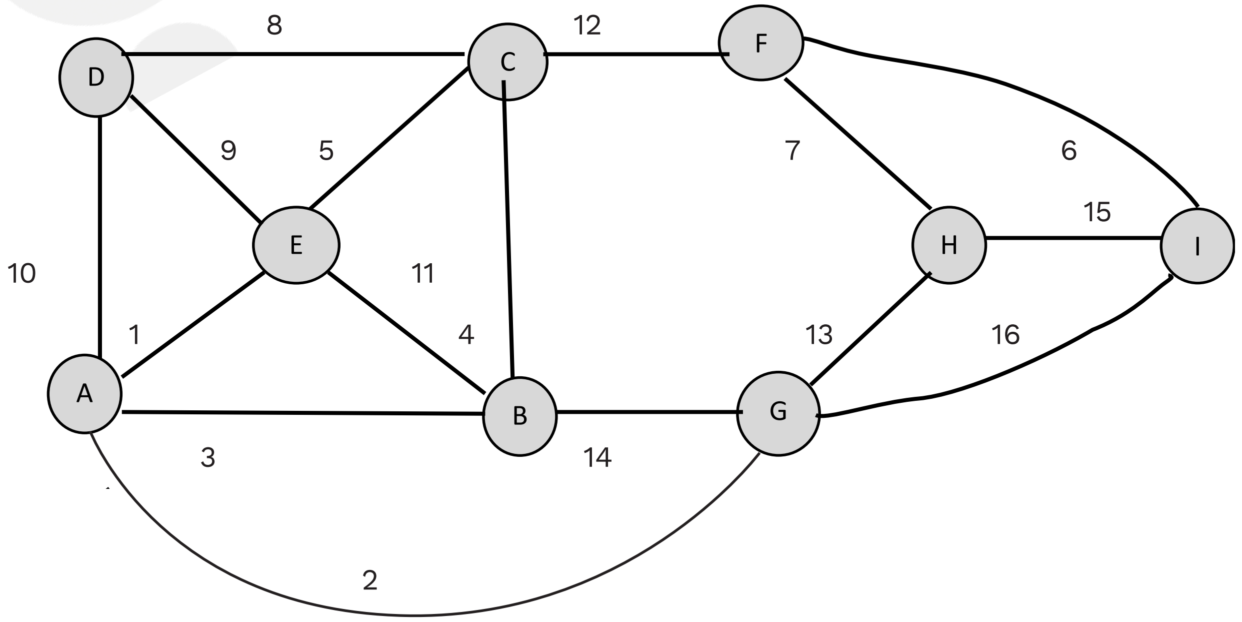
b) 2p ≤ n, (2p + 1) ≤ n, A[2p + 1] > A [2p]

c) 2p ≤ n, (2p + 1] ≥ n, A [2p + 1] < A [2p]

d) p ≤ n, (2p + 1) ≤ n, A [2p + 1] < A [2p]

Ans: (b)

Q.4 Consider the weighted undirected graph below



Assume Prim's algorithm and kruskal's algorithm are executed on the above graph to find the minimum spanning tree. For a particular edge (e_i) which is included is minimum spanning tree and the position of an edge in minimum spanning tree is denoted by e_{pi}. Where 1≤ e_{pi}≤ 8 (where position defines the order in which edges are included in the MST). Then what is the maximum value of $|(e_{pi})_{prim's}$- $|(e_{pi})_{kruskal's}|$?

Ans: (6)

Q.5 Complexity of Kruskal algorithm for finding the minimum cost spanning tree of an undirected graph contain n-vertices and m-edges, if the edges are already sorted.

Ans: O(m)

Q.6 Let T be a MST of G. Suppose that we decreased the weight of one of the edge present in G but not in T. Then how much time will take to construct MST for the modified graph G?

Ans: O(v)

Q.7 The height of a binary tree is the maximum number of edges in any root to leaf path.

The maximum number of nodes in a binary tree of height h is

a) $2^h - 1$

b) 2^{h-1} -1

c) 2^{h+1} -1

d) 2^{h+1}

[GATE-2007]

Q.8 The maximum number of binary trees that can be formed with three unlabelled nodes is

a) 1

b) 5

c) 4

d) 3

[GATE-2007]

Q.9 A program takes as input a balanced binary search tree with n leaf nodes and computes the value of a function g(x) for each node x. If the cost of computing g(x) is min (number of leaf-nodes in leaf-subtree of x) then the worst case time complexity of the program is

a) $\Theta(n)$

b) $O(n \log n)$

c) $O(n)^2$

d) $O(n^2 \log n)$

[GATE-2004]

Q.10 In a complete k-ary, every internal node has exactly k children. The number of leaves in such a tree with n internal nodes is

a) n k

b) (n – 1) k + 1

c) n (k – 1) + 1

d) n (k – 1)

[GATE-2005]

Q.11 Suppose there are log n sorted lists of n/log n elements each. The time complexity of producing a sorted list of all these elements is:

(Hint: Use a heap data structure)

a) $O(n \log\log n)$

b) $\Theta(n \log n)$

c) $\Omega(n \log n)$

d) $\Omega(n^{3/2})$

[GATE-2005]

Q.12 How many undirected graphs (not necessarily connected) can be constructed out of a given set V = $\{V_1, V_2, \dots, V_n\}$ of n vertices?

a) $\frac{n(n-1)}{2}$

b) 2^n

c) n!

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

[GATE-2001]

Q.13 The number of leaf nodes in a rooted tree of n nodes, with each node having 0 or 3 children is

a) $\frac{n}{2}$

b) $\frac{(n-1)}{3}$

c) $\frac{(n-1)}{2}$

d) $\frac{(2n + 1)}{3}$

[GATE-2001]

Q.14 Consider a rooted n node binary tree represented using pointers. The best upper bound on the time required to determine the number of subtrees having exactly 4 nodes is $O(n^a \log^b n)$. Then the value of a +10b is_____.

[GATE-2014]

Q.15 Adjacency list is preferred over adjacency matrix when the graph is

a) Planar

b) Dense

c) Clique

d) None of these

[DRDO-2008]

Q.16 The minimum number of interchanges needed to convert the array 89, 19, 40, 17, 12, 10, 2, 5, 7, 11, 6, 9, 70 into a heap with maximum element at the root is

a) 0

b) 1

c) 2

d) 2

[GATE-2006]

Q.17 The number of elements that can be stored in Θ(log n) time using heap sort is

a) Θ(1)

b) Θ (√log n)

c) $\Theta\left(\frac{\log n}{\log\log n}\right)$

d) Θ(logn)

[GATE-2013]

Q.18 An algorithm performs $(\log N)^{1/2}$ find operations, N insert operations, $(\log N)^{1/2}$ delete operations, and $(\log N)^{1/2}$ decrease-key operations on a set of data items with keys drawn from a linearly ordered set. For a delete operation, a pointer is provided to the record that must be deleted. For the decrease-key operation, a pointer is provided to the record that has its key decreased. Which one of the following data structures is the most suited for the algorithm to use, if the goal is to achieve the best total asymptotic complexity considering all the operations?

a) Unsorted array

b) Min-heap

c) Sorted array

d) Sorted doubly linked list

Q.19 Construct the Max Heap assuming the following set of integers were inserted into it in given order 20, 32, 1, 3, 4, 5, 6, 7, 10, 23, 45 Postorder traversal of the resultant max heap was stored in a array A with an index variable i inorder (Starting from 0). Similarly level order traversal was stored in the array B using index variable j in order (Starting from O). For particular element, respective i and j location values from A and B were obtained and | i – j | is calculated. What could be the maximum possible value for | i – j |?

Q.20 A data structure is required for storing a set of integers such that each of the following operations can be done in (log n) time, where n is the number of elements in the set.

1. Deletion of the smallest element.

2. Insertion of an element if it is not already present in the set.

Which of the following data structures can be used for this purpose?

a) A heap can be used but not a balanced binary search tree

b) A balanced binary search tree can be used but not a heap

c) Both balanced binary search tree and heap can be used

d) Neither balanced binary search tree nor heap can be used

[ISRO-2009]

Q.21 Postorder traversal of a binary search tree is given as follows 34, 40, 55, 60, 50, 100. Then the given tree is:

a) Minheap tree

b) Maxheap true

c) Strict binary tree

d) None of these

Q.22 The worst case running time complexity to search for an element in a balanced binary search tree with (2n)! elements?

a) $O(n^n)$

b) $O(n \log n)$

c) $O(n)$

d) $O(n^2)$

Q.23 Consider the process of inserting an element into a Max Heap, where the Max Heap is represented by an array. Suppose we perform a binary search on the path from the new leaf to the root to find the position for the newly inserted element, the number of comparisons performed is

a) $\theta(\log_2 n)$

b) $\theta(\log_2 \log_2 n)$

c) $\theta(n)$

d) $\theta(n \log_2 n)$

[GATE-2007]

Q.24 We have a binary heap on n elements and wish to insert n more elements (not necessarily one after another) into this heap. The total time required for this is

a) $\Theta(\log n)$

b) $\Theta(n)$

c) $\Theta(n \log n)$

d) $\Theta(n^2)$

[GATE-2008]

Q.25 Which one of the following statements are correct regarding Bellman-Ford shortest path algorithm?

P: Always finds a negative edge weight cycle if one exists.

Q: Find whether any negative edge weight cycle reachable from the source.

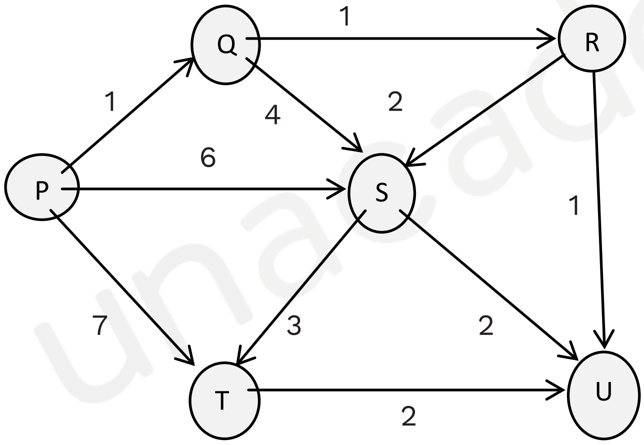
a) P only

b) Q Only

c) Both P and Q

d) Neither P nor Q

Q.26 Suppose we run Dijkstra's single source shortest-path algorithm on the following edge-weighted directed graph with vertex P as the source.



In what order do the nodes get included into the set of vertices for which the shortest path distances are finalized

a) P, Q, R, S, T, U

b) P, Q, R, U, S, T

c) P, Q, R, U, T, S

d) P, Q, T, R, U, S

[GATE-2004]

Q.27 Let G (V, E)an undirected graph with positive edge weights. Dijkstra's single source-shortest path algorithm can be implemented using the sorted linked list data structure and adjacency list. What will the time complexity?

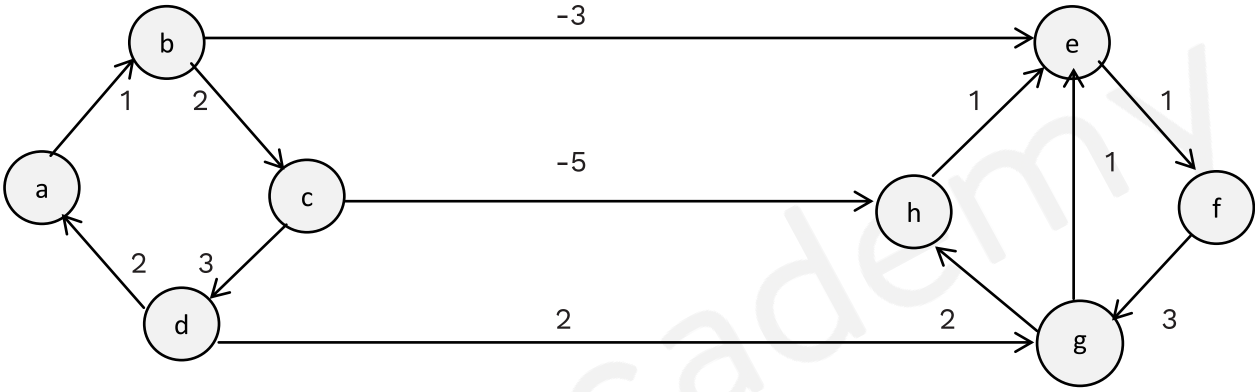
a) O(E| V |)

b) O(| V $|^3$)

c) O(| V | log | V |)

d) O((| E | + | V |) log | V |)

Q.28



Dijkstra's single source shortest path algorithm when run from vertex a in the above graph, computes the corrects shortest path distance to

a) Only vertex a

b) Only vertices a, e, f, g, h

c) Only vertices a, b, c, d

d) All the vertices

[GATE-2008]

Q.29 Which one of the following statements are correct regarding Bellman-Ford shortest path algorithm?

P: Always finds a negative edge weight cycle if one exists.

Q: Find whether any negative edge weight cycle reachable from the source.

a) P Only

b) Q only

c) Both P and Q

d) Neither P nor Q

Q.30 Consider the following statements:

I. For every weighted graph and any two vertices s and t, Bellman-ford algorithm starting at s will always return a shortest path to t.

II. At the termination of the Bellman-Ford algorithm, even if graph has negative weight cycle, a correct shortest path is found for a vertex for which shortest path is well-defined.

Which of the above statements are true?

a) Only I

b) Only II

c) Both I and II

d) None of these

Q.31 If graph contains negative weight edges then which of the following is correct when we run Dijkstra's algorithm?

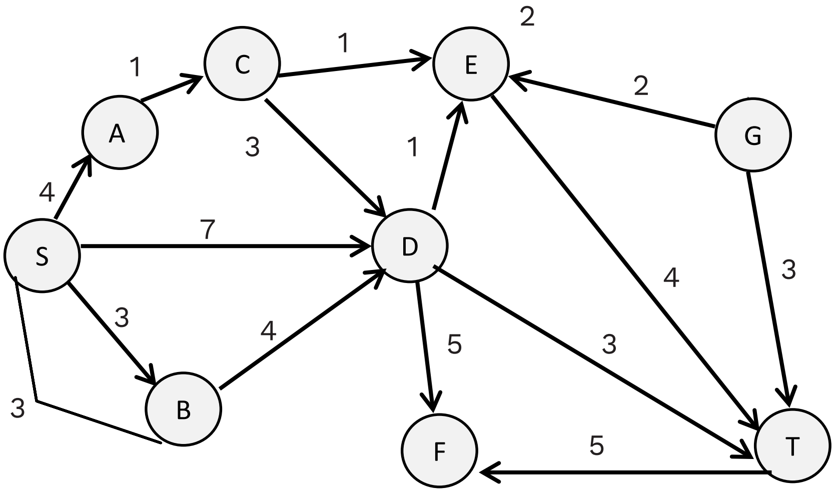
a) It may not terminate

b) It terminates but may produce incorrect results

c) It never terminates due to cycles in graph

d) None of these

Q.32 Consider the directed graph shown in the figure below. There are multiple shortest paths between vertices S and T. Which one will be reported by Dijkstra's shortest path algorithm? Assume that, in any iteration, the shortest path to a vertex *V* is updated only when a strictly shorter path to *V* is discovered.



a) SDT

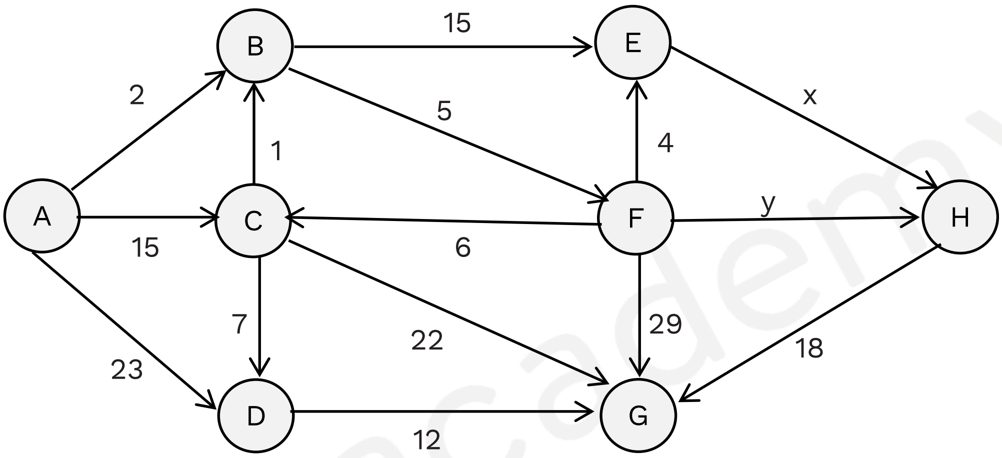
b) SVDT

c) SACDT

d) SACET

[GATE-2012]

Q.33 Suppose that you are running Dijkstra's algorithm on the edge-weighted digraph below, starting from vertex A.

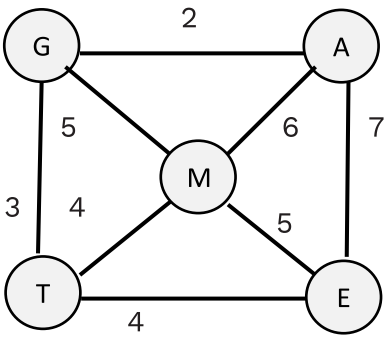


The table gives 'Distance' and 'Parent' entry of each vertex after E has been deleted from the priority queue and relaxed.

Vertex	Distance	Parent
A	0	NULL
B	2	A
C	13	F
D	23	A
E	11	F
F	7	B
G	36	F
H	19	E

What could be the possible value of expression x + y?

Q.34 Assume Dijkstra's algorithm is used to find the shortest paths from node 'G' in the following graph.



Find the number of edges which are not included in any of the shortest paths from node G.

Q.35 What is the upper bound on the number of edge disjoint spanning trees in a complete graph of n vertices

a) n

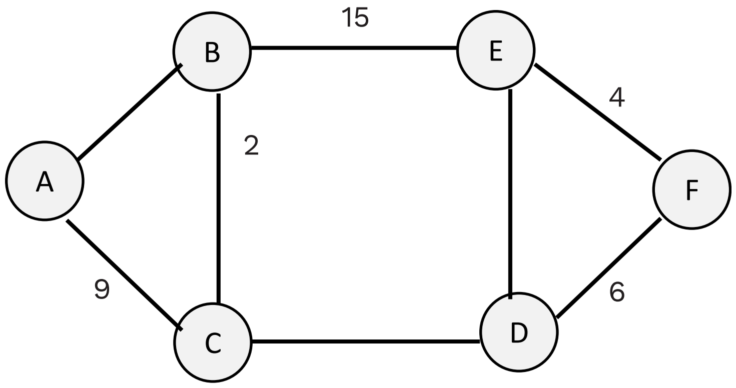
b) n – 1

c) $\left[\frac{n}{2}\right]$

d) $\left[\frac{n}{3}\right]$

[DRDO-2009]

Q.36 The graph shown below has 8 edges with distinct integer edge weights. The minimum spanning tree (MST) is of weight 36 and contains the edges: { (A, C), (B, C), (B, E), (E, F), (D, F)}. The edge weights of only those edges which are in the MST are given in the figure shown below. The minimum possible sum of weights of all 8 edges of this graph is_____.



Q.37 Let G be an undirected connected graph with distinct edge weight. Let e_{max} be the edge with maximum weight and e_{min} with minimum weight. Which of the following statements is false?

a) Every minimum spanning tree of G must contain e_{min}

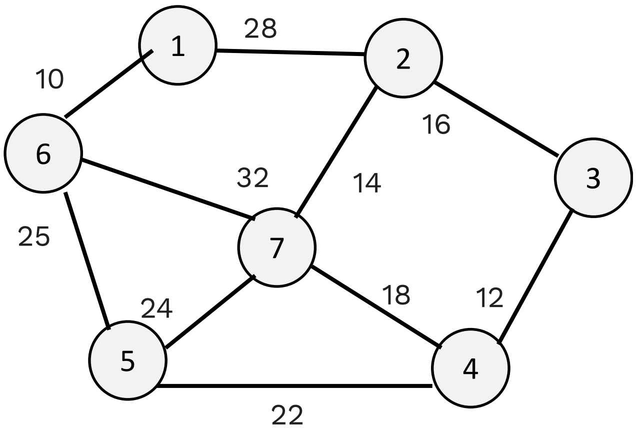
b) If e_{max} is in a minimum spanning tree, then its removal must disconnect G

c) No minimum spanning tree contains e_{max}

d) G has a unique minimum spanning tree

[GATE-2000]

Q.38 Consider the following graph where the numbers denotes the weight of the particular Edge



Now, calculate the minimum cost spanning tree of the above graph using either prim's or Kruskal's algorithm.

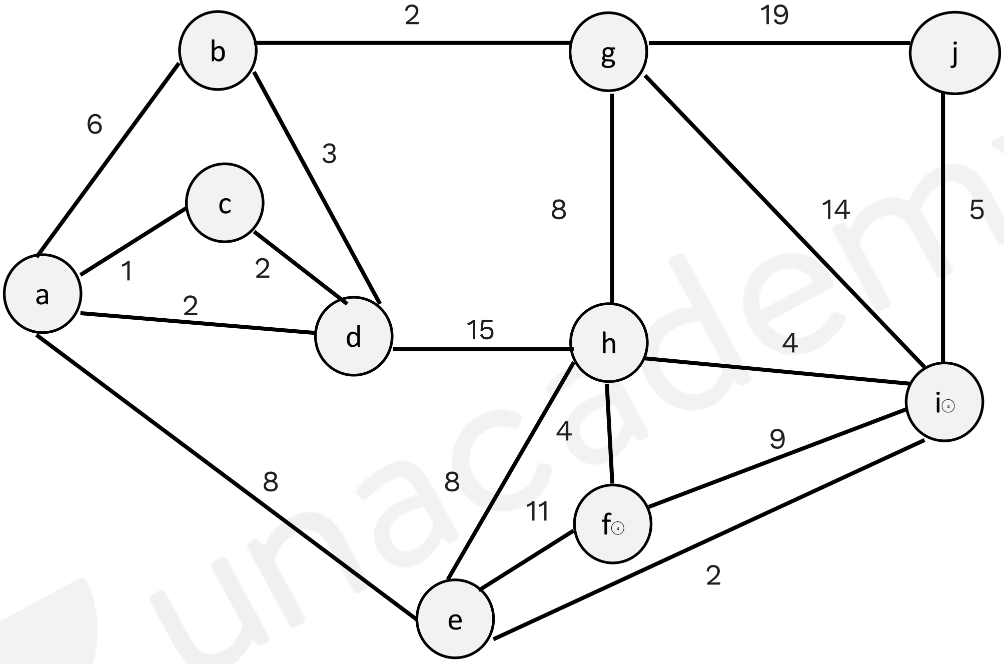
a) 92

b) 99

c) 102

d) 123

Q.39 What is the weight of a minimum spanning tree of the following graph?



a) 29

b) 31

c) 38

d) 41

[GATE-2003]

Common Data for Q. 40 & Q.41

Consider a complete undirected graph with vertex set {0, 1, 2, 3, 4}. Entry W_{ij} in the matrix W below is the weight of the edge {i, j}.

$$W = \begin{pmatrix} 0 & 1 & 8 & 1 & 4 \\ 1 & 0 & 12 & 4 & 9 \\ 8 & 12 & 0 & 7 & 3 \\ 1 & 4 & 7 & 0 & 2 \\ 4 & 9 & 3 & 2 & 0 \end{pmatrix}$$

Q.40 What is the minimum possible weight of a spanning tree T in this graph such that vertex 0 is a leaf node in the tree T?

a) 7

b) 8

c) 9

d) 10

[GATE-2010]

Q.41 What is the minimum possible weight of a path P from vertex 1 to vertex 2 in this graph such that P contains at most 3 edges?

a) 7

b) 8

c) 9

d) 10

[GATE-2010]

Q.42 Consider a weighted complete graph G on the vertex set {v_1,v_2,....., vn} such that the weight of the edge (v_i, v_j) is 2 | i – j |. The weight of a minimum spanning tree of G is

a) n – 1

b) 2n – 2

c) (n/2)

d) n^2

[GATE-2006]

Q.43 Let w be the minimum weight among all edge weights in an undirected connected graph. Let e be a specific edge of weight w. Which of the following is FALSE?

a) There is a minimum spanning tree containing e.

b) If e is not in a minimum spanning tree T, then in the cycle formed by adding

e to T, all edges have the same weight.

c) Every minimum spanning tree has an edge of weight w

d) e is present in every minimum spanning tree

[GATE-2007]

Q.44 Consider the following statements:

I. Let T be a minimum spanning tree of a graph G. Then for any two vertices u and v the path from u to v in T is the shortest path from u to v in the graph G.

II. Suppose that average edge weight for a graph G is A_{avg}. Then the minimum spanning tree of G will have weight at most (n – 1) A_{avg}. Where n is number of vertices in graph G.

Which of above statements are true?

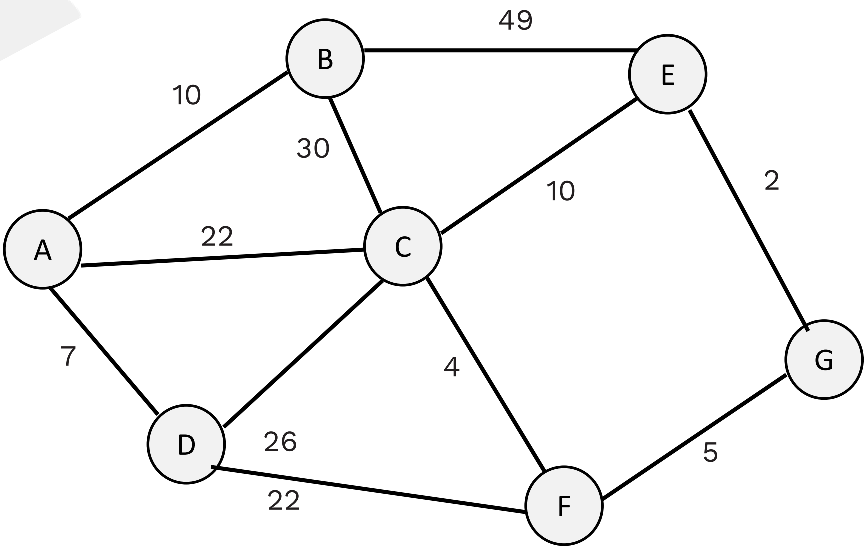
a) Only I

b) Only II

c) Both I and II

d) None of these

Q.45 Consider the undirected graph below:



Using Prim's algorithm to construct a minimum spanning tree starting with node A, which one of the following sequences of edges represents a possible order in which the edges would be added to construct the minimum spanning tree?

a) (E, G), (C, F), (F, G), (A, D), (A, B), (A, C)

b) (A, D), (A, B), (A, C), (C, F), (G, E), (F, G)

c) (A, B), (A, D), (D, F), (F, G), (G, E), (F, C)

d) (A, D), (A, B), (D, F), (F, C), (F, G), (G, E)

[GATE-2004]

Q.46 Let G be a weighted undirected graph and e be an edge with maximum weight in G. Suppose there is a minimum weight spanning tree in G containing the edge e. Which of the following statements is always TRUE?

a) There exists a cutest in G having all edges of maximum weight.

b) There exists a cycle in G having all edges of maximum weight.

c) Edge e cannot be contained in a cycle.

d) All edges in G have the same weight.

[GATE-2005]

Q.47 G= (V,E) is an undirected simple graph in which each edge has a distinct weight, and e is a particular edge of G. Which of the following statements about the minimum spanning trees (MSTS) of G is/are TRUE?

I. If e is the lightest edge of some cycle in G, then every MST of G includes e.

II. If e is the heaviest edge of some cycle in G, then every MST of G excludes e.

a) I only

b) II only

c) Both I and II

d) Neither I nor II

[GATE-2016]

Q.48 In a simple connected undirected graph with n nodes (where n ≥ 2) the maximum number of nodes with distinct degrees is

a) n - 1

b) n – 2

c) n – 3

d) 2

[DRDO-2008]

Q.49 A complete n-array tree is a tree in which each node has n children or no children. Let *I* be the number of internal nodes and *L* be the number of leaves in a complete n-array tree. If *L* = 41, and *I* = 10, what is the value of n

a) 3

b) 4

c) 5

d) 6

[GATE-2007]

Q.50 Suppose we have a balanced binary search tree holding n-numbers. We are given tow numbers L and H and wish to sum up all the numbers in T that lie between L and H. Suppose there are m such numbers in T.

If the tightest upper bound on the time to compute the sum is $O(n^{a} \log^{b} n + m^{c} \log^{d} n)$, the value of a + 10b + 100c +1000d is_.

[GATE-2014]

Q.51 Consider any array representation of an n element binary heap where the elements are sorted from index 1 to index n of the array. For the element sorted at index I of the array (I ≤ n), the index of the parent is

a) $i - 1$

b) $\left[\frac{i}{2}\right]$

c) $\left[\frac{i}{2}\right]$

d) $\frac{(i + 1)}{2}$

[GATE-2001]

Q.52 In a heap with n elements with the smallest element at the root, the 7th smallest element can be found in time

a) $\Theta(n \log n)$

b) $\Theta(\log n)$

c) $\Theta(n)$

d) $\Theta(1)$

[GATE-2003]

Q.53 The minimum element in a max-heap represented by an array can be computed in time

a) $\Theta(n \log n)$

b) $O(n)$

c) $\Theta(n^2)$

d) $O(1)$

[GATE-2009]

Q.54 A weight-balanced tree is a binary tree in which for each node, the number of nodes in the left sub tree is at least half and at most twice the number of nodes in the right sub tree. The maximum possible height (number of nodes on the path from the root to the furthest leaf) of such a tree on n nodes is best described by which of the following?

a) $\log_2 n$

b) $\log_{4/3} n$

c) $\log_3 n$

d) $\log_{3/2} n$

[GATE-2002]

Q.55 A binary search tree is used to locate the number 43. Which of the following probe sequences are possible and which are not?

a)	61	52	14	17	40	43
b)	2	3	50	40	60	43
c)	10	65	31	48	37	43
d)	81	61	52	14	41	43
e)	17	77	27	66	18	43

[GATE-1996]

Q.56 Let T(n) be the number of different binary search trees on n distinct elements. Then

$$T(n)=\sum_{k=1}^{n} T(k-1)\,T(x)$$

a) n – k + 1

b) n – k

c) n – k – 1

d) n – k – 2

[GATE-2003]

Q.57 Which one of the following arrays satisfied max-heap property?

a) 16, 10, 12, 8, 3, 5

b) 16, 8, 5, 10, 12, 3

c) 16, 12, 8, 3, 5, 10

d) 10, 16, 12, 8, 5, 3

[DRDO-2008]

Common Data for Q.58 & Q.59

A 3-ary max heap is like a binary max heap, but instead of 2 children, nodes have 3 children. A 3-aray heap can be represented by an array as follows: The root is stores in the first location, a[0], nodes in the next level, from left to right, is stored from a[1] to a[3]. The nodes from the second level of the tree from left to right are stored from a [4] location onward. An item x can be inserted into a 3-ary heap containing n items by placing x in the location a [n] and pushing it up the tree to satisfy the heap property.

Q.58 Which one of the following is a valid sequence of elements in an array representing 3-ary max heap?

a) 1, 3, 5, 6, 8, 9

b) 9, 6, 3, 1, 8, 5

c) 9, 3, 6, 8, 5, 1,

d) 9, 5, 6, 8, 3, 1

[GATE-2006]

Q.59 Suppose the elements 7, 2, 10, and 4 are inserted, in that order, into the valid 3-ary max heap found in the question, Q.58 Which one of the following is the sequence of items in the array representing the resultant heap?

a) 10, 7, 9, 8, 3, 1, 5, 2, 6, 4

b) 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

c) 10, 9, 4, 5, 7, 6, 8, 2, 1, 3

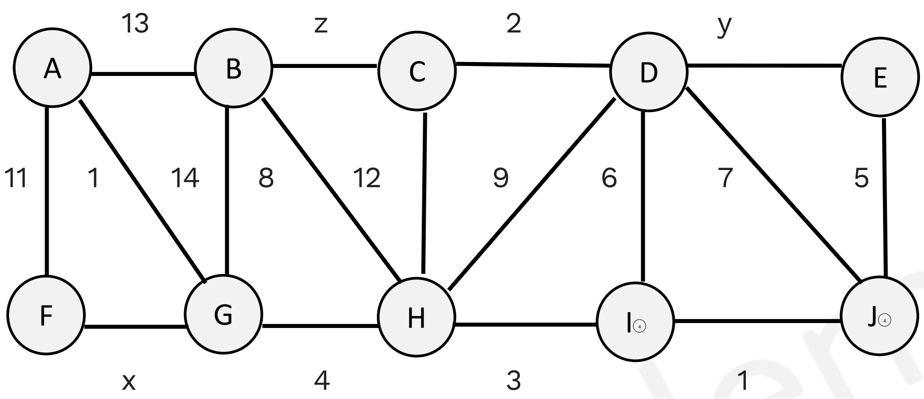
d) 10, 8, 6, 9, 7, 2, 3, 4, 1, 5

[GATE-2006]

Q.60 Let G be a complete undirected graph on 4 vertices, having 6 edges with weights being 1,2, 3, 4, 5, and 6. The maximum possible weight that a minimum weight spanning tree of G can have is______.

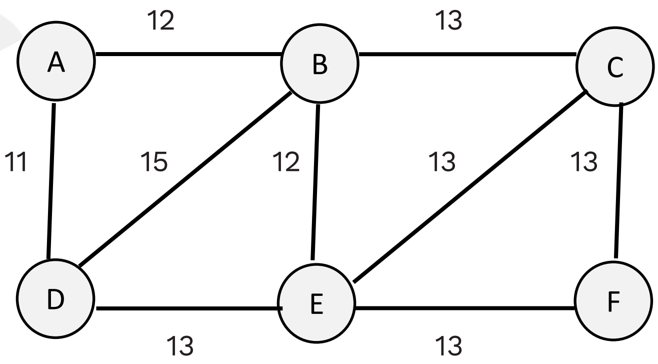
[GATE-2016]

Q.61 Suppose that minimum spanning tree of the following edge weighted graph contains the edges with weights x, y and z



What is the maximum value of x+y+z?

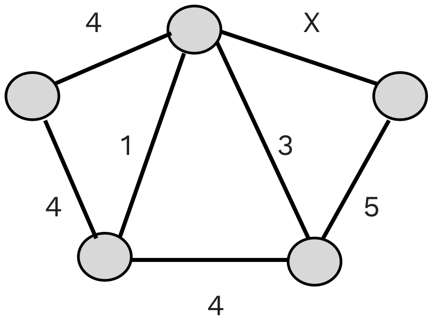
Q.62 Consider the following graph G.



Find the number of minimum cost spanning trees using Kruskal's algorithm or Prim's algorithm.

Q.63 Let G be connected undirected graph of 100 vertices and 300 edges. The weight of a minimum spanning tree of G is 500. When the weight of each edge of G is increase by five, the weight of a minimum spanning tree becomes__________.

Q.64 Consider the following undirected graph G:



Choose a value of x that will maximize the number of minimum weight spanning trees (MWSTs) of G. The number of MWSTs of G for this value of x is__________.

[GATE-2018]

Q.65 Let G = (V, E) be any connected undirected edge-weighted graph. The weights of the edges in E are positive and distinct. Consider the following statements:

I. Minimum Spanning Tree of G is always unique.

II. Shortest path between any two vertices of G is always unique.

Which of the above statements is/are necessarily true?

a) I only

b) II only

c) Both I nor II

d) Neither I nor II

[GATE-2017]

Q.66 A file contains characters a, e, i, o, u, s and t with frequencies 10, 15, 12, 3, 4, 13 and 1 respectively. If we use Huffman coding for data compression then the average code length will be.

a) $\frac{140}{58}$

b) $\frac{146}{58}$

c) $\frac{150}{58}$

d) $\frac{174}{58}$

[DRDO-2009]

Linked Answer for Q.67 & Q.68

Consider the following message:

aabbbbabccdddccccbbdd

Q.67 Find the number of bits required for Huffman encoding of the above message.

a) 30

b) 38

c) 42

d) 46

Q.68 If Huffman tree coded as left child with '0' and right child with '1' from every node then what is the decoded message for 110100

a) abc

b) bcd

c) acb

d) bda

Common Data for Q.69 & Q.70

Suppose the letters a, b, c, d, e have probabilities $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}$

Q.69 Which of the following is the Huffman code for the letters a, b, c, d, e?

a) 0, 10, 110, 1110, 1111

b) 11, 10, 011, 010, 001

c) 11, 10, 01, 001, 0001

d) 110, 100, 010, 000, 00

[2007 : 2 Marks]

Q.70 What is the average length of the correct answer to above question?

a) 3

b) 2. 1875

c) 2.25

d) 1.781

[GATE-2007]

Q.71 Suppose P, Q, R, S, T are sorted sequences having lengths 20, 30, 35, 50 respectively. They are to be merged into a single sequence by merging together two sequences at a time. The number of comparisons that will be needed in the worst case by the optimal algorithm for doing this is______.

Q.72 The average number of key comparisons done on a successful sequential search in list of length n is

a) log n

b) $\frac{n-1}{2}$

c) $\frac{n}{2}$

d) $\frac{n+1}{2}$

[GATE-1996]

Common Data for Q.73 & 74

We are given 9 tasks T_1, T_2,....T_9. The execution of each task required one unit of time. We can execute one task at a time. T_i has a profit P_i and a deadline d_i profit P_i is earned if the task is completed before the end of the d^{th} unit of time

Task	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
Profit	15	20	30	18	18	10	23	16	25
Deadline	7	2	5	3	4	5	2	7	3

Q.73 Are all tasks completed in the schedule that gives maximum profit?

a) All tasks are completed

b) T_1 and T_6 are left out

c) T_1 and T_8 are left out

d) T_4 and T_6 are left out

[GATE-2005]

Q.74 What is the maximum profit earned?

a) 147

b) 165

c) 167

d) 175

[GATE-2005]

Q.75 The following are the starting and ending times of activities A, B, C, D, E, F, G, and H respectively in chronological order:

“ a_s b_s c_s a_e d_s c_e e_s f_s b_e d_e g_s e_e f_e h_s g_e h_e”

Here, x_s denotes the starting time and x_e denotes the ending time of activity X. We need to schedule the activities in a set of rooms available to us. An activity can be scheduled in a room only if the room is reserved for the activity for its entire duration. What is the minimum number of rooms required?

a) 3

b) 4

c) 5

d) 6

[GATE-2003]

Q.76 Consider the Knapsack instance:

- Capacity of Knapsack is 15 and 17 objects
- Profits: $(P_1, P_2, \ldots\ldots, P_7) = (10, 5, 15, 7, 6, 18, 3)$
- Weights: $(w_1, w_2, \ldots\ldots, w_7) = (2, 3, 5, 7, 1, 4, 1)$
- Objects: $(x_1, x_2, \ldots\ldots, x_7)$

If Knapsack problem is solved using maximum profit per unit weight then find the object which is partially placed in the Knapsack

a) x_1

b) x_2

c) x_3

d) x_4

Q.77 Suppose you want to move from 0 to 100 on the number line. In each step, you either move right by a unit distance or you take a shortcut. A shortcut is simply a pre-specified pair of integers i, j with i < j. Given a shortcut i, j if you are at position i on the number line, you may directly move to j. Suppose T(k) denotes the smallest number of steps needed to move from k to 100. Suppose further that there is at most 1 shortcut involving any number, and in particular from 9 there is a shortcut to 15.

Let y and z be such that T(9) = 1 + min (T(y), T(z)). Then the value of the product yz is____.

[GATE-2014]

Q.78 Given an adjacency-list representation of directed graph G(V,E). Then how much time does it take to compute the out-degree of each vertex.

Ans: O(E + V)

Q.79 If the graph G(V, E) is represented using adjacency matrix then to find universal sink (in-degree is V- 1 and out- degree is 0).

Ans: O (V)

Q.80 An operator deleted (i) for a binary heap data structure is to be designed to delete the item in the i^{th} node. Assume that the heap is implemented in an array and i refers to the i^{th} index of the array. If the heap tree has depth d(number of edges on the path fromthe root to the farthest leaf), then what is the time complexity to re-fix the heap efficiently after the removal of the element?

a) O(1)

b) O(d) but not O(1)

c) $O(2^d)$ but not O(d)

d) $O(d2^d)$ but not O(2d)

[GATE-2016,]

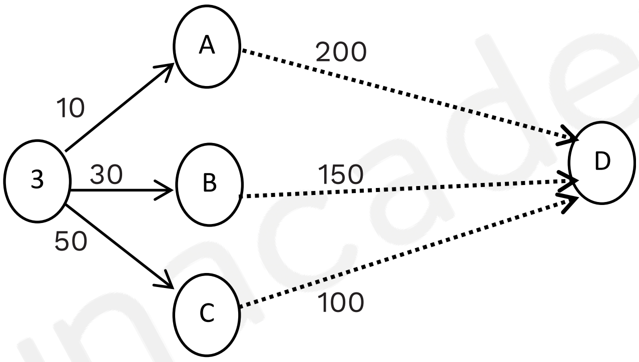
Ans: (b)

Dynamic Programming (Principle Of Optimality)

- [Used to solve optimisation problem]
- DP solves problem by using principle of optimality design technique.

Principle of optimality:-

Whatever the initial state, the decision cost over the remaining period must be optimal for the remaining problem, with the state resulting from the early decisions taken to be the initial condition.



$$\text{Cost }(S,D) = \min \left[\begin{array}{l} 10 + \text{cost }(A, D) \\ 30 + \text{cost }(B, D) \\ 50 + \text{cost }(C, D) \end{array} \right.$$

GreedyAlgorithm	Dynamic Programming
I) Uses static approach chooses 1^{st} decision irrespective of cost Of remaining decision	=> Uses principle of optimality
II) Greedy Algorithm always run in polynomial	=> Dynamic prog. Computes cost of all
Time Complexity (computes only one Feasible solution based on greedy technique)	feasible solution (Time Complexity may exponential)

III) Greedy Algorithm failed to solve some Optimisation problem	=> Dynamic programing used solve every optimisation problem.

Dynamic Programming	Brute Force Algorithm
I) Computes cost of every feasible solution Recursively by avoiding re-computations	=> Computes cost of every feasible solution && return optimal solution.
II) Because of avoiding re-computation Problem which has exponential feasible solution can be solved in polynomial Time Complexity.	=> if optimisation problem consist many exponential feasible solution then Brute Force Algorithm required exponential Time Complexity.
III) Mare space complexity to store result of every sub problem.	=> less space complexity

Dynamic Programming Examples

1. n^{th} fib number
2. Matrix chain problem ***
3. Optimal Binary search tree construction
4. Largest common Subsequence (LCS) problem
5. O/1 Knapsack problem
6. Sum of subset problem.
7. All pair shortest path Algorithm
8. Transitive closure of graph

} floyd warshal Algorithm

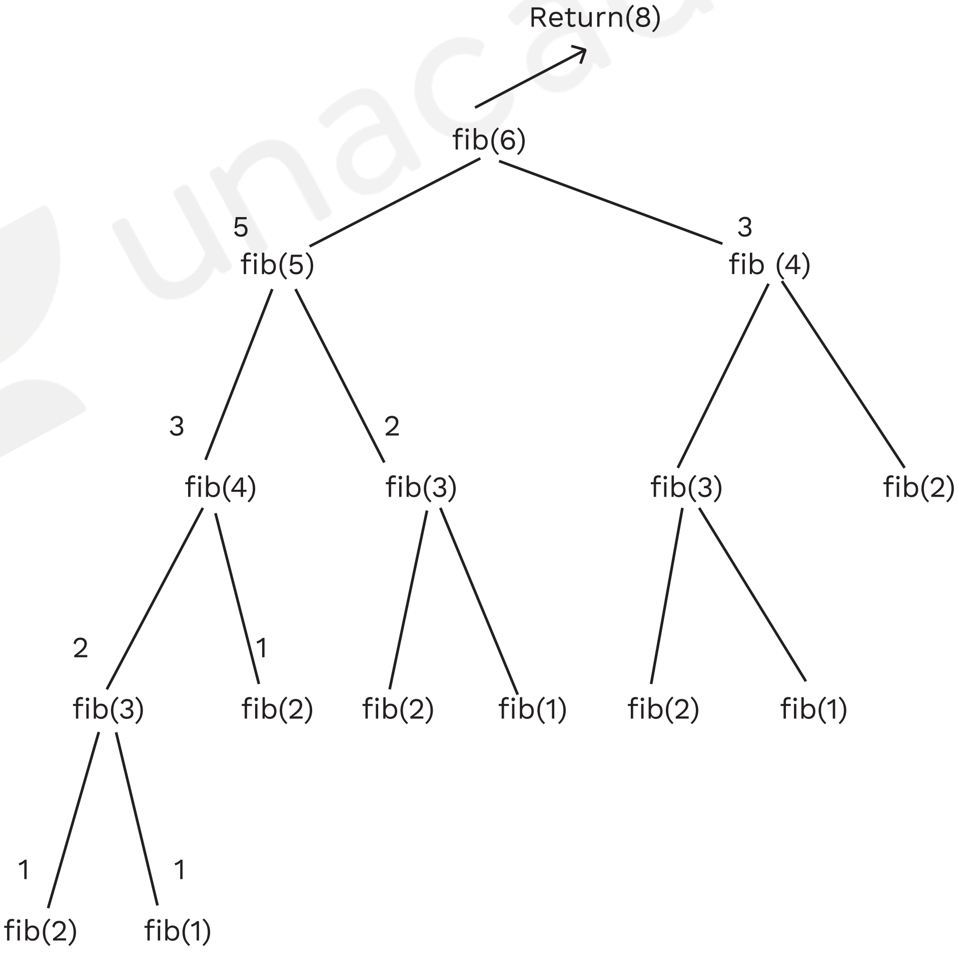
Time Complexity:O(n^3)

- Nth Fibinocci Number

1] Recurrence Relation :-

$$\text{fib (n)} \begin{cases} 1 & , \quad n = 1 \\ 1 & , \quad n = 2 \\ \text{Fib (n-1) + Fib (n-2)} & , \quad n > 2 \end{cases}$$

```
Algorithm Fib (n)
{
    if ( n ≤ 2 )
        return (1)
      else
        return (Fib (n-1) + Fib (n -2))
}
```



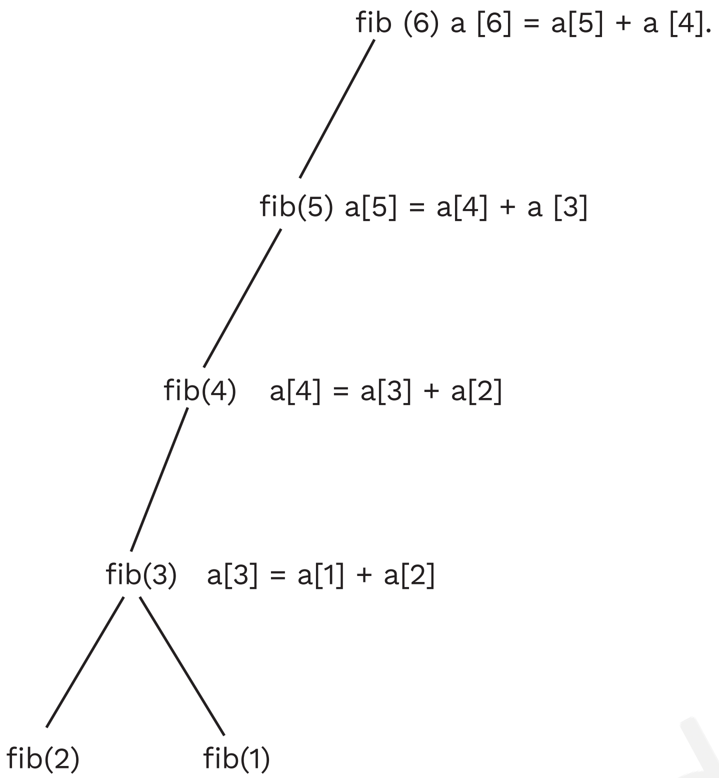
TIME COMPLEXITY: $O(2^n)$ || number fun calls almost CBT of O(n) levels.

SPACE COMPLEXITY : O(n)

2] DP nth Fib Num: -

```
Algorithm Fib (n)
    {
        for ( i = 1 ; i ≤ n ;i++)
    {
        a [i] = - 1 ; }
        if (n ≤ 2)
     {
        a[i] = 1
        return (1)
    }
else
{    if (a [n -1] = = -1)
    { a [ n – 1 ] = fib (n – 1 ) ; }
    if (a [ n – 2 ] = = - 1)
  { a [ n – 2 ] = fib (n – 2) ; }

a [n] = a [n -1] + a [n – 2 ]
return (a [n] )
}
}
```

	1	2	3	4	5	6
a	-1	-1	-1	-1	-1	-1


fib (6) a [6] = a[5] + a [4].
fib(5) a[5] = a[4] + a [3]
fib(4) a[4] = a[3] + a[2]
fib(3) a[3] = a[1] + a[2]
fib(2)
fib(1)

- n fun calls
- TIME COMPLEXITY: O(n)
- SPACE COMPLEXITY : array [1...n]

  Stack n

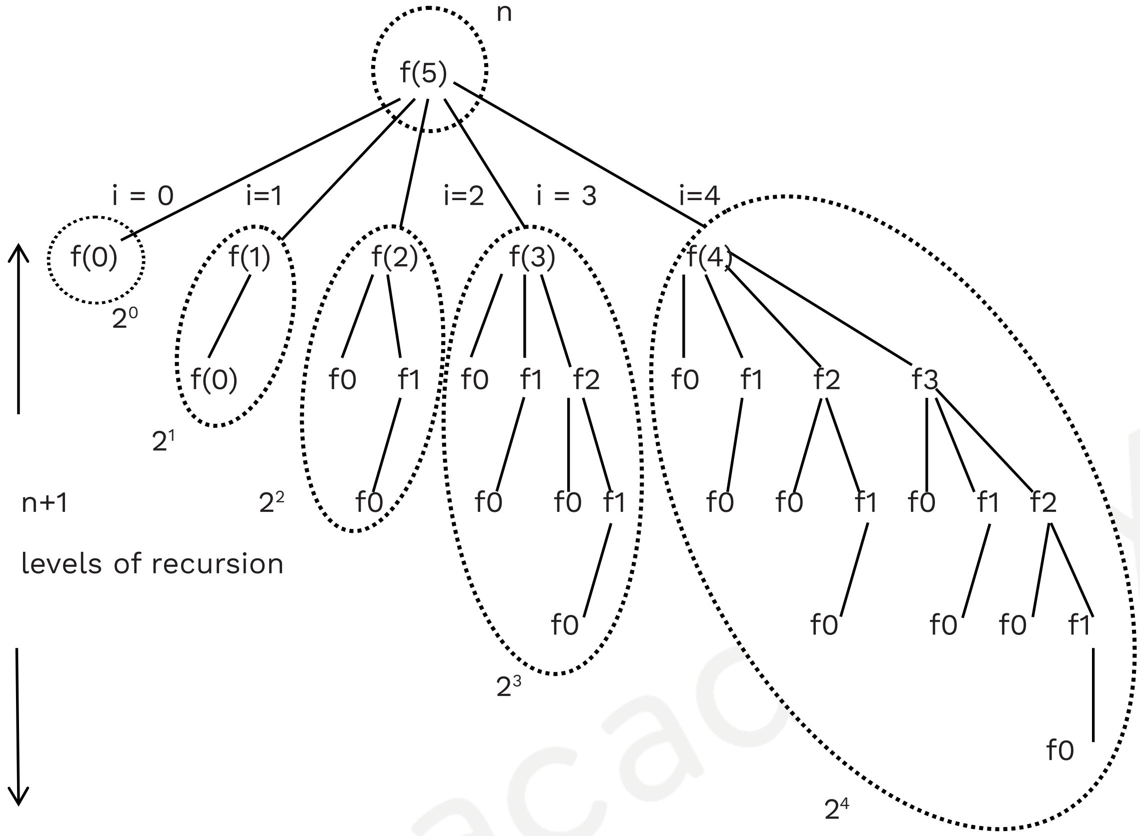
  $\theta(n)$

Q. foo(n)

```
{
    if (n = = 0)
        return (1)

    else

{

    sum = 0

    for ( i = 0 ; i < n ; i++)

{

    sum +  =  foo(2)      }

    return(sum))

}
```

TIME COMPLEXITY}

SPACE COMPLEXITY }



number of fun calls => $2^0 + 2^1 + 2^2 + \ldots\ldots + 2^{n-1} = 2^n$

TIME COMPLEXITY: $O(2^n)$

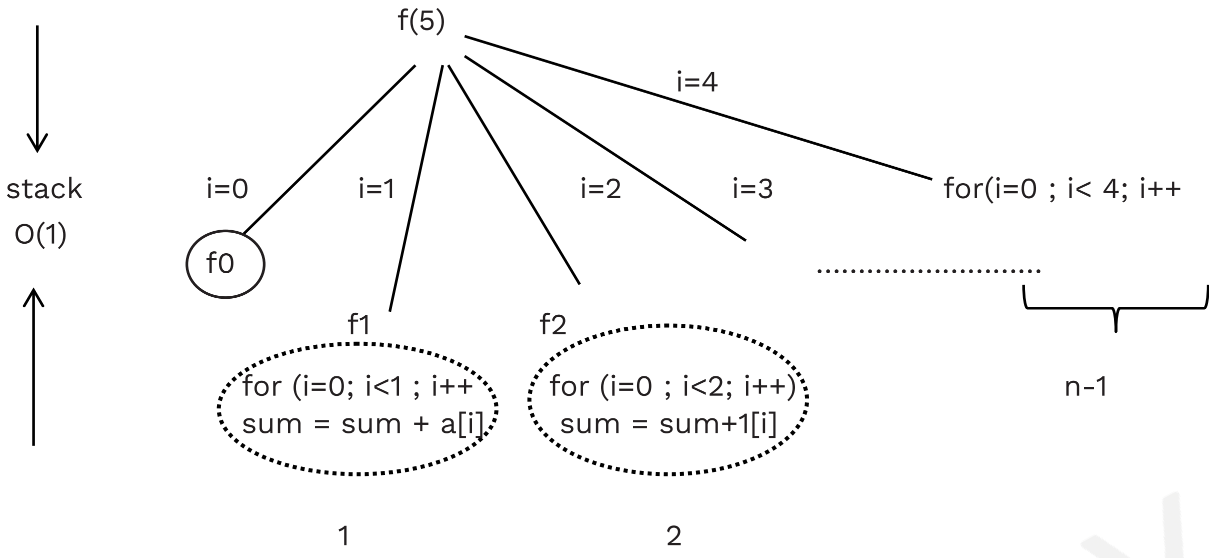
SPACE COMPLEXITY : O(n)

|| stack space

Q.62 Algorithm computer foo(n)

```
   {
      for ( i = 0 ;  i ≤ n ; i++)
        {
          a[i] = -1 }
        if ( n = = 0)
          {
           a [n] = 1
           return (1)
        }
     else
      {
         sum =0
     for ( Type something... = 0 ; i < n ; i++)
     {
       if (a[i] = = -1)
     {
        a[i] = foo (2) }
        sum + = a[i]
      }
    }
  }
}
```

	0	1	2	3	4	5
a	-1	-1	-1	-1	-1	-1



TIME COMPLEXITY=> [1 + 2 + 3 + ...(n – 1)] + (n + 1) = θ (n^2)

loop's cost fun calls

space complexity = O(n) [because of array]

DP Matrix Chain Problem:-

Given n matrices [$A_1 A_2$....A_n]

order set [P_0, P_1, P_2,.....P_n]

Minimum cost of multiplication of n matrices : c (1, n)

($A_1 A_2$....A_n)

$$\overset{P_0 \times P_k}{\longleftarrow X \longrightarrow} \quad \overset{P_k \times P_n}{\longleftarrow Y \longrightarrow}$$

$$C\ (1,n) \Rightarrow \quad (A_1 A_2 ....A_K) \quad (A_{K+1}........A_n)$$

$$\underbrace{P_0 \times P_1 .... P_{k-1} \times P_k}_{C[1,k]} \quad \underbrace{P_k \times P_{k+1} .....P_{n-1} \times P_n}_{C\ [K+1,\ n]}$$

$$C(1, n)=\underset{1 \le K < n}{\text{minimum}} \qquad c\,(1, k) + c\,(K+1, n) + P_0 \text{ x } P_k \text{ x } P_n$$

Recurrence Relation of Matrix chain problem using DP

$< A_i\, A_{i+1} \ldots\ldots\ldots\ldots A_j >$ matrices

$< P_{i-1}\, P_i\, P_{i+1} \ldots\ldots\ldots P_j >$ order set

C (i,j = $\quad 0 \quad$, if $\quad i = j$

Minimum {c(i, k) + c(K + 1, j) + $P_{i-1}\, P_k\, P_{j,}$ $\;$ i<k<= j-2

if i<j

n = 4 , A_1 $\quad A_2$ $\quad A_3$ $\quad A_4$ matrices

< $\quad P_0$ $\quad P_1$ $\quad P_2$ $\quad P_3\, P_4$> orders are

< 20, 40, 30, 10, 15 >

I] Minimum cost of multiplication of given 4 matrices

II] Parentherization of 4 matrices which is optimal cost of multiplication.

Solution.

$$c(1,4) = \text{Minimum} \left\{ \begin{array}{ll} & 0 \quad 18000 \quad 20.\ 40.\ 15 = 30000 \\ K = 1 & c_{11} + c_{24} + \quad P_0.\ P_1.\ P_4 \\ & \quad 24000 \quad 4500 \quad 20.\ 30.\ 15 \\ K = 2 & c_{12} + c_{34} + \quad P_0.\ P_2.\ P_4 \\ & \quad 2000 \quad 0 \quad 20.\ 10.\ 15 \\ K = 3 & c_{13} + c_{44} + \quad P_0.\ P_3.\ P_4 \end{array} \right.$$

$$\underbrace{(23000)}_{O(n)}$$

A [ ][]　　j　　$O(n^2)$

$O(n)$

1

	1	2	3	4
1	0	24000 k=1	20000 k=1	3000 k=3
2	X	0	12000 k=2	18000 k=3
3	X	X	0	4500 k=3
4	X	X	X	0

Final value j-2 = 3

j-i = 2

j-i = 11

initialization Bottom

0　45000　40 x 30 x15

j-i=2

c_{24} = minimum　$\left\{\begin{array}{ll} K=2 & c_{22} + c_{34} + P_1.\ P_2.\ P_4 \\ K=3 & c_{23} + c_{44} + P_1.\ P_3.\ P_4 \end{array}\right.$

[18000]

12000　0　40.10.15

0　12000　20.40.15

c_{13}=Minimum　$\left\{\begin{array}{ll} K=1 & c_{11} + c_{23} + P_0.\ P_1.\ P_3. \\ K=2 & c_{12} + c_{33} + P_0P_2P_3 \end{array}\right.$

[2000]

24000 + 0 +20x30x10

+6000

j-1=1

C(1,2)=　K=1　$\underbrace{c_{11}}_{0} + \underbrace{c_{22}}_{0} + \underbrace{P_0.\ P_1.P_2.}_{24000}$

C (2,3) =

C (3,4) =

$A_1 A_2 A_3 A_4$ minimum multiplication cost = 23000

K value = 3

OfC[1,4] $(A_1 A_2 A_3 A_4) A_4$

K value = 1 of

C [1,3]

$$\underbrace{A_1 (A_1 A_3)}$$

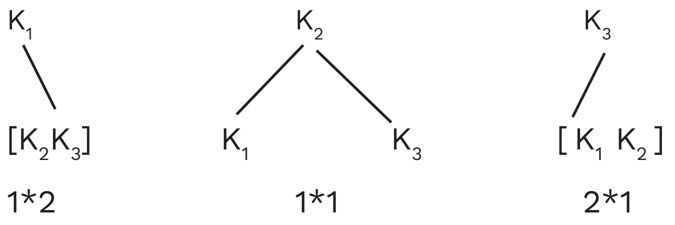
$$\downarrow$$

$$\underbrace{(A_1 (A_2 A_3) A_4}$$

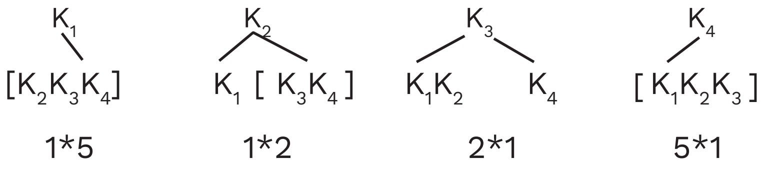
Optimal parentherization Time Complexity of Matrix chain problem is θ (n^3)

- Number of possible Binary Search tree for given n distinct keys.

Keys		Number of BST's
0		1
1	K_1	1
2	$K_1 < K_2$	2
3	$K_1 < K_2 < K_3$	5



4. $K_1 < K_2 < K_3 < K_4$ 14



T(n) = number of BST for n distinct keys

T(0) = 1

T(1) =1 K_1 K_2 K_3 K_n

T(n) = T(0). T(n-1) + T_1. T(n-2) + T_2(n-3) + ...+ T(n-1) T(0) kn

$$T(n) = \begin{cases} 1 \quad , & \text{for } n<=1 \\ \sum_{k=1}^{n} T(k-1)T(n-k); & \text{for } n>1 \end{cases}$$

Longest Common Subsequence (LCS):-

Subsequence:- “sequence of char” from string “x” which may not consecutive.

Eg. X: “abdacba”

Subsequence: adba

abaca

Common Subsequence :- Subsequence which is common for two strings x & y.

Eg. X : abadcdac
Y : aadabc } Common subsequence [aadac]

Common Subsequence of two string whose length is largest

X: aabcad
Y: adcab } aca
LCS :3

X String : n char } 2^n Possible Sequences

Y String : m char ← 2^m Possible sequences

Dynamic Programming LCS:-

String X: [$X_1 X_2 X_3 \ldots\ldots\ldots\ldots X_{n-1}\ X_n$]

String Y: [$Y_1 Y_2 Y_3 \ldots\ldots\ldots\ Y_{m-1}\ Y_m$]

LCS (n,m) : function to return largest length common subsequence of x & y strings with n, m char

$$\text{LCS }(n,m) = \begin{cases} 0 \quad , \quad n=0 \,||\, m=0 \\ 1+\text{LCS }(n-1,\ m-1)\ n > 0\ \&\&\ m > 0\ \&\& \\ \qquad\qquad X_n = Y_m \\ \text{Max} \begin{cases} \text{LCS }(n-1,\ m) \\ \qquad n > 0\ \&\&\ m > 0 \\ \text{LCS }(n,m-1,\ X_n \neq Y_m \end{cases} \end{cases}$$

$$\text{LCS }(i,j)= \begin{cases} 0 \quad , \quad i=0 \quad |\ |j=0 \\ 1 + \text{LCS }(i-1,\ j-1,\ i>0\ \&\&\ j> 0\ \&\& \\ \qquad\qquad X_i = Y_j \\ \text{Max} \begin{cases} \text{LCS}(i-1,j), & I > 0\ \&\&\ j > 0 \\ \text{LCS}(i,j-1) & X_i \neq Y_j \end{cases} \end{cases}$$

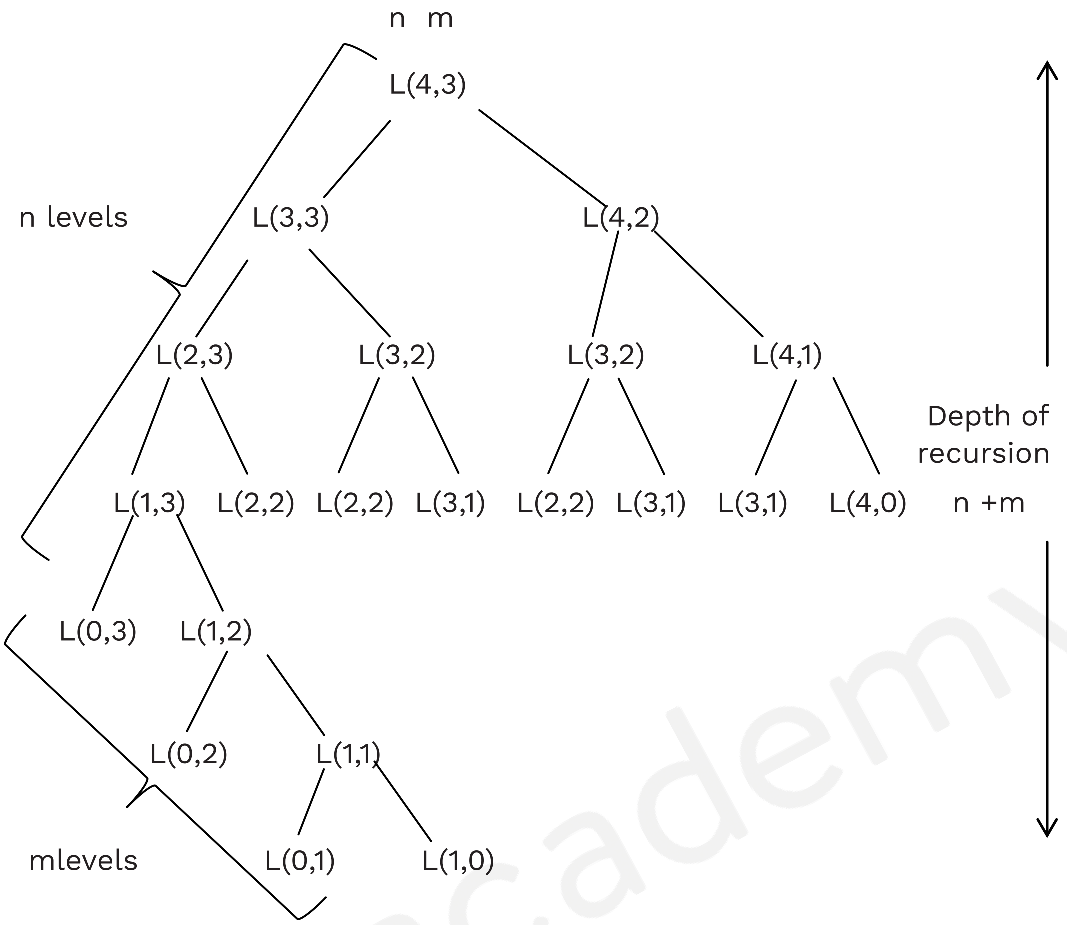
LCS (n,m) = 1+ LCS (n-1, m-1)

X : { abcda}

Y : { abada }

LCS (n-1,m-1)

$$\left.\begin{array}{ll} \overset{n}{\underset{n-1}{X : [abcda]}} & \text{LCS }(n,m-1) \\ \\ \overset{m-1}{\underset{N}{Y : [abadac]}} & \text{LCS }(n-1,\ m) \end{array}\right\} \text{Max}$$



Number of fun calls : 2^{n+m}

Time Complexity : θ $(2^n. 2^m)$

SC : θ (n + m)

Dynamic Programming implementation of LCS: - [DNA Test application]

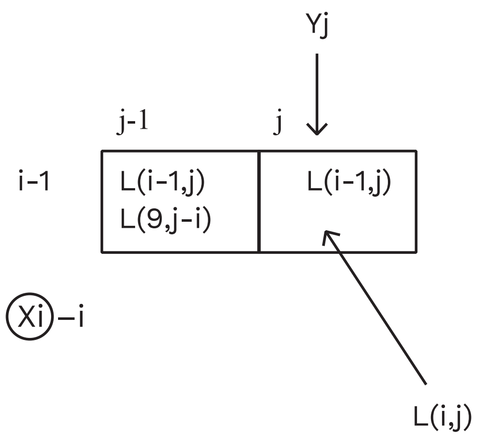
L (0.......n , 0 .....m) | | 2D array

	0	1	2	3 ....m
0	0	0	0	0......................0
1	0..			
2	0...			
:	:			
:	:			
n	0...>			

← Bottom

LCS (n,m) final value

1. First Row & First column initialize "O"
2. LCS array compute either Row major order or column major order



Example: X : [a b c ad]

Y : [b a b c a d]

String Y

{ ←, ↑ } means char not matches

			b	(a)	(b)	(c)	(a)	(d)
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
	(a) 1	0	← 0	1 ↖	1 ←	1 ←	1 ↖	1 ←
string	(b) 2	0	1 ↖	1 ←	2 ↖	2 ←	2 ←	2 ←
	(c)3	0	1 ↑	1 ←	2 ↑	3 ↖	3 ←	3 ←
X	(d) 4	0	1 ↑	2 ↖	2 ←	3 ↑	4 ↖	4 ←
	(e) 5	0	1 ↑	2 ↑	2 ←	3 ↑	4 ↑	5 ←

LCS (5,6)

LCS (5,6) = 5

```
        Algorithm LCS  ( X [n] , Y [m], n , m) {
            for (i = 0 ;  i ≤ n; i++   ⎫  O(n)
                L [i] [0] = 0;         ⎭

            for (i = 0;  i ≤ m ;i++ )  ⎫  O(m)
                L [0] [j] = 0;         ⎭

            for (i = 1 ; i ≤ n ; i++

            {
                for (j = 1 ; j ≤ m ; j++)
                {
                    if ( Xi = = Yj)
                    {
                      L[i] [j] = 1 + L (i=1) (j-1) ;
                      P [i] [j] = “↖“.
                    }
                    else
n       m           {        if (L[i] [j-1] ≥ L[i-1] [j] )
times   times                {

        L [i] [j] = L[i] [j-1]
        P [i] [j] = “←”
      }
        else
      {
        L [i] [j] = L [i-1] [j]
          P [i] [j] = “↑”
        }
      }
    }
  }
}
```

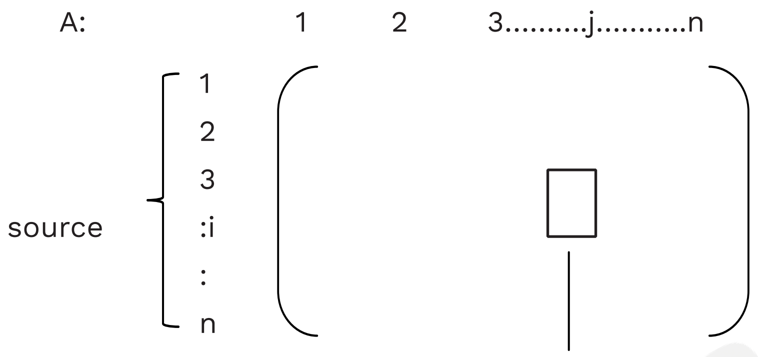
Time Complexity of LCS using DP : θ (n *m)

SC of LCS using DP : θ (n * m)

All pair shortest path Algorithm:-

[Floyd warshal Algorithm]

Given graph G(n,e). All pair shortest path Algorithm should determine shortest path distance b/w every pair of vertices



Minimum distance b/w vertex i to vertex j.

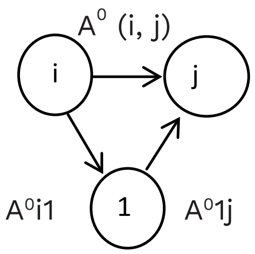
Cost [1...n, 1....n] Given

$$\text{Cost}[i, j] = \begin{cases} \text{edge cost if } (i, j) \in \text{edge} \\ 0 \quad \text{if} i=j \\ \infty \quad \text{if } (i, j) \neq \text{edge} \end{cases}$$

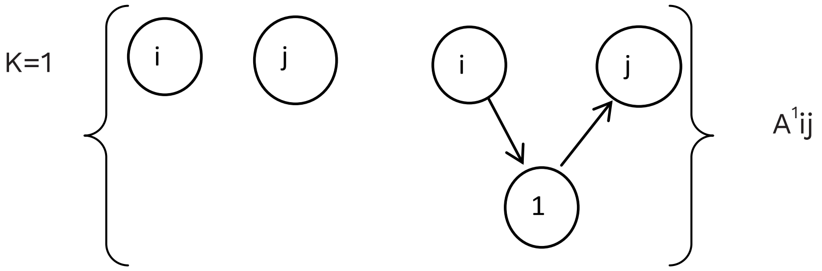
DP All Pairs shortest path Algorithm:-

$A^0(i, j)$ = cost (i, j) | |Initialization

1]

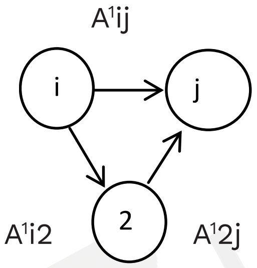


A^1 (i, j) = {minimum , A^0 (i, j) A^0 (i, 1) + A^0(1, j)

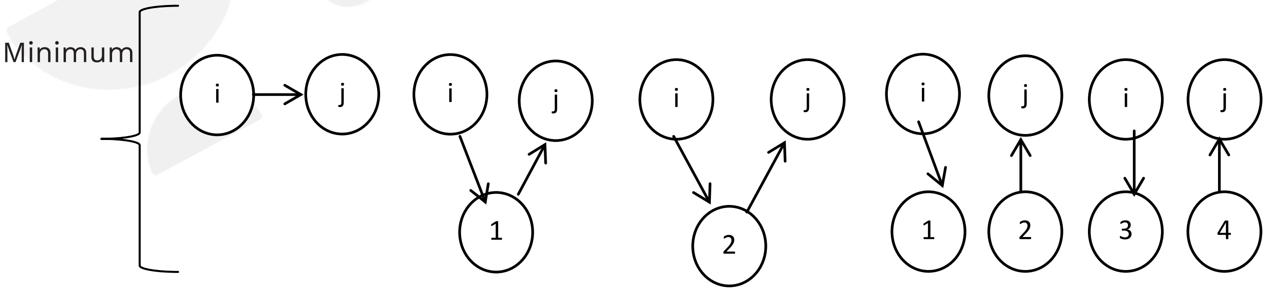


A^1(i ,j) shortest path from i to j going through vertex 1.

2]

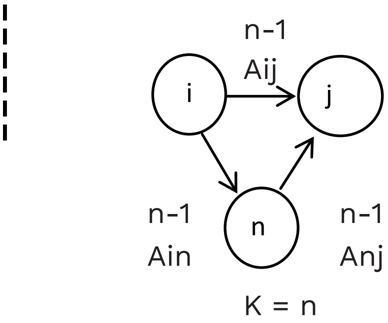


A^2((i, j) = minimum { A^1(i, j), A^1(i, 2), + A^1(2, j),



A^2ij shortest path cost from i to j going through h (1) and (2)
or

n]



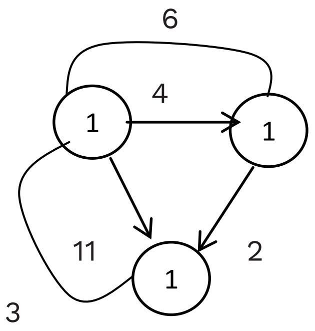
$$A^n(i, j) = \text{Minimum} \qquad \text{min} \begin{cases} A^{n-1}(i, j) \\ A^{n-1}(i, n) + A^{n-1}(n, j) \end{cases}$$

$$\downarrow$$

Shortest path from i to j going through h (1) $\underset{\text{or}}{\underline{\text{and}}}$ (2) $\underset{\text{or}}{\underline{\text{and}}}$ (3) .... $\underset{\text{or}}{\underline{\text{and}}}$(n)

$$A^n i, j = \text{Minimum} \quad \{A^{K-1}ij, \; A^{K-1}ik + A^{K-1}kj\}$$

$$1 \leq k \leq n$$



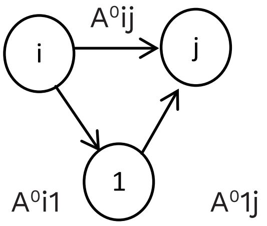
$$\text{Cost} \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix}$$

Solution

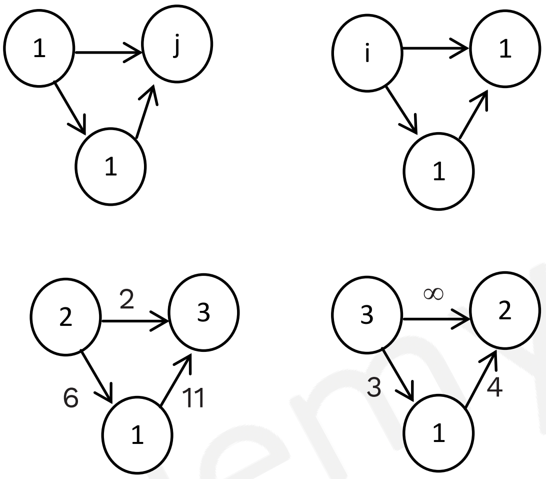
$$O(n^2) \left\{ A^0 = \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix} \right.$$

Destination

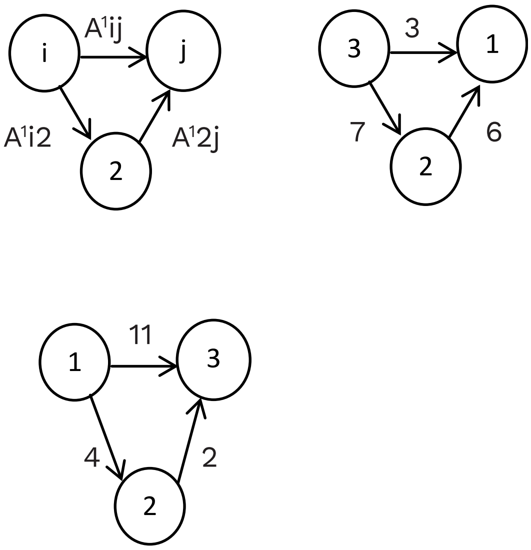
$$O(n^2) \begin{cases} A^1 \;= \\ \text{source} \\ \end{cases} \begin{pmatrix} & 1 & 2 & 3 \\ 1 & 0 & 4 & 11 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{pmatrix}$$



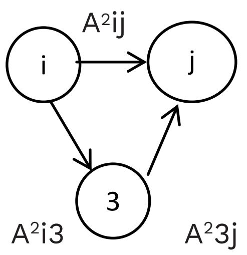
n times



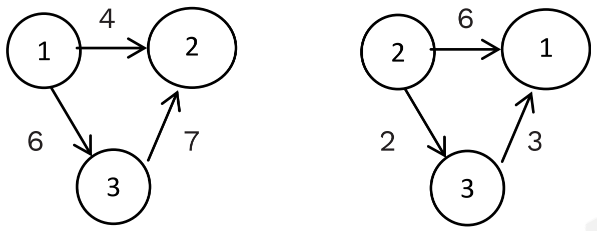
$$O(n^2) \begin{cases} A^2 \;= \\ \\ \end{cases} \begin{pmatrix} & 1 & 2 & 3 \\ 1 & 0 & 4 & 6 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{pmatrix}$$



$$O(n^2) \left\{ A^3 = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 6 \\ 2 & 5 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{array} \right.$$



All pair shortest path distances



All pair SPA (cost [ ] [ ] , n) || Floyd warshall Algorithm

```
θ(n²)  {
           for (i = 1 ; i ≤ n ;i++)
         {  for (j = 1; j ≤ n ; j++)
           Aij = Cij
       }

for (k = 1 ; k ≤ n ; k++)
       {
           for (i = 1; i ≤ n ; i++)
θ(n³)          for (j = 1 ; j ≤ n ; j++)
                   Aij = minimum { Aij, Aik + Akj}
        }      return (A [ ] [])
       }
```

Time Complexity Of Floyd Warshal = $\theta(n^3)$

Graph Traversal Algorithm

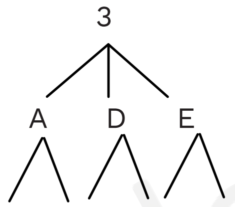
Graph Traversal Algorithm:-

1) Breath First Search
2) Depth First Search

Breath First Search:-

[Also called level order traversal]

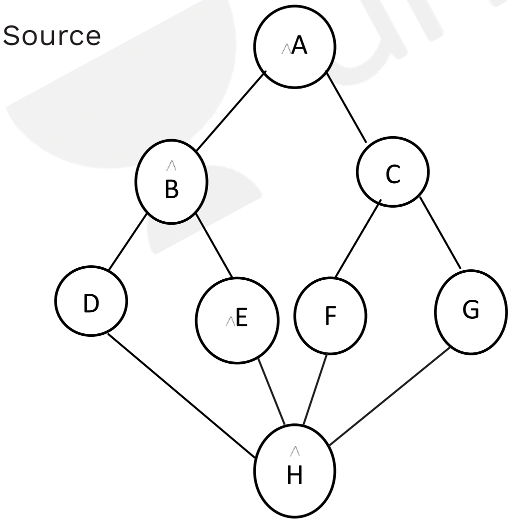
graph G(n,e) && source (s)



Idea: 1) Visit Source

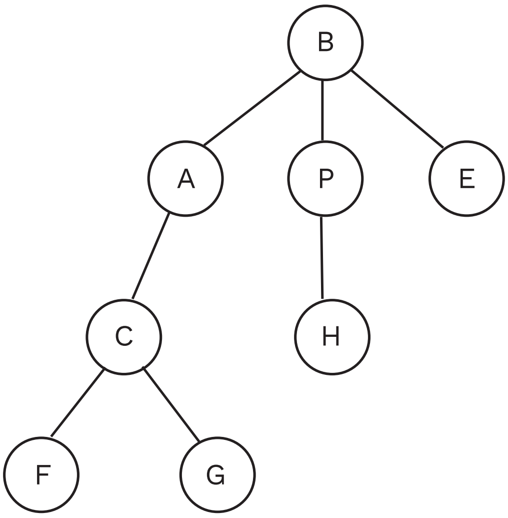
2) Visit all vertices with one edge distance from source

3) Visit all vertices with two edge distance from source and soon.



[Undirected graph]

=>



$\underbrace{\text{B A D E C H F G}}_{\text{BFS order}}$

$\underbrace{\hspace{8em}}_{\text{BFS spanning Tree}}$

↑ deleted

A	D	E	C	H	F	G

Queue	F1F0	B adj	:	A,D,E
		A adj	:	B,C
		D adj	:	B,H
		E adj	:	B,H
		C adj	:	A F G
		H adj	:	D E F G
		F adj	:	C H
		G adj	:	C H

Unexplored Vertex:- Unvisited vertex

[White Vertex]

Exploring Vertex:- Vertex (V) is visited but their adjacent vertices may not visited

[GRAY Vertex]

Explored Vertex: vertex (V) is visited && their adjacent vertices also visited

[Black Vertex]

Example:-

Bs Be As Ae Ds De Es Ee Cs Ce Hs He Fs Fe Gs Ge

In BFS vertex starting exploration order same vertex ending exploration.

Algorithm BFS (G, n, e) s)

```
      {
        for (i = 1 ; i ≤ n ; i++)
O(n)      { visited [Vi] = 0}
```

```
                visit [s] = 1
                while (True)
                  {  || s starts exploring
                     V is set of Adj of s => deg (s) or O(n)
                     for (each vertex of v set )
$\theta(e)|\theta(n^2)$        {
                      if (visited [v] = = 0)
                       {
times                      visited [v] =1
           deg(s)                 Insert (Queue (Q) , V}
$\theta(e)$                    }
                      }
                     // s finished exploration

            if (Queue (Q) is empty) return ;
            else
            s = Delete (Queue (Q) )
          }
        }
```

Time Complexity of BFS Algorithm:-

1] Using Adj list graph

$\theta(n) + \theta\ (e) = \theta\ (\ n + e)$

2] Using Adj matrix graph

$\theta(n) + \theta\ (n^2) = \theta\ (n^2)$

Space complexity of BFS Algorithm

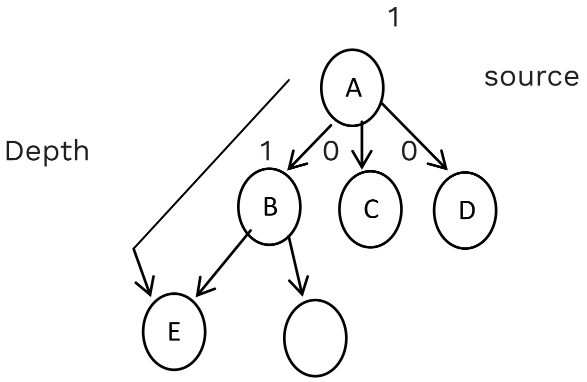
Visited [1...n] : n

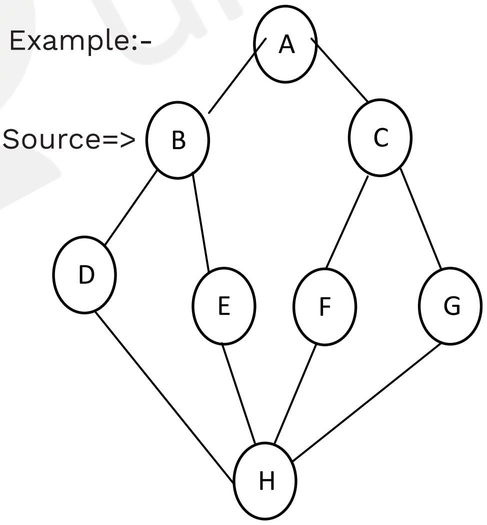
Queue : $\underline{n}$
$\theta(n)$

Depth First Search:

[Visit Vertices based on depth]

1. Visit Source
2. Visit one of Adjacency vertex (V) of source
3. Visit one of Adjacency vertex (W) of vertex (V) and so on.





Vset

DFS(B) :{A,D,E}

↘ DFS(A) :{B,C}

↓

DFS(c) :{A,F,G}

↓

DFS (F) : { C, H}

↓

DFS (H) : {D E F G}

↓

DFS (D) : {B, H}

DFS (E) = { B, H}

DFS (G) = { C,H}

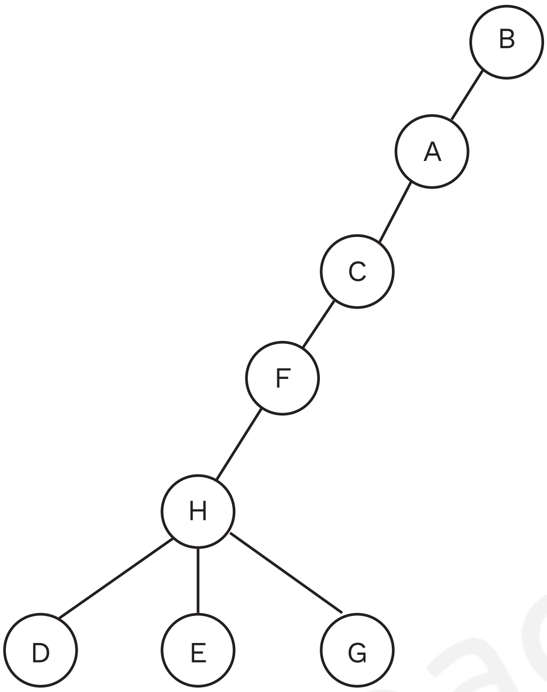
Bs As Cs Fs Hs Ds De Es Ee Gs Ge He Fe Ce Ae Be

⟶

- DFS sequence (vertex visited order)

  B A C F H D E G

- DFS spanning Tree



Total 6 level

```
Algorithm Depth first search ( G, n, e, s)
    {
      for (i = 1 ;  i ≤ n  ; i++)                    color [Vi] = “w”
          { visited [Vi] = 0 ; } →

    DFS (s)
     {  color [s] =“G”
      visited [s] =1                  deg(s) orO(n)
      V is set of Adj of s →
     for (each vertex of V set )
       {
n funcall   deg (s) if (visited [V] = = 0)
                 DFS[V]
```

```
            }  color [s] = “B”
        }
    }
```

Time Complexity of DFS Algorithm:-

a) Using Adj list graph

$\theta(n) + \theta(e) = \theta (n + e)$

b) Using Adj matrix graph

$\theta(n) + \theta (n^2) = \theta(n^2)$

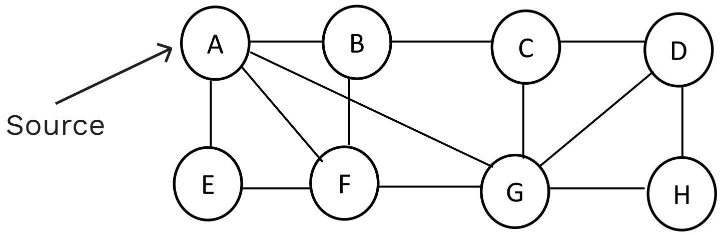
- **Space complexity of DFS Algorithm:-**

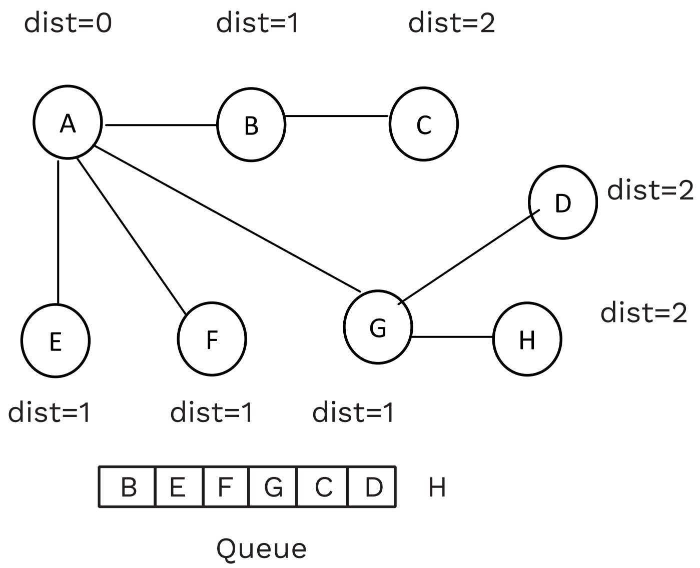
  Visited [1...n] : O(n) stack Data Structure : at most n stack entries [Depth of recursion]

  SC of DFS : $\theta(n)$

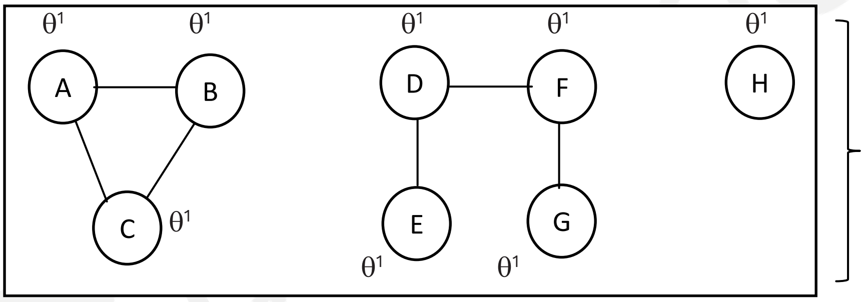
Applications by using BFS | DFS | algorithm

(1) BFS algorithm can be used as single source shortest path algorithm for unweighted graph. => $\theta(n + e)$ or $\theta (n^2)$





(2) DFS/BFS Algorithm can used to find number of connected components of undirected graph (or) to test undirected graph G is single connected component or not?

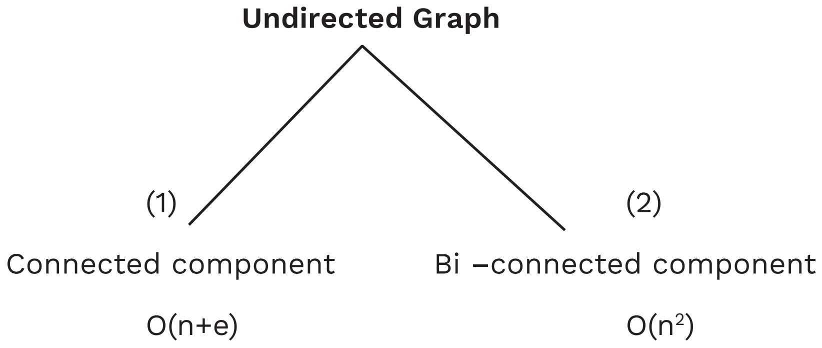


Graph “G” of 7 vertices

```
Algorithm sample (G, n, e)
    {  count = 0
         visited [Vi] = = 0 ;
      for ( i = 1 ; i ≤ n ;i++))
       {
              if (visited [Vi] = = 0)
           {
              DFS [vi]
              count++;
           }
       }
    }  return(count)
```

(number of DFS calls = number of connected components)

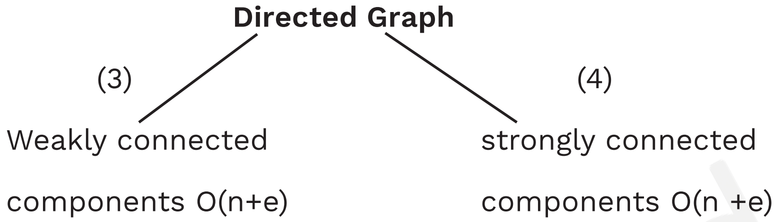
Time Complexity: θ (n + e) or θ (n^2)



Undirected Graph

(1) Connected component O(n+e)

(2) Bi –connected component O(n^2)



Directed Graph

(3) Weakly connected components O(n+e)

(4) strongly connected components O(n +e)

[BFS | DFS used to test]

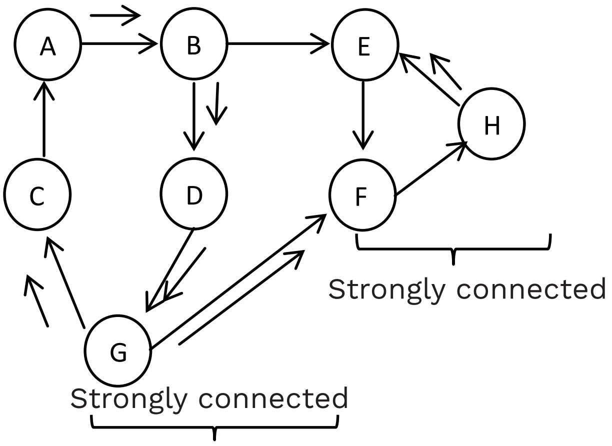
(3) DFS / BFS can used to test cycle in undirected graph: θ(n +e) |O(n^2)

(4) BFS / DFS can used to test cycle in directed graph. θ(n +e) /O(n^2)

BFS/DFS can be used to count number of strongly connected components of directed graph.

Strongly connected component: for every pair of vertices (u,v) there must be path from u to v && v to u.

Eg. G:



Procedure

1. Call DFS for graph G && start exploration. Push completion order of vertices in stack (X) θ(n + e)

2. Compute GT (Transpose graph)

   θ(e)

3. Call DFS for GT based on unvisited top of stack (x) as source.

   θ(n + e)

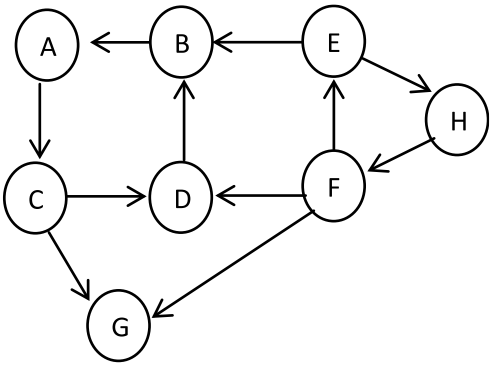
   || number of DFS call to traverse GT = number of strongly components

1] DFS

~~A~~
~~B~~
~~D~~
~~G~~
~~F~~
~~H~~
~~E~~
~~C~~

Stack (X)

2]



DFS (A) :

A, C, D, B, G

DFS (f) : F, E, H

Time Complexity: θ(n + e)

Weakly connected : G => All Vertices
or GT => All Vertices

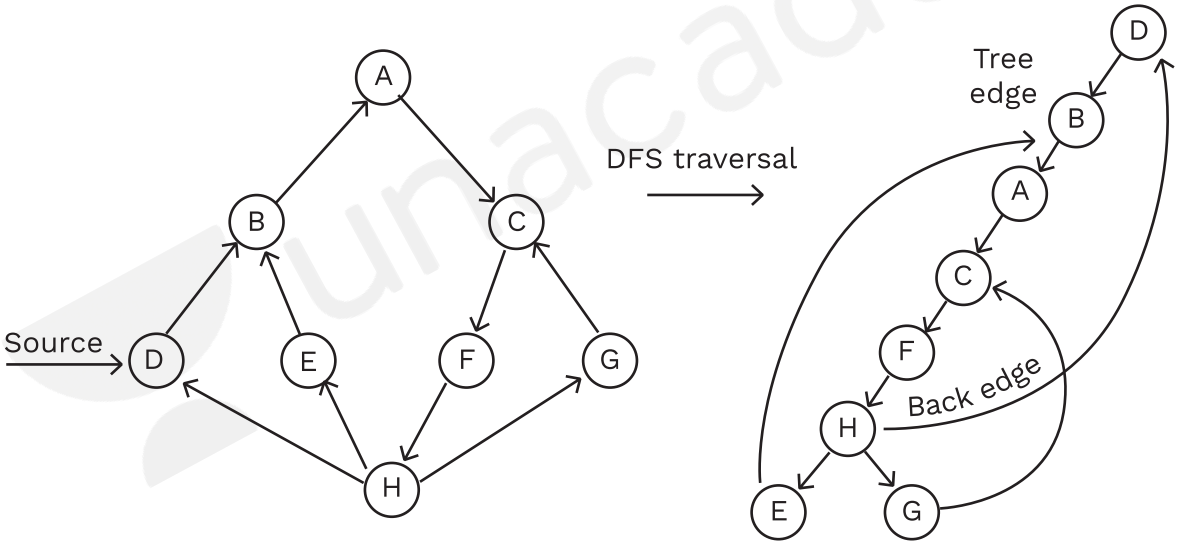
Undirected Graphs

DFS/ BFS Traversal

- Tree Edges

  [DFS/ BFS Spanning tree edges]

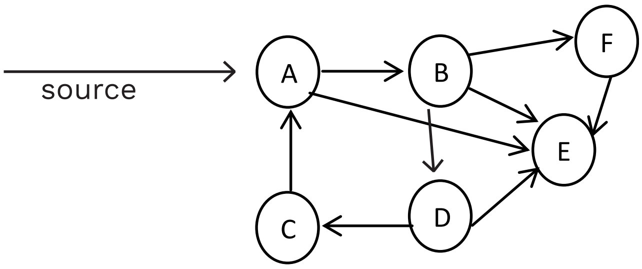
- Back Edges [number Back edges] = e – (n-1) = e-n+1 [other edges of graph except tree edges].



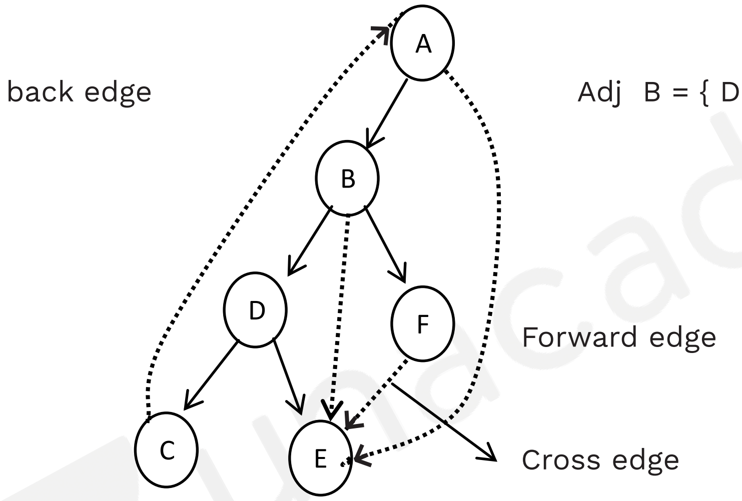
Directed Graph

BFS/DFS traversal divides edges into

- Tree edges [DFS/BFS spanning tree edges]
- Back edges [descendent to parent]
- Cross edge [edge: one path of Tree to other path]
- Forward edge [parent to descendent]



Source vertex [A] [adj visited lexicographic DFS traversal order].



Adj B = { D E F}

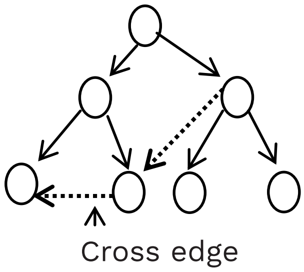
Number of tree edges = 5

Number of cross edge = 1

Number of back edge = 1

Number of forward edge = 2

⇨In directed graph if there exist back edge of BFS/DFS traversal then graph is cyclic.



Sorting Algorithm

Sorting Algorithm:-

1. Comparison based sorting Algorithms

	Sorting Algorithm	Best Case (BC)	Average Case (AC)	Worst Case ((WC)	Stable sorting	Inplace sorting
1.	Quick Sort	θ(nlogn)	θ(nlog n)	θ(n^2)	No	Yes
2.	Merge Sort	θ(nlogn)	θ(n log n)	θ(nlogn)	Yes	No
3.	Heap Sort	θ(nlog n)	θ(n log n)	θ(nlogn)	No	Yes
4.	Bubble Sort	θ(n)	θ(n^2)	θ(n^2)	Yes	Yes
5.	Selection Sort	θ(n^2)	θ(n^2)	θ(n^2)	No	Yes
6.	Insertion Sort	θ(n)	θ(n^2)	θ(n^2)	Yes	Yes

2. Non Comparison based sorting Algorithm

1. Counting Sort

Time Complexity = θ(n + k)

[stable but not inplace sorting]

2. Radix Sort (bucket sort)

Time Complexity = θ (d.n)

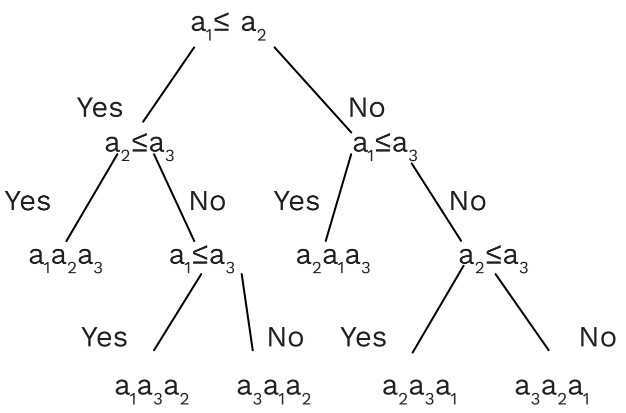
[stable sorting and inplace sorting algorithm].

Worst case comparisons required for any comparison based sorting algorithm to sort array of elements is Ω (nlogn).

[a_1 a_2 a_3] elements

Sort given 3 elements using comparison based sorting :-

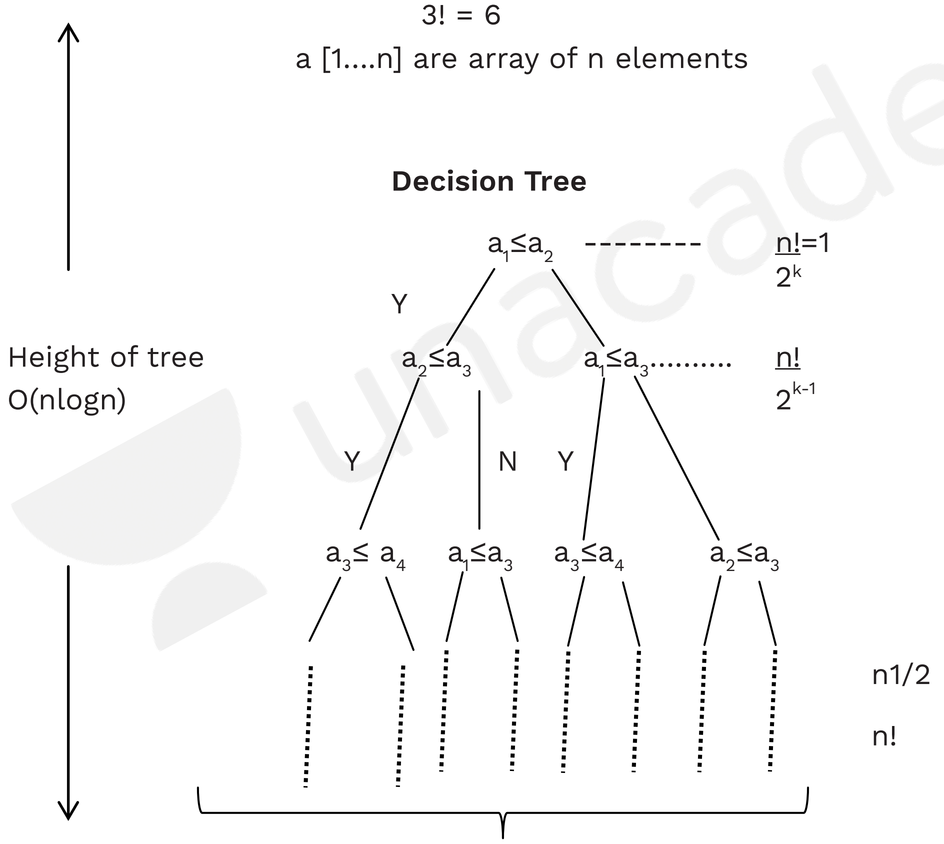
Decision Tree



3! = 6

a [1....n] are array of n elements

Decision Tree



n! possible results [n! leaf nodes]

$\frac{n!}{2^k}=1$ $2^k = n!$

K = logn!= n logn

- To sort n elements atleast one comparison required for each level of decision. In this worst case number of comparison Ω (nlogn))

  In this best case no. of comparison = (n-1)

Bubble Sorting Algorithm:-

(stable && inplace sorting)

Idea: Compare && exchange adjacency elements

Procedure:- a [1...n] array of n elements

Pass: compare a[j] , a [j + 1]

swap if a [j] > a [j + 1]

for (j = 1 to n-1)

// a [n] correct position

Pass 2 : compare a[j] , a [j+1]

swap if a [j] > a[j+1]

for j = 1 to n-2

// a [n-1, n] correct position

⋮ ⋮ ⋮

Pass (n-1): compare a[j] , a[j+1]

swap if a [j] > a [j+1]

for j = 1 to 1

// sorted array

Q. a

1	2	3	4	5	6
20	15	18	15	12	10

Pass 1. 15 20 20 20 20 20

18 15 12 10

Pass 1 :

15	18	15	12	10	20

15 18 18 18

12 10

Pass 2.

15	15	12	10	18	20

Pass2

Pass 5

```
Algorithm Bubble Sort (a,n
{
    for ( i = 1 ) ; i ≤ n-1 ; i++)
     {
        for ( j = 1 ; j ≤ n-2 , j++)
     {
          if (a [j] > a[j+1]
          swap (a[j] , a[j+1] ;
        }
    }
return (a [1...n] )
}
```

Time Complexity

	Comparison	**swaps**
Pass1	n–1	[0 to n-1]
Pass2	n–2	[0 to n-2]
Pass3	n–3	[0 to n-3]
⋮		
Pass(n-1)	1	[0 to 1]
	$\frac{n(n-1)}{2}$ +	$\left\{ 0 \text{ to } \frac{n\ (n-1)}{2} \right\}$

Time comparison = $\frac{n(n-1)}{2}$ + 0 to $\frac{n(n-1)}{2}$

= $\theta\ (n^2)$ in all cases.

Selection Sort:- (Inplace but not stable)

Idea : Find position of minimum element from a [i] to a[n] and swap with a [i] where

i = 1 (pass)

i = 2 (pass)

i = 3 (pass)

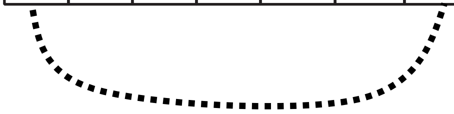
: : :

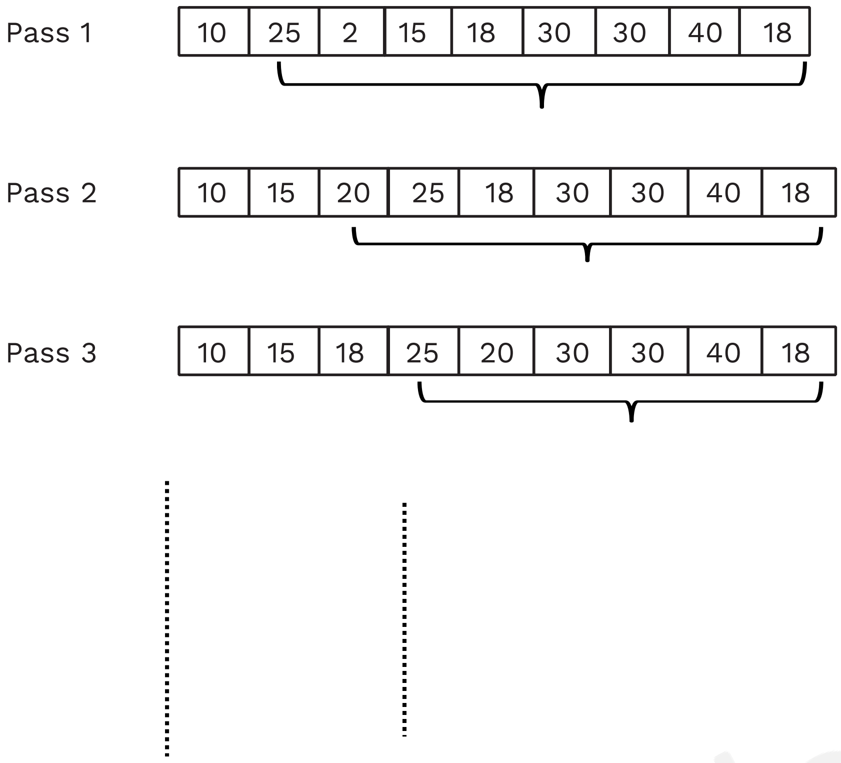
: : :

i = n-1 (pass n-1)

Eg. a

1	2	3	4	5	6	7	8	9
30	25	20	15	18	30	10	40	18





Pass 1

10	25	2	15	18	30	30	40	18

Pass 2

10	15	20	25	18	30	30	40	18

Pass 3

10	15	18	25	20	30	30	40	18

```
Algorithm selection sort (a,n)

{
   for ( i = 1 ; i ≤ n-1 ; i++)
   {
       minpos = i
     for (j=i+1; j ≤ n ; j++)

     {
        if (a [j] < a[min pos] )
         {
             minpos = j }
         }
            swap (a [i] , a[minpos])

         }
             return (a [1...n])

  }
```

Time Complexity

	Number of comp		swaps
Pass 1	n –1		1
Pass 2	n –2		1
Pass 3	n –3		1
:	:		:
:	:		:
Passn-1	1		1
Time Comp	$\frac{n(n-1)}{2}$	+	(n – 1)

= θ (n^2) in All cases.

Insertion Sort Algorithm:- (inplace && stable)

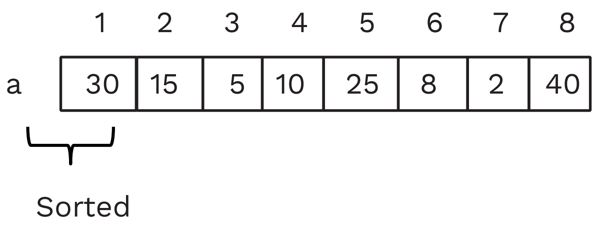
Idea :- Place a[i] element into correct position of sorted part of array which is a [1...n – 1].
i => 2 to n

Pass 1 : Place a[2] into correct position of sorted parts of array a[i...1].

Pass 2 : Place a[3] into correct position of sorted part of array a[1....2]

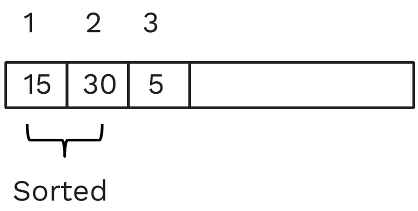
⋮

Pass n-1 : Place a[n] into correct position of sorted part of array a [1.... n-1]

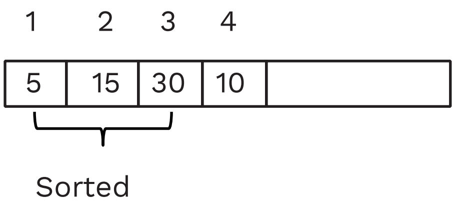


	1	2	3	4	5	6	7	8
a	30	15	5	10	25	8	2	40

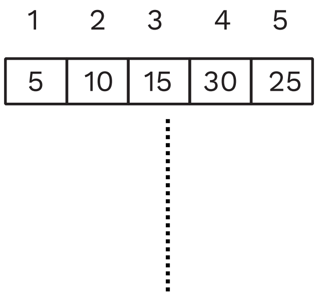
Pass1.



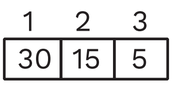
Pass2.



Pass 3.

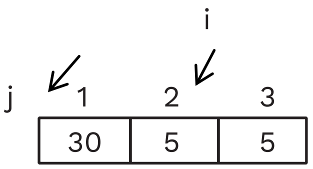


```
Algorithm Insertion sort (a, n)
    {
        for (i =1 ; i ≤ n-1 ; i++)
         {
              item = a[i+1] ; j=i
            while (j ≥ 1 && a[i] ;
                 j--;

                 a[j+1] = item
    } return (a[1...n] )
}
```



Pass 1

item = 15 j = X^0



Pass 2

5 15 30

item = 5

j =2

1

0

Time Complexity

Pass	Minimum && Minimum Comparison array shift	Maximum && Maximum comparison array	Average && average comparison array shift
Pass 1	1 && 0	1 && 1	1 && 1
Pass 2	1 && 0	2 && 2	2/2 && 2/2
Pass 3	1 && 0	3 && 3	3/2 && 3/2
:	:	:	:
:	:	:	n-1/2 && n-1/2
Pass(n+1)	$\underbrace{1 \quad \&\& \quad 0}_{}$	n- 1 && n – 1	n-1 && n – 1
	(n-1) comparison	$\frac{n(n-1)}{2} + \frac{n(n-1)}{2}$	4 + 4
			= θ(n^2)
Best Case time complexity θ(n)		Worst Case time complexity θ(n2)	Average case time complexity

Q8. Divide and conquer

a) Heap Sort --- O(nlogn)

b) Quick Sort ---O(n^2)

c) Merge Sort --- O(n logn)

d) Insertion Sort --- O(n)

Q. Insertion Sort Worst Case Time Complexity $O(n^2)$ uses linear search to identity position of element in sorted part of array.

If Binary search used to identity position of element in sorted part of array.

a) remain $\theta(n^2)$

Binary insertion sort

	Maximum Comparison	&&	Maximum array shift
Pass 1	1	&&	1
Pass 2	log2	&&	2
Pass 3	log3	&&	3
:	:	:	:
:	:	:	:
n-1	log(n-1)	&&	n 1

$$\theta\ (n\log n) + \frac{n(n-1)}{2}$$

Worst Case Time Complexity: $\theta(n^2)$.

Counting Sort Algorithm:-

[stable but not inplace sorting algorithm]

Pre-requirement: a[1...n] array of m integer && each element in array must be in range of 0 to k.

Example: Array of 10 integers whose range of values 0 to 8

	1	2	3	4	5	6	7	8	9	10
a	7	5	8	4	2	5	8	0	4	7

	0	1	2	3	4	5	6	7	8
count	0	0	0	0	0	0	0	0	0
	1		1		2	2		2	2

1. Initialize Count [0...K] by 0's
2. Increment Count by 1 count [a[i]] + 1 ;

   count [a[i]] = count [a [i]] + 1 ;
3. Count [i] = count [i-1] + count [i].

	0	1	2	3	4	5	6	7	8		
count	~~1~~	1	2	2	~~4~~	6	6	~~8~~	10	9	8
	0		1		~~3~~ 2	~~5~~	4	~~7~~			

	1	2	3	4	5	6	7	8	9	10	
b	0	2	4	4	5	5	7	7	8_a	8_b	} sorted

4. b [count [a[i]]] = a[i]

   count [a[i]] = count [a[i]] – 1 } i = n to 1

```
Algorithm counting sort (a, n)
 {
    for ( i = 0 ; i ≤ k ; i++)
        count [i]=0;                              $\theta(k)$
 for ( i = 1 ; i ≤ n ; i++)
        count [a[i] ] = count [a[i] ]+1           $\theta(n)$
 for ( i = 1 ; i ≤ k ; i++)
        count a[i] = count [i-1] + count[i];      $\theta(k)$
 for ( i = 1 ; i ≥ 1 ; i- -)
        {                                         $\theta(n)$
          b [count [a[i] ] ] = a[i] ;
          count [a[i] ] = count [a[i] ] – 1
        }
           return (b [1...n]
        }
```

$\theta(n + k)$ Time Complexity

Time Complexity of counting sort : θ (n + k)

if k ≤ n time complexity of counting sort θ(n)

SC of counting sort : θ(n + k)

{ for count array and b (auxiliary array)

Radix Sort Algorithm [bucket sorting]

[Bucket Sorting] [stable and inplace sorting algorithm]

Radix (G) : G different symbols used to represent any number system whose values [0 , 1, 2 ,..........., G-1]

Decimal numbers { 0, 1, 2, ...9} symbols (G = 10)

Hexadecimal numbers { 0, 1, 2, ...9, A, B,C,D,E,F } [G =16]

Radix Sort:-

a [1...n] array of n integers && radix of numbers :G && number of digits of each number : d

1. Define G Queue Data Structure && whose addresses 0 , 1, ...,G-1
2. Place each element of array into Queue based Kth least significant digit of numbers. && Retrieve elements from Queue 0 to Queue (G-1) and store in array “a”
3. Repeat (2) for K = 1 to d.

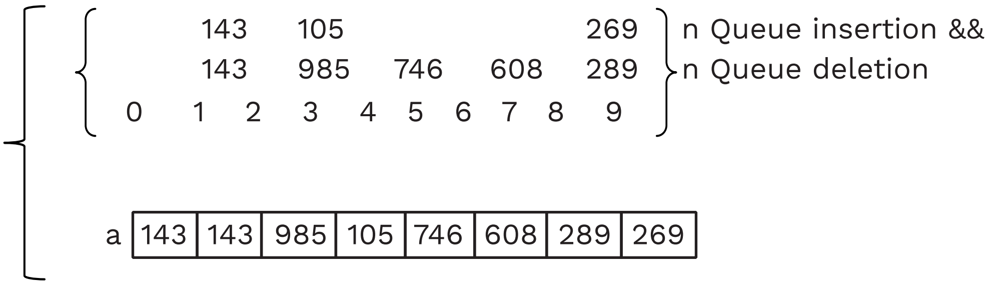
a[1...8] array of 8 decimal no. && each element ‘3’ digit number

	1	2	3	4	5	6	7	8
a	746	289	608	143	269	985	143	105

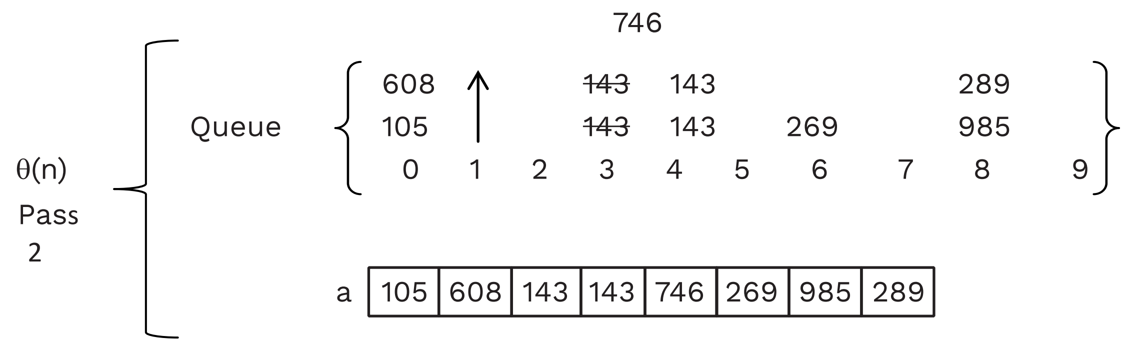
Queue

Pass 1

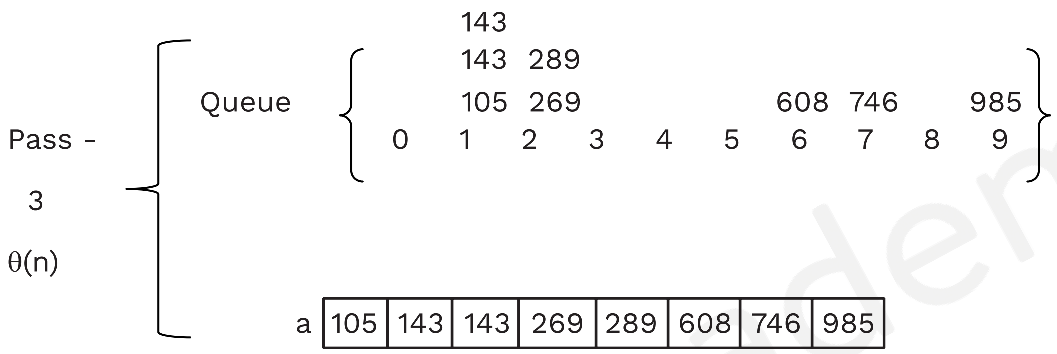
θ(n)



a	143	143	985	105	746	608	289	269



a	105	608	143	143	746	269	985	289



a	105	143	143	269	289	608	746	985

Time Complexity of Radix Sort : θ (d.n)

d passes and each pass {n Queue insert operation, as well as n Queue delete operation}

⇨ Radix Sort time complexity is θ(n) if d is constant

Que: Using which algorithm an array of n elements in the range [1...n^3] can be sorted in O(n) time?

Method 1

Radix sort : Apply radix sort array

Time Complexity :θ (d.n) θ(nlogn)

element X =n^3

number digits of X : $\log_{10}n^3 = 3.\log_{10}n$

Method 2

1. for each element of array convert decimal NS to radix-n NS.

   element X : $(n^3)_{10}$= (3 digits)n

$\theta(n)$ $\left\{ \begin{array}{c} \text{d : number of digits for X} \\ \text{using radix -n} \end{array} \right\}$ $\log_n n^3$= 3 [constant]

$\theta((n)$ { 2. Apply Radix sort for above array.

$\theta(n)$ { 3. For each element array convert radix –n number to radix-10 number

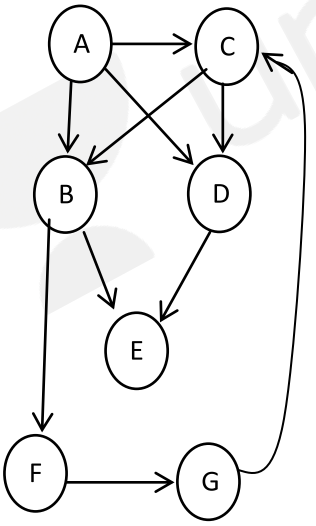
Time Complexity : $\theta(n)$

Q. Find tree , back and cross edges in below graph.

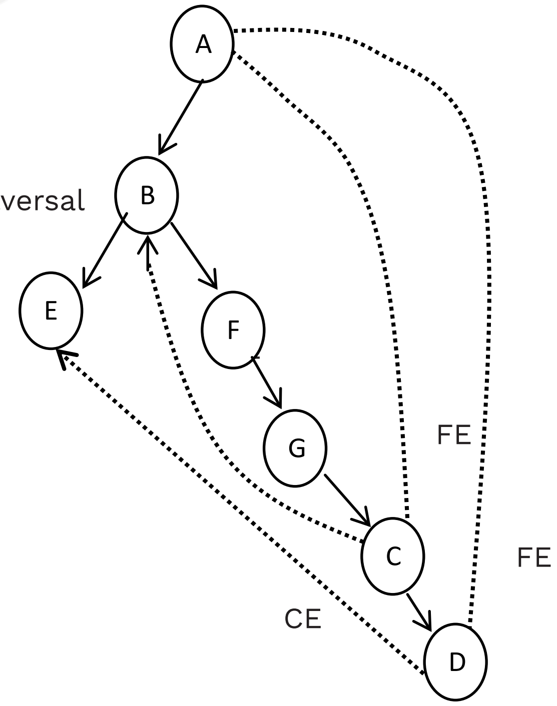
BE: back edge

FE: forward edge

CE: cross edge



After DFS traversal
=>



- Assume A source
- DFS applied
- Adj are visited lexicographic ascending order

Practice Problem:

Q.1 The pseudo code for insertion sort is presented as a procedure called **Insertion-sort,** which takes array A[1...n] as parameter containing sequence of n elements to be sorted. Find the missing statements at x and Y respectively.

Insertion-sort (A)

```
{   For j ← 2 to length [A]
    {       Key ←A [ j]
            i = $\boxed{x}$
            While (i > 0 && A[i] > key)
        {   A [i + 1] ← A[i]
            i = $\boxed{Y}$  }
            A[i + 1 ] ← key.
    }
}
```

(a)

X	Y
j – 1	i + 1

(b)

X	Y
j + 1	i + 1

(c)

X	Y
j – 1	i - 1

(d)

X	Y
i – 1	i - 1

Ans: (c)

Q.2 Given n integers in the range of 0 to k we want to preprocess the input in such a way that to any query about how many of the n-integers fall in the range a...b is O(1). Then what will the preprocess time?

Ans: O(n)

Q.3 Let G be a undirected graph on n-nodes. Any two of the following statements implies the 3rd is it True/False?

1. G is connected
2. G don't have cycle
3. G contain n – 1 edges

Ans: True

Q.4 Let G = (V, E) be s simple undirected graph and s be a particular vertex in it called the source. For x ϵ V, let d(x) denote the shortest distance in G from s to x. A breadth first search (BFS) is performed starting at s. Let T be the resultant BFS tree. If (u, v) is an edge of G that is not in T, then which one of the following CANNOT be the value of d(u)-d(v)?

a) –1

b) 0

c) 1

d) 2

Ans: (d)

Q.5 The number of ways in which the number 1, 2, 3, 4, 6, 7 can be inserted in an empty binary search tree, such that the resulting tree has height 6, is____.

Note: The height of a tree with a single node is 0).

[GATE-2016,]

And: (64)

Q.6 Given a sequence of n-real numbers a_1, a_2 a_3... a_n then to find continuous subsequence a_i a_{i+1} a_{i+2} a_j. Such that its sum is maximum. How much time the above problem will take if you use dynamic program?

Ans: O(n)

Q.7 Let A_1, A_2, A_3, and A_4b e four matrices of dimensions 10 x 5, 5 x 20, 20 x 10, and 10 x 5 respectively. The minimum number of scalar multiplications required to find the product $A_1 A_2 A_3 A_4$ using the basic matrix multiplication method is______.

[GATE-2016]

Ans: (1500)

Q.8 Consider a sequence A of length n which is sorted except for one item that appears out of order. Which of the following can sort the sequence in O(n) time?

a) Heapsort

b) Quicksort

c) Merge sort

d) Insertion sort

[DRDO-2008]

Q.9 Consider the following algorithm for searching for a given number x in an unsorted array A[1..n] having n distinct values:

1. Choose an I uniformly at random from [1..n]

2. If A[i] = x then Stop else Goto 1;

Assuming that x is present A, what is the expected number of comparisons made by the algorithm before it terminates?

a) n

b) n - 1

c) 2n

d) n/2

[GATE-2002]

Q.10 The usual $\theta(n^2)$ implementation of insertion sort to sort an array uses linear search to identify the position where an element is to be inserted into the already sorted part of the array. If, instead, we use binary search to identify the position, the worst case running time will

a) remain $\theta(n^2)$

b) become $\theta(n (\log n)^2)$

c) become θ (n log n)

d) become θ (n)

[GATE-2003]

Q.11 Let A be a sequence of 8 distinct integers sorted in ascending order. How many distinct pairs of sequences, B and C are there such that (i) each is sorted in ascending order, (ii) B has 5 and C has 3 elements, and (iii) the result of merging B and C gives A?

a) 2

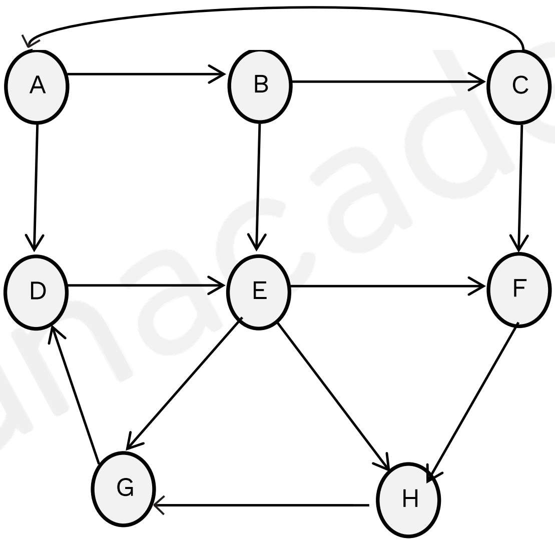
b) 30

c) 56

d) 256

[GATE-2003]

Q.12 Find the number of strong components in the following graph.



Q.13 Consider an undirected unweighted graph G. Let a breath-first traversal of G be done starting from a node r. Let d(r, u) and d(r, v) be the lengths of the shortest paths from r to u and v respectively in G. If u is visited before v during the breadth-first traversal, which of the following statements is correct?

a) d(r, u) < d(r, v)

b) d(r, u) > d(r, v)

c) d(r, u) ≤ d(r, v)

d) None of these

[GATE-2001]

Q.14 Let G be an undirected graph. Consider a depth-first traversal of G, and let T be the resulting depth-first search tree. Let u be a vertex in G and let v be the first new (unvisited) vertex visited after visiting u in the traversal. Which of the following statements is always true?

a) {u, v} must be an edge in G, and u is a descendant of v in T

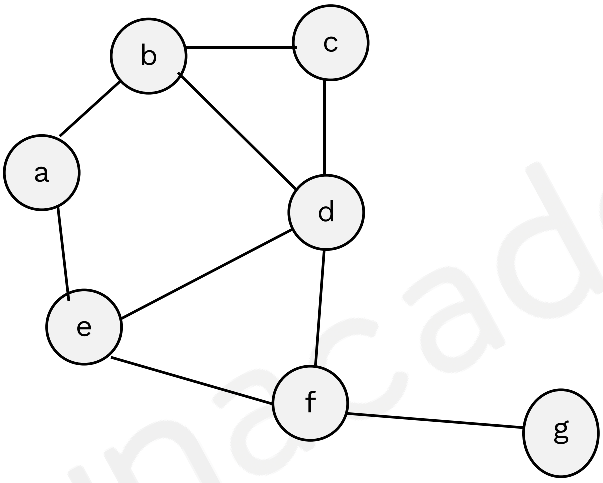
b) {u, v} must be an edge in G, and v is a descendant of u in T

c) If {u, v} is not an edge in G then u is a leaf in T

d) If {u, v} is not an edge in G then u and v must have the same parent in T

[GATE-2000]

Q.15 Consider the following graph:



A possible Depth First Search (DFS) sequence for the above graph is

a) d, e, b, a, c, f, g

b) b, a, e, c, d, f, g

c) d, b, a, e, f, c, g

d) b, c, d, e, f, g, a

[DRDO-2009]

Q.16 The most efficient algorithm for finding the number of connected components in an undirected graph on n vertices and m edges has time complexity.

a) $\Theta(n)$

b) $\Theta(m)$

c) $\Theta(m + n)$

d) $\Theta(m\ n)$

[GATE-2007]

Q.17 Which of the following statement is false?

a) The depth of any DFS (Depth First Search) tree rooted at a vertex is at least as much as the depth of any BFS tree rooted at the same vertex

b) If all edges in a graph have distinct weight then the shortest path between two vertices is unique

c) For a directed graph, the absence of back edges in a DFS tree means graph has no cycle

d) BFS takes $O(V^2)$ time in a graph G(V, E) if graph is represented with an adjacency matrix.

Q.18 Consider the tree arcs of a BFS traversal from a source node w in an weighted, connected, undirected graph with equal edge weights. The tree T formed by the tree arcs is a data structure for computing.

a) The shortest path between every pair of vertices

b) The shortest path from w to every vertex in the graph

c) The shortest path from w to only those nodes that are leaves of T

d) The longest path in the graph

Q.19 Let G e a simple undirected graph. Let T_D be a depth first search tree of G. Let T_B be a breadth. First search tree of G. Consider the following statements:

I. No edge of G is a cross edge with respect to T_D. (A cross edge in G is between two nodes neither of which is an ancestor of the other inT_D).

II. For every edge (u, v) of G, if u is at depth i and v is at depth j in $T_{B,}$ then | I – j| = 1

Which of the statements above must necessarily be true?

a) I only

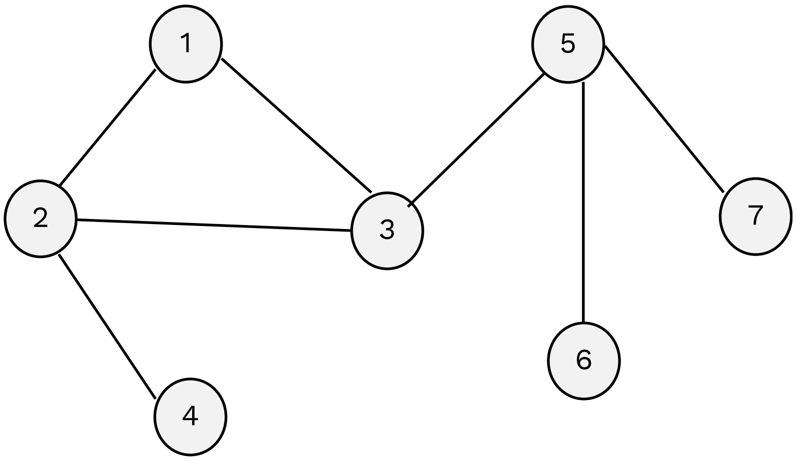
b) II only

c) Both I and II

d) Neither I nor II

[GATE-2018]

Q.20 The number of articulation points of the following graph is



a) 0

b) 1

c) 2

d) 3

[GATE-1999]

Q.21 A depth-first search is performed on a directed acyclic graph. Let d[u] denoted the time at which vertex u is visited for the first time and f[u] the time at which the DFS call to the vertex u terminates. Which of the following statements is always true for all edges (u, v) in the graph?

a) d[u] < d[v]

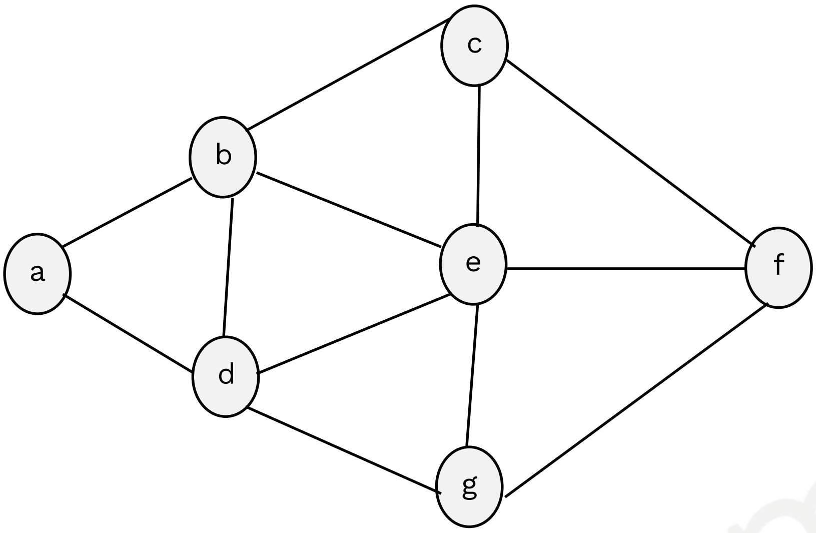
b) d[u] < f[v]

c) f[u] < f[v]

d) f[u] > f[v]

[GATE-2007]

Q.22 Consider the following sequence of nodes for the undirected graph given below:



1. a b e f d g c
2. a b e f c g d
3. a d g e b c f
4. a d b c g e f

A Depth First Search (DFS) is started at node a. The nodes are listed in the order they are first visited. Which all of the above is (are) possible output(s)?

a) 1 and 3 only

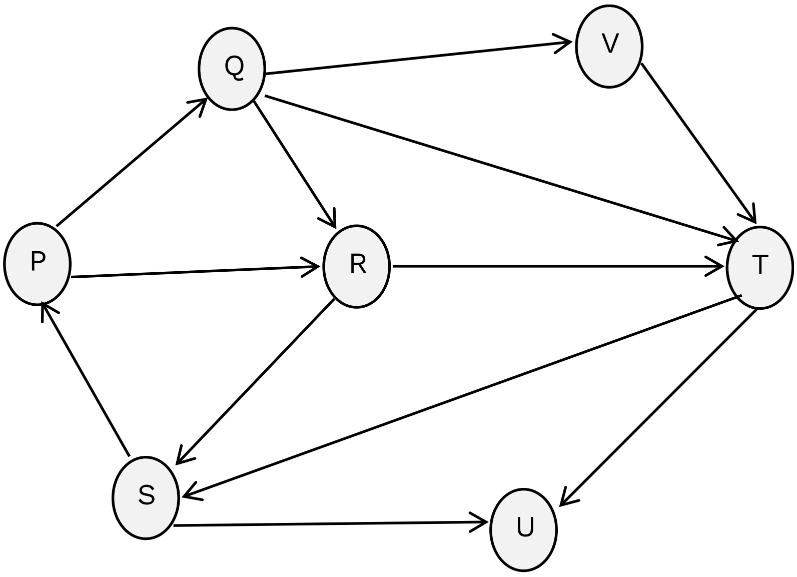
b) 2 and 3 only

c) 2, 3 and 4 only

d) 1, 2 and 3 only

[GATE-2008]

Q.23 Which of the following is the correct decomposition of the directed graph given below into its strongly connected components



a) {P, Q, R, S} {T}, {U}, {V}

b) {P, Q, R, S, T, V,} {U}

c) {P, Q, S, T, V}, {R}, {U}

d {P, Q, R, S, T, U, V}

[GATE-2006]

Q.24 A sort method is said to be stable if the relative order of keys is the same after the sort as it was before the sort. In which of the following pairs both sorting algorithms are stable?

a) Quick-sort and Insertion-sort

b) Insertion-sort and Bubble-sort

c) Quick-sort and Heap-sort

d) Quick-sort and Bubble-sort

[DRDO-2009]

Q.25 Give the correct matching for the following pairs:

List-I	**List-II**
A. O(log n)	P. Selection
B. O(n)	Q. Insertion sort
C. O (n log n)	R. Binary search
D. O(n^2)	S. Merge sort

Codes:

(a) A – R B – P C – Q D – S

(b) A – R B – P C – S D – Q

(c) A – P B – R C – S D – Q

(d) A – P B – S C – R D – Q

[GATE-1998]

Q.26 The tightest lower bound on the number of comparisons, in the worst case, for comparison-based sorting is

a) $O(n)$

b) $O(n^2)$

c) $\Omega(n \log n)$

d) $\Omega(n \log^2 n)$

Q.27 Assume that the algorithms considered here sort the input sequences in ascending order.

If the input is already in ascending order, which of the following are TRUE?

I. Quick sort runs in $\Theta(n^2)$ time

II. Bubble sort runs in $\Theta(n^2)$ time

III. Merge sort runs in $\Theta(n)$ time

IV. Insertion sort runs in $\Theta(n)$ time

a) I and II only

b) I and III only

c) II and IV only

d) I and IV only

[GATE-2016]

Q.28 Using which algorithm an array of n-elements in the range [1... n^3] will be sorted using

O(n) time?

a) Merge sort

b) Quick sort

c) Radix sort

d) Insertion sort

Q.29 If we use Radix Sort to sort n integers in the range ($n^{k/12}$, n^k], for some k > 0 which is independent of n, the time taken would be

a) Θ(n)

b) Θ(kn)

c) Θ(n log n)

d) Θ(n^2)

[GATE-2008]

Q.30 The most appropriate matching for the following pairs:

X: Depth first search	1: Heap
Y: Breadth-first search	2: Queue
Z: Sorting	3: Stack

a) X-1, Y-2, Z-3

b) X-3, Y-1, Z-2

c) X-3, Y-2, Z-1

d) X-2, Y-3, Z-1

[GATE-2000]

Q.31 Match **List-I** with **List-II** select the correct answer using the codes given below the Lists:

List-I

A. All pairs shortest paths

B. Quicksort

C. Minimum weight spanning tree

D. Connected components

List-II

1. Greedy
2. Depth-first search
3. Dynamic programming
4. Divide and conquer

Codes:

	A	B	C	D
(a)	2	4	1	3
(b)	3	4	1	2
(c)	3	4	2	1
(d)	4	1	2	3

[GATE-1997]

Q.32 Consider the tree arcs of a BFS traversal from a source node W in an unweighted, connected, undirected graph. The tree T formed by the tree arcs is a data structure for computing

a) The shortest path between every pair of vertices.

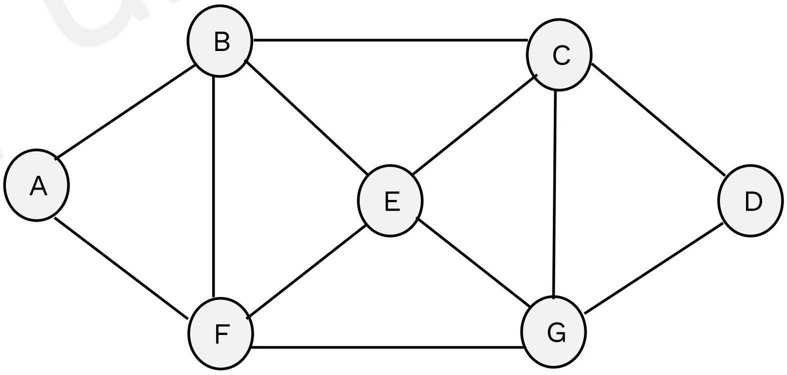
b) The shortest path from W to every vertex in the graph.

c) The shortest paths from W to only those nodes that are leaves of T.

d) The longest path in the graph.

[GATE-2014]

Q.33 Consider the following graph:



The maximum and minimum size or queue and stack required when performing BFS and DFS respectively on the above graph is_.

Q.34 Breadth First Search (BFS) is started on a binary tree beginning from the root vertex. There is a vertex t at a distance four from the root. If t is the n^{th} vertex in this BFS traversal, then the maximum possible value of n is________.

[GATE-2016]

Common Data for Q.35 &Q.36

A sub-sequence of a given sequence is just the given sequence with some elements (possibly none or all) left out. We are given two sequences X[m] and Y[n] of lengths m and n, respectively, with indexes of X and Y starting from 0.

Q.35 We wish to find the length of the longest common sub-sequence (LCS) of X[m] and Y[n] as *l*(m, n), where an incomplete recursive definition for the function *l* (i, j) to compute the length of the LCS of X[m] and Y[n] is given below:

$l(i,j)$ = 0, if either i = 0 or j =0

= expr1, if i, j > 0 and X [i – 1] = Y [j – 1]

= expr2, if i, j > 0 and X [i – 1] ≠ Y [j – 1]

Which one of the following options is correct?

a) expr1 ≡ *l*(i – 1, j) + 1

b) expr1 ≡ *l*(i, j-1)

c) expr2 ≡ max (*l*(i -1, j), *l*(i, j – 1))

d) expr2 ≡ max (*l*(i -1, j-1), *l*(i, j))

[GATE-2009]

Q.36 The values of *l* (i, j) could be obtained by dynamic programming based on the correct recursive definition of *l*(i, j) of the form given above, using an array L [M, N], where M=m + 1 and N = n + 1, such that L[i, j] =*l*(i,j).

Which one of the following statements would be TRUE regarding the dynamic programming solution for the recursive definition of *l*(i, j)?

a) All elements of L should be initialized to 0 for the values of *l*(i, j) to be properly computed

b) The values of *l*(i, j) may be computed in a row major order or column major order of L[M,N]

c) The values of *l*(i, j) cannot be computed in either row major order or column major order of L[M, N]

d) L [p, q] needs to be computed before L [r, s] if either p < r or q < s

[GATE-2009]

Q.37 The weight of a sequence a_0, a_1,... $a_{n-1}$of real number is defined as a_0+ a_1/2 + ... $a_{n-1}/2^{n-1}$. A subsequence of a sequence is obtained by deleting some elements from the sequence, keeping the order of the remaining elements the same. Let X denote the maximum possible weight of a subsequence of a_0, a_1,... a_{n-1} and Y thee maximum possible weight of a subsequence of a_1, a_2, ..., a_{n-1}Then X is equal to

a) max (Y, a_0+ Y)

b) max (Y, a_0+ Y/2)

c) max (Y, a_0+ 2Y)

d) a_0+ Y/2

[GATE-2010]

Q.38 Let P_0= 5, P_1= 6, P_2= 7, P_3= 1, P_4= 10, P_5= 2 and for 1 ≤ i ≤ 5. Let Xi be a matrix with P_{i-1} rows and P_i columns, and X = X_1, X_2, X_3, X_4, X_5, Which of the following is optimum parenthesization for computing X?

a) $((X_1(X_2 X_3)) X_4) X_5$

b) $(X_1(X_2 X_3)) (X_4 X_5)$

c) $(X_1 X_2) ((X_3 X_4) X_5)$

d) $((X_1 X_2) (X_3 X_4)) X_5$

Q.39 The correct matching for the following pairs is

List-I

A. All pairs shortest paths

B. Quick Sort

C. Minimum weight spanning tree

D. Connected Components

List-II

1. Greedy
2. Depth-First search
3. Dynamic Programming
4. Divide and Conquer

Codes:

(a) A – 2 B – 4 C – 1 D – 3

(b) A – 3 B – 4 C – 1 D – 2

(c) A – 3 B – 4 C – 2 D – 1

(d) A – 4 B – 1 C – 2 D – 3

[GATE-1997]

Common Data for Q.40 & Q.41

A_1 10 x 100,

A_2 is 100 x 5

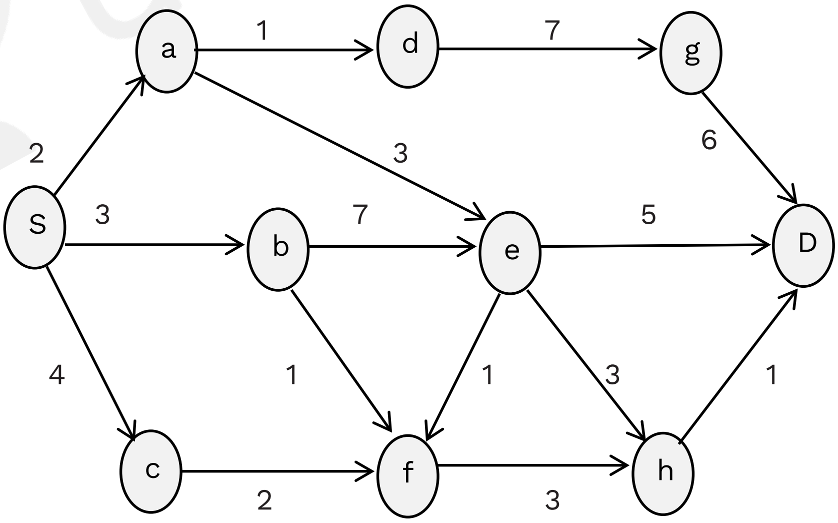
A_3 is 5 x 50, and

A_4 is 50 x 1

Q.40 Find the number of orderings that are possible to compute A_1 A_2 A_3 A_4

Q.41 Find the maximum number of multiplications required to compute A_1 A_2 A_3 A_4

Q.42 Consider the following graph G.



What is minimum distance from S to D?

Q.43 Which one of the following statements is false?

a) Optimal binary search tree construction can be performed efficiently using dynamic programming

b) Breadth-first search cannot be used to find connected components of a graph

c) Given the prefix and postfix walks over a binary tree, the binary tree cannot be uniquely constructed

d) Depth-first search can be used to find connected components of a graph

[GATE-1994]

Q.44 Match **List-I** with **List-II** select the correct answer using the codes given below the Lists:

List-I

A. Strassen's matrix multiplication

B. Kruskal's minimum spanning tree

C. Bi-connected components algorithm

D. Floyd's shortest path

List-II

A. Greedy method

B. Dynamic programming

C. Divide and Conquer

D. Depth first search

Codes:

	A	B	C	D
(a)	3	1	4	2
(b)	3	4	1	2
(c)	2	4	1	3
(d)	2	1	4	3

Q.45 Four matrices M_1, M_2, M_3 and M_4 of dimensions p x q, q x r, r x s and s x t respectively can be multiplied in several ways with different number of total scalar multiplications. For example when multiplied as ((M_1 x M_2) x (M_3 x M_4)), the total number of scalar multiplications is pqr + rst + prt. When multiplied as (M_1 x M_2) x M_3) x M_4), the total number of scalar multiplications is pqr + prs +pst.

If p = 10, q = 100, r = 20, s = 5 and t = 80, then the minimum number of scalar multiplications needed is

a) 248000

b) 44000

c) 19000

d) 25000

[GATE-2011]

unacademy

LET’S CRACK IT!