INTRODUCTION :

- **Notion of the Algorithm :**

Problem

↓

Algorithm (solution of a problem 'logic' in form of algo)

↓ (converted into program using language like C, C++ etc)

Input → [computer] → output.

* Designing an algorithm which runs on a computer.

Representation

* Graphical ~~Notation~~ (Flow chart)
* Natural language (General eng.).
* Pseudo code (General eng. + some Emp. statement)

→ Pseudo code is language Endependent. (can have use of programing language general logic)

Ans. Find the GCD of two Integer number

Method 1 : Successive substraction Method (Euclid's algo)

```
3       7       while (m != n).
m       n       {  if (m > n)
                      m = m - n
                   else  n = n - m .
                }
```

m	n
3	7
3	4
3	1
3	1
2	1
1	1

3.

17

27

37

47

Euclid's algorithm for computing gcd(m, n).

Step 1 : If n = 0, return the value of m as the answer and stop ; otherwise, proceed to step 2.

Step 2 : Divide m by n and assign the value of the remainder to r.

Step 3 : Assign the value of n to m and the value of r to n. Go to step 1

$m = 3$ $n = 7$.

Pass 1 → $r = 3 \% 7$ $(m \% n)$

$r = 3$.

$m = 7$ $n = 3$

Pass 2 → $r = 7 \% 3$

$r = 1$

$m = 3$ $n = 1$

Pass 3 → $r = 3 \% 1$

$= 0$.

$m = 1$ $n = 0$

* Euclid's algo will work for all input i,e including zero as a input.

* Euclid's will take least time as compared to no. of iteration for all other two methods

⇒ Euclid's is the most efficient algorithm.

Pseudocode Representation (Syntax)

```
ALGORITHM Euclid(m, n)
// compute gcd(m, n) by euclid's algorithm.
// Input : Two non negative, not-both-zero integers m and n.
// output : Greatest common divisor of m and n.
    while n ≠ 0 do
        r ← m mod n
        m ← n
        n ← r
    return m
```

Method 2 :

consecutive integers checking algorithm for computing gcd(m,n)

Step 1 : Assign min(m,n) to t.

Step 2 : Divide m by t. If the remainder of this division is 0, go to step 3 ; otherwise, goto step 4.

Step 3 : Divide n by t. If the remainder of this division is 0, return the value of t as the answer and stop : otherwise, proceed to step 4.

Step 4 : Decrease the value of t by 1. Go to step 2.

$t \leftarrow min\{m, n\}$

$m = 3 \qquad n = 7.$

Pass 1 ; $t = 3$; $m \% t = 3 \% 3$
$= 0$

$n \% t = 7 \% 3$
$= 1$

Pass 2 : $t = 2$; $m \% t = 3 \% 2$
$= 1$

Pass 3: $t = 1$; $m \% t = 3 \% 1$
$= 0$

$n \% t = 7 \% 1$
$= 0$

$t = 1.$

Method 5:

Middle School procedure for computing gcd (m, n).

Step 1: Find the prime factor of m.

Step 2: Find the prime factor of n.

Step 3: Identify all the common factors in the two prime expansion found in step 1 & step 2.
(If p is a common factor occuring p_m & p_n times in m, n, respectively, it should be repeted min (p_m, p_n) times)

Step 4: compute the product of all the common factors & return it as the GCD of the numbers given.

m = 3 n = 7.

prime factor of m (3) = 3, 1

" " " n (7) = 7, 1

10 Find Prime number less than or equal to N.
(Using Sieve of Eratosthenes method)

ALGORITHM

ALGORITHM Sieve(n)
// Implements the sieve of Eratosthenes
// Input : An integer $n \geq 2$.
// output : Array L of all prime number less than or equal to n.

for $p \leftarrow 2$ to n do $A[p] \leftarrow p$

for $p \leftarrow 2$ to $\lfloor \sqrt{n} \rfloor$ do // see note L J lower limit.

if $A[p] \neq 0$ // p hasn't been eliminated on previous pass

$j \leftarrow p * p$

while $j \leq n$ do

$A[j] \leftarrow 0$

$j \leftarrow j + p$.

// copy the remaining elements in L from A

$i \leftarrow 0$

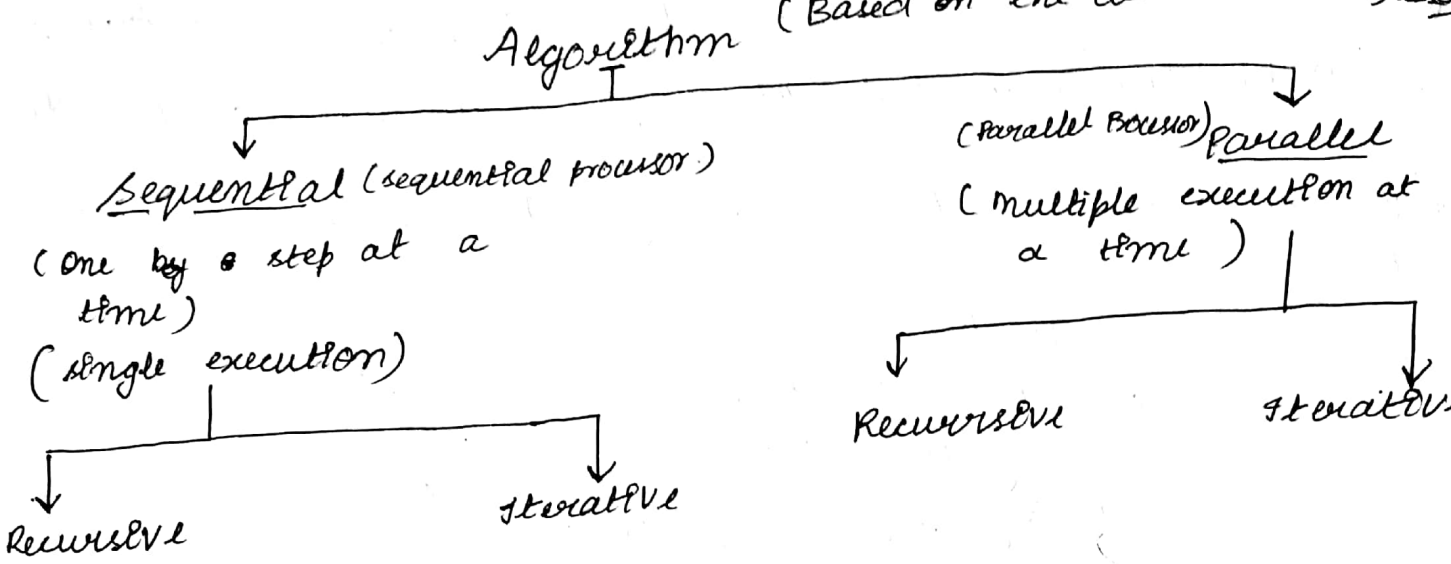
for $p \leftarrow 2$ to n do

if $A[p] \neq 0$

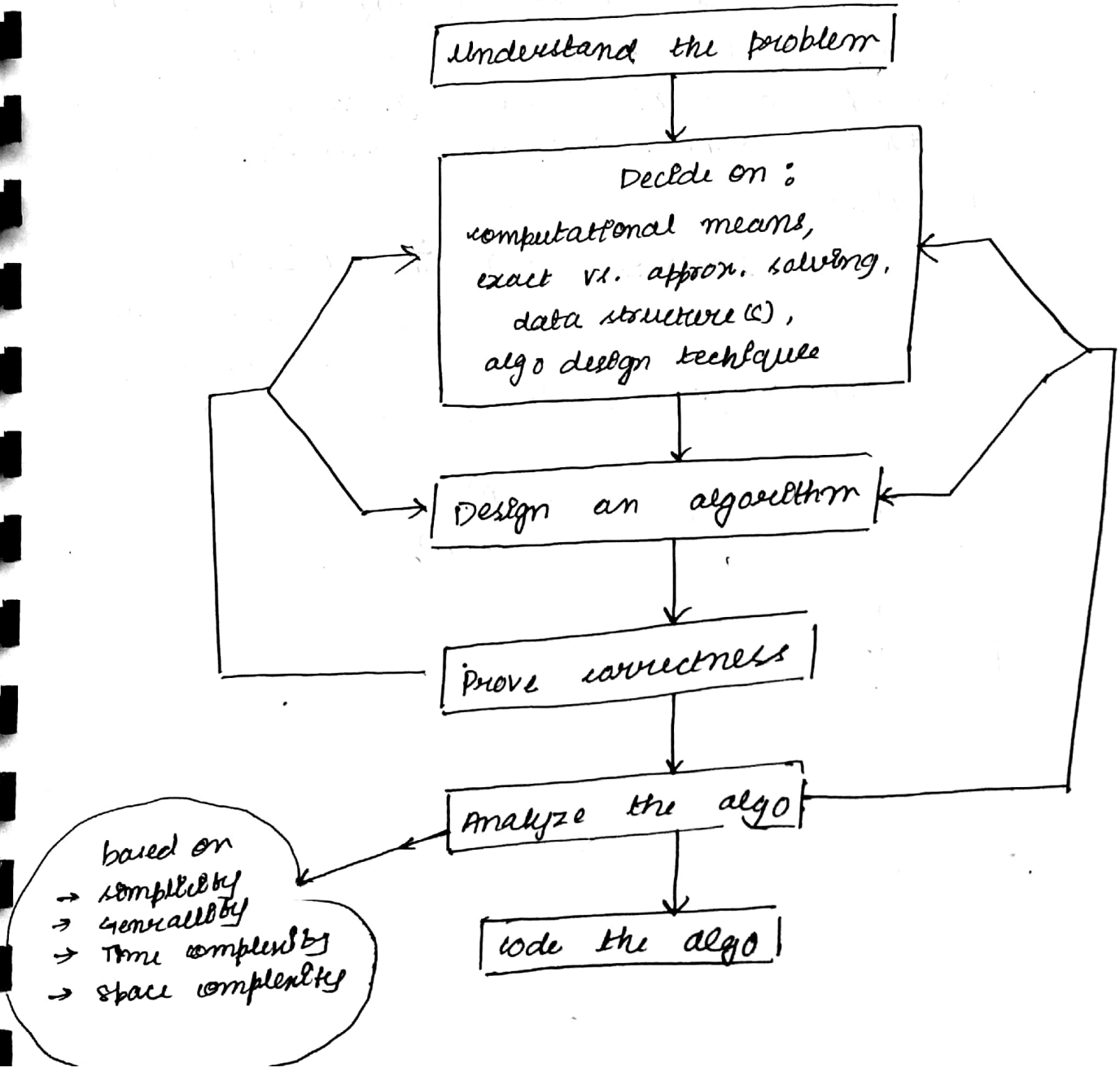
$L[i] \leftarrow A[p]$

$i = i + 1$.

return L.



ALGORITHM DESIGN PROCESS



* Time complexity is represented by mathematical equation.
  ↳ Depending upon the complexity & type we have different representation

* To compare the growth order of algorithm we have two Notations.

① Asymptotic Notations
  ↳ (i) Big O Notation
  (ii) Big theta Notation
  (iii) Big omega Notation.

} can be used to compare growth order of two function.

(I) O - notation : (for worst case) algorithm running time. growth order function.

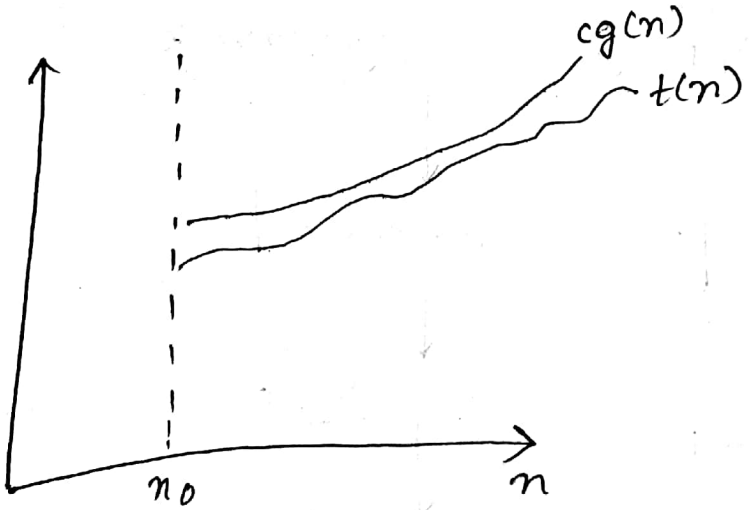
A function $t(n)$ is said to be in $O(g(n))$, denoted $t(n)$ $t(n) \in O(g(n))$, if $t(n)$ is bounded above some constant multiple of $g(n)$ for all large n i.e if $\exists$ some (+)ve constant c and some non(-)ve Integer n_0 such that

$$t(n) \leq cg(n) \quad \forall \quad n \geq n_0.$$

$n \rightarrow$ input at that instance. (input instance)

$n_0 \rightarrow$ initial condition.

[growth order of $t(n) \leq$ g.o. of $(g(n))$]



Ex. $100n + 5 \in O(n^2)$.

Indeed, $100n + 5 \leq 100n + n \quad (\forall\ n \geq 5)$

$= \quad 101n \leq 101n^2$

Thus, as value of $c \to 101$, $n_0 \to 5$

$t(n) = 101n$.

or $100n + 5 \leq 100n + 5n$

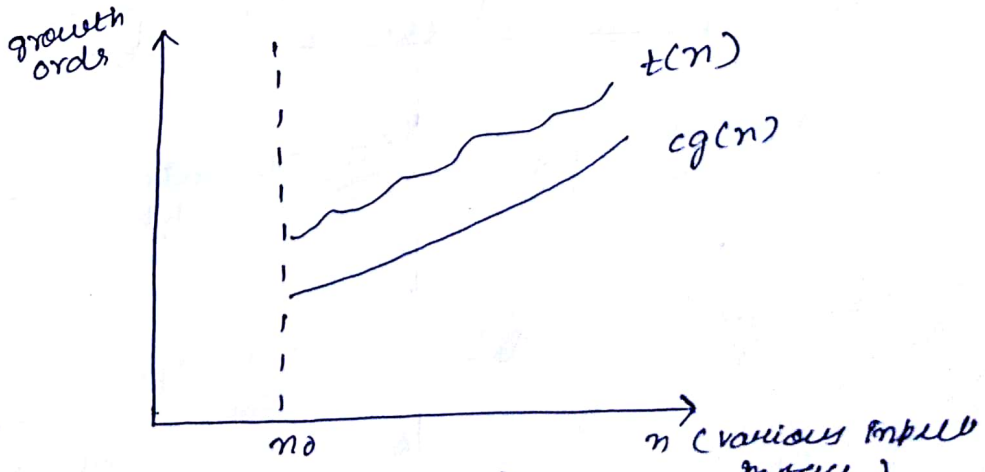
$= \quad 100n + 5 \leq 105n. \quad \forall\ n \geq 1.$

$c \to 105 \qquad n \to 1.$

II> Ω - notation : (for best case time complexity)

A function $t(n)$ is said to be in $\Omega(g(n))$, denoted $t(n) \in \Omega(g(n))$, if $t(n)$ is bounded below some positive constant multiple of $g(n)$ $\forall$ large n, i.e if $\exists$ some +ve constant c & some non -ve integer no. s.t.

$$t(n) \geq cg(n) \quad \forall \quad n \geq n_0.$$



$t(n) \in \Omega(g(n))$.

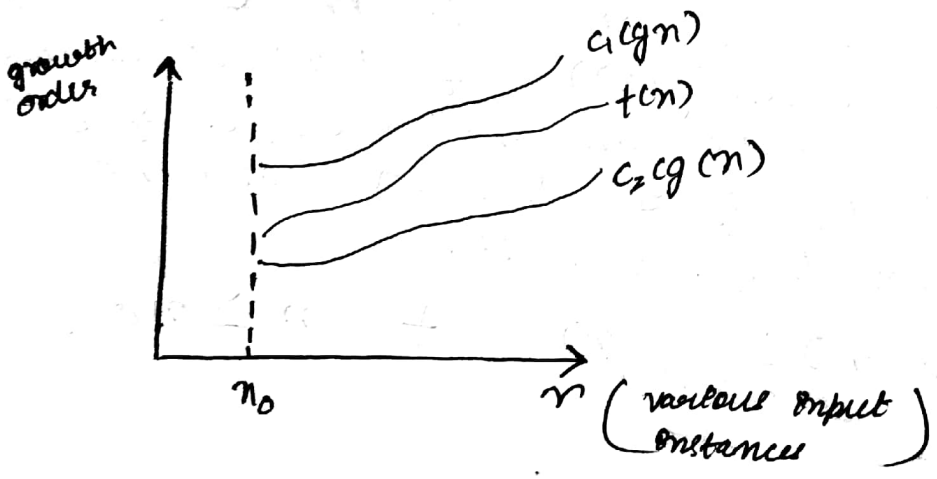
Ex. $n^3 \in \Omega(n^2)$:

$n^3 \geq n^2 \quad \forall \quad n \geq 0.$

i,e $c = 1$ and $n_0 = 0$

III) **Θ Notation**: A function $t(n)$ is said to be in $\Theta(g(n))$, denoted $t(n) \in \Theta(g(n))$, if $t(n)$ is bounded both above and below by some +ve constant multiple of $g(n)$ $\forall$ large n i.e. if $\exists$ +ve constant c_1 & c_2 & some non -ve integer no. such that

$$c_2 g(n) \leq t(n) \leq c_1 g(n) \quad \forall \quad n \geq n_0.$$



Using Limits for comparing Orders of Growth :

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & \Rightarrow t(n) \text{ has smaller order of growth than } g \\ c>0 & \Rightarrow t(n) \text{ has the same order of growth as } g \\ \infty & \Rightarrow t(n) \text{ has larger order than of growth than } g(n)^3 \end{cases}$$

$t(n)$ grows slower than $g(n)$ i.e. worst case. (O) . $t(n)$ bounded above by $cg(n)$.

$t(n)$ & $g(n)$ grows simultaneously

$t(n)$ grows faster than $g(n)$ i.e best case (Ω) $t(n)$ bounded below $cg(n)$

Ex. compare the orders of growth of $\frac{1}{2}n(n-1)$ and n^2.

Ans. $$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \frac{\frac{1}{2}[n(n-1)]}{n^2} = \frac{1}{2}\frac{n^2-n}{n^2}$$

$$\lim_{n\to\infty} \frac{1}{2}\left(1-\frac{1}{n}\right) = \frac{1}{2}.$$

⇒ both the functions grows simultaneously.

i.e. we will consider avg. case time complexity.

Ex. 2. compare order of growth of $\log_2(n)$ & $\sqrt{n}$.

$$\lim_{n\to\infty} \frac{\log_2 n}{\sqrt{n}} = \lim_{n\to\infty} \frac{(\log_2 n)'}{(\sqrt{n})'} = \lim_{n\to\infty} \frac{(\log_2 e)\frac{1}{n}}{\frac{1}{2\sqrt{n}}}$$

$$= 2\log_2 e \lim_{n\to\infty} \frac{\sqrt{n}}{n}$$

$$= 0.$$

$$\log_2 n = \log_2 e * \log_e n$$

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

$g(n) = (\sqrt{n})$ grows faster than $\log n$ as $\lim_{n \to \infty} \frac{t(n)}{g(n)} = 0$.

Ex. 3: $t(n) = n!$ $\quad g(n) = 2^n$.

$$\lim_{n \to \infty} \frac{n!}{2^n} = \lim_{n \to \infty} \frac{\sqrt{2\pi n}\,(n/e)^n}{2^n}$$

$\{ n! = \sqrt{2\pi n}(n/e)^n$ using stirling's formula $\}$

$$= \lim_{n \to \infty} \frac{\sqrt{2\pi n}\, n^n}{2^n e^n}$$

$$= \lim_{n \to \infty} \sqrt{2\pi n}\left(\frac{n}{2e}\right)^n = \infty.$$

$t(n)$ grows faster than $g(n)$. (Best case time complexity)

EFFICIENCY CLASSES:

class	Name	(best case)
1	constant	- short of best case efficiency, very few reasonable example can be given since an algo running time typically goes to ∞ when its input size grows ~~oly~~ large Ex. linear search to find first element.
$\log n$	logarithmic	- cut the problem into sub problem & solve only one sub problem.

n linear — algo that scan a list of size n (worst case) Ex. linear search, element is not present.

$n \log n$ "n-log-n" — we divide the problem into sub problem & conquer all the sub problem to find the final complexity. Ex. algo following divide & conquer.

n^2 "quadratic" - when there is nesting of loops. when it has two nested loops. Ex. printing $n \times n$ matrix

n^3 "cubic" - when there are 3 nested loops. Ex. matrix multiplication.

2^n "exponential" - Typical for algo that generates all subset of an n-element set. ~~Ex. Rude mathematical problem.~~

$n!$ "factorial — Typical for algo that generate all permutations of an n-element set.

How to write $t(n)$ and $g(n)$?

MATHEMATICAL ANALYSIS OF ALGORITHMS

1. NON RECURSIVE ALGORITHMS:

(For analysing time efficiency of nonrecursive algos)

i) Decide on a parameter (or parameters) indicating an input's size.

ii) Identify the algorithm's basic operation. (As a rule, it is located in its innermost loop.)

iii) Check whether the number of times the basic operation is executed depends only on the size of an input. If it also depends on some additional property, the worst case, avg-case and if necessary best-case efficiency have to be investigated separately.

imp.

iv) Set up a sum expressing the number of times the algorithm's basic operation is executed.

v) Using standard formulas and rules of sum manipulation, either find a closed form formula for the count or, at the very least, establish its order of growth.

- write an algorithm (non-recursive) to find maximum of n numbers and derive the time complexity expression.

```
ALGORITHM maxElement (A[0..n-1])
// Determening the value of the largest element in a given arr
// Input : An array A[0...n-1] of real numbers.
// Output : The vale of the largest element in A.
    maxval ← A[0]
    for i ← 0 to n-1 do
        if A[i] > maxval
            maxval ← A[i]
    return maxval
```

Brute force approch

// (i) Input is an array of size n.

(ii) Basic operation is comparision of two elements.

(iii) No. of time basic operation is executed is only depending on size of input.

{ if array is sorted in decending order, then best case.
if array " " " ascending order, then worst case }

$$T(n) = \sum_{i=1}^{n-1} (1) \rightarrow \text{unit time to execute one comparision}$$

$$= (n-1) - 1 + 1 \quad \{ \text{opening summision} \} \; [UL - LL + (\text{this})]$$

$T(n) = (n-1)$ → growth order function.

$T(n) = O(n)$ → linear the efficiency class

* if for loop is used go for non-recursive algo.

- Write an algorithm to find uniqueness of an array using non-recursive method and derive the time complexity expression.

ALGORITHM UniqueElements (A[0...n-1])

// Determine wether all the elements in a given array are
// distinct
// Input : An array A[0..n-1]
// output : Returns "true if all elements in A are distinct and
// false otherwise.

for i ← 0 to n-2 do
    for j ← i+1 to n-1 do
        if A[i]=A[j] return false.
return true

} Brute force.

$$T(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \quad \text{-->}$$

∴ basic operation is comparison

$$= \sum_{i=0}^{(n-2)} (n-1) - (i+1) + 1$$

$$= \sum_{i=0}^{(n-2)} n-1-i-1+1$$

$$= \sum_{i=0}^{(n-2)} (n-1) - \sum_{i=0}^{(n-2)} i$$

$$= (n-1)\sum_{i=0}^{n-2} 1 - \sum_{i=0}^{(n-2)} i$$

$= (n-1)(n-2-0+1) - \frac{(n-2)(n-1)}{2}$

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

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

$= (n^2-2n+1) - \left(\frac{n^2-3n+2}{2}\right)$ ← growth order func^n

$T(n) = O(n^2)$. quadratic efficiency class

- write an algorithm to find the product of two matrices of order $n \times n$

ALGORITHM matrixmultiplication $(A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])$

// multiplies two $n \times n$ matrices by the defination based algo

// Input: Two $n \times n$ matrices A and B

// output matrix $C = A \cdot B$.

} Brute force.

```
for i ← 0 to n-1 do
    for j ← 0 to n-1 do
        C[i,j] ← 0.0
        for k ← 0 to n-1 do
            C[i][j] ← C[i,j] + A[i,k] * B[k,j].

return C.
```

→ Three nested loops
→ Basic operation : multiplication.

→ $\sum\sum\sum(1)$ → unit time to compute 1 multiplication based on the performance of system.

Write an algorithm to convert Integer number to binary number using Non recursive method and derive the time complexity expression.

2 | 10	0
2 | 5	1
2 | 2	0
2 | 1	1
0	

1010

$\log(n) = O(n)$

ALGORITHM decimal_to_binary (n)

// To find the binary equivalent for a decimal number

// Input: a decimal number

// Output: Binary equivalent.

```
Repeat till n != 0
{
  i = n%2;
  arr[j] = i // j = 0
  n = n/2;
  ++j
}

/* for j = j till j = 0
   display arr[j];
   --j;   */
```

$$T(n) = \sum_{i=1}^{\log_2 n} 1$$

$$= \log_2 n - 1 + 1$$

$$= \log_2 n$$

Basic operation: division and modulus

$N/2 = O(n)$

$\log_2(n)$

Recursive Algorithm

General Plan for Analyzing time efficiency of recursive algorithm.

i) Decide on a parameter (or parameters) indicating an input's size.

ii) Identify the algorithm's basic operation

iii) check whether the number of times the basic operation is executed can vary on different inputs of the same size. if it can, the worst case, avg-case, and best case efficiencies must be investigated seperately.

iv) Set up a recurrence relation, with an appropriate initial condition, for the number of time basic operation is executed.

v) solve the recurrence or at least ascertain the order of growth of its solution.

* In recursive function there must be relation b/w input and output, that defins the termination condition of the ~~recursive~~ recurrance relation.

Q) Write an algorithm to count the number of binary digit in a decimal integer number using recursive method.

ALGORITHM BinRec(n)

// Input: A positive decimal integeron.

// output: The numba of binary digits in n's binary representation

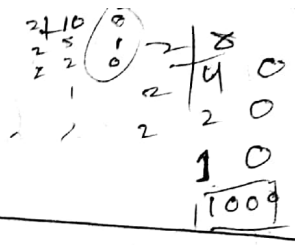
// if n=1 return 1

... return BinRec(⌊n/2⌋) + 1

$\lfloor \rfloor$ → floor function

Ex. 3.5 will be considered as 3.

4.9 4



Basic operation is division

Time required to solve problem of input instance 'n'

$$T(n) = \begin{cases} \text{If } (n=1) & \text{Termination} \\ \text{else} & T(n/2)+1 \end{cases}$$

General expression; $T(n) = T(n/2) + 1$

/* To find the efficiency class we have to solve recursive function:

i) Backward
ii) Forward
iii) Master Theorem
iv) Recursive Tree method. */

Using Backward substitution

$$T(n) = T(n/2) + 1$$
$$= [T(n/4)+1] + 1$$
$$= [T(n/2^2) + 2]$$
$$= [T(n/2^3)+1] + 2$$
$$= T(n/2^3) + 3$$
$$\vdots$$

at k^{th} recursive call

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

$n/2^k = T(n) = (1)$

$(n/2^k = T(1)) \quad (2)$

$n = 2^k$

$\log_2 n = k \log_2 2$

$k = \log_2 n$

$n = 2^k$ (When $n = 2^k$ order then algo will take maximum time).

$\log_2 n = k \log_2 (2)$

$(k = \log_2 n)$

$T(n) = T(n/2^k) + k$ $\quad \because (n = 2^k)$

$= T(n/n) + k$

$= T(1) + \log_2 n$

$T(n) = 1 + \log_2 n$

$= O(\log_2 n)$ $\quad$ { $\log_2 n$

Imp.

Write an algorithm to find n^{th} Fibonacci number using recursive method and derive an alternative formula to find n^{th} Fibonacci number using recursive technique.

0 1 1 2 3 5 8 ....

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{else } f(n-2) + f(n-1) \end{cases}$$

$f(n) = f(n-2) + f(n-1)$ — (I)

$f(n) - f(n-1) - f(n-2) = 0$ { II order homogeneous recursive function — (II).

/* ※ if $f(n) - f(n-1) - f(n-2) = k$ { non homogeneous } */

When II order homogenous recursive function is given then we have to find characteristic equation for the following.

The characteristic equation of the generalized homogenous function is

$ax^2 + bx + c = 0$ — (A)

$ax(n) + bx(n-1) + cx(n-2) = 0$ — (III).

comparing (II) & (III).

$a = 1$; $b = -1$; $c = -1$.

substitute the values of (A) (B) (C) in (A).

$$x^2 - x - 1 = 0$$

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

$$x = \frac{1 \pm \sqrt{1+4}}{2} = \frac{1 \pm \sqrt{5}}{2}$$

$$\Rightarrow x_1 = \frac{1+\sqrt{5}}{2} \quad ; \quad x_2 = \frac{1-\sqrt{5}}{2}$$

General solution of the (II) order homogeneous equation is given by

$$f(n) = \alpha x_1^n + \beta x_2^n \quad \text{—— (III)}$$

$$f(n) = \alpha \left(\frac{1+\sqrt{5}}{2}\right)^n + \beta \left(\frac{1-\sqrt{5}}{2}\right)^n \quad \text{—— (V)}$$

for $n = 0$

$$f(0) = \alpha \left(\frac{1+\sqrt{5}}{2}\right)^0 + \beta \left(\frac{1-\sqrt{5}}{2}\right)^0$$

$$0 = \alpha + \beta \Rightarrow \boxed{\alpha = -\beta}$$

for $n = 1$

$$f(1) = \alpha \left(\frac{1+\sqrt{5}}{2}\right)^1 + \beta \left(\frac{1-\sqrt{5}}{2}\right)^1$$

$$1 = \alpha \left(\frac{1+\sqrt{5}}{2}\right) + (-\alpha)\left(\frac{1-\sqrt{5}}{2}\right)$$

$$1 = \frac{\alpha}{2} + \frac{\alpha\sqrt{5}}{2} - \frac{\alpha}{2} + \frac{\alpha\sqrt{5}}{2}$$

$$\frac{2\alpha\sqrt{5}}{2} = 1$$

$$\boxed{\alpha = 1/\sqrt{5}} \Rightarrow \boxed{\beta = -\frac{1}{\sqrt{5}}}$$

substitute α and β in (V) (neglecting)

$$f(n) = \left(\frac{1}{\sqrt{5}}\right)\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

(as the n value increase the second term becomes smaller.)

$$f(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n$$

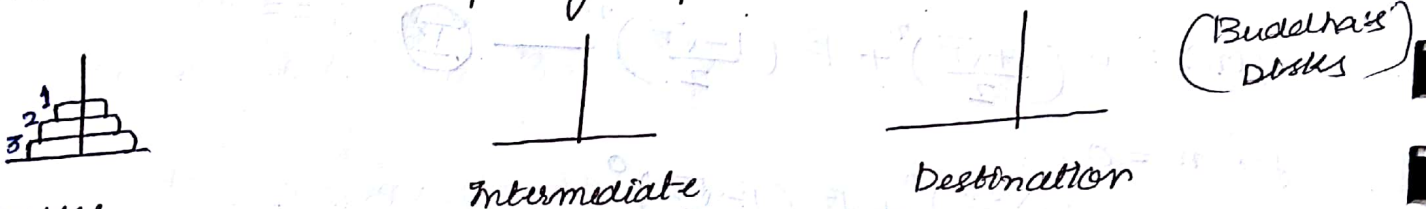
$$= \frac{1}{\sqrt{5}}(\Phi)^n = \frac{1}{\sqrt{5}}(1.618)^n$$

$\left\{\because \frac{1+\sqrt{5}}{2} \simeq 1.618\right\}$

$$\boxed{f(n) = \frac{1}{\sqrt{5}}(1.618)^n} \quad \text{---} \quad (*)$$

(*) is the alternative formula to find the n^{th} fibonacci number.

4) Write an algorithm to find the minimum number of disk movements in "Towers of Hanoi" problem and derive the time complexity expression to solve this problem.



Task is to move the set of disks from source to destination

Conditions:

1) Smaller disk must be above the larger disk.
2) Transfer only one disk from one pole to other pole at a time.
3) Find the minimum no. of disk movements.

$$M(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3 & \text{if } n = 2 \\ 7 & \text{if } n = 3 \\ m(n-1)+m(n-1)+1 & n \geq 2 \end{cases}$$

Step 1 : Transfer $(n-1)$ no. of disk from $S \to T \Rightarrow m(n-1)$
using destination pole as intermediate

Step 2 : Transfer n^{th} disk from $S \to D \Rightarrow 1$

Step 3 : Transfer $(n-1)$ disks from $T \to D$ using source as intermediate $\Rightarrow m(n-1)$

$$m(n) = 2\,m(n-1) + 1$$

Generalized expression.

$$m(n) = 2\,m(n-1) + 1 \quad \text{—— (I)}$$

(first order recurrence function)

$$= 2\left[2\,m(n-2)+1\right] + 1$$

$$= 2^2 m(n-2) + 2^1 + 2^0$$

$$= 2^2\left[2\,m(n-3)+1\right] + 2^1 + 2^0$$

$$= 2^3 m(n-3) + 2^2 + 2^1 + 2^0$$

$$\vdots$$

$$= 2^i\, m(n-i) + 2^{i-1} + 2^{i-2} + \ldots 2^0$$

substitute $i = n-1$

$$= 2^{n-1}\, m(n-n+1) + 2^{n-1-1} + \ldots 2^1 + 2^0$$

$$= 2^{n-1}\, m(1) + 2^{n-2} + \ldots 2^1 + 2^0$$

$$= 2^{n-1}(1) + 2^{n-2} + \ldots 2^1 + 2^0$$

$$= 2^{n-1} + 2^{n-2} + \ldots 2^1 + 2^0 \quad \text{—— (II)}$$

$$\# \boxed{\sum_{i=0}^{n} a^i = \frac{a^{i+1}-1}{a-1}}$$

{ It is the solution of (II) the binomial distribution }

$$m(n) = 2^n - 1$$

$$T(n) = O(2^n) - (\text{Polynomial class})$$

ALGORITHM Tower_Hanoi (desk, source, dest, aux)
// Find the minimum number of disk movements
// Inputs: Number of disk
// Output: Total number of disk movements from source to
// destination.

if desk == 1, then
move desk from source to dest.

else
Tower_Hanoi (desk-1, source, aux, dest) // step 1
Tower_move n^{th} desk from source to dest // step 2
Tower_Hanoi (desk-1, aux, dest, source) // step 3

end if

4. Write an algorithm to arrange list of n-data in ascending order using bubble sort and derive the time complexity expression.

Note:
- ✓ Ascending : Non decreasing order.
- ✓ Descending : Non increasing order

* Increasing is different ~~order~~ from ascending
* Decreasing is different from descending.

- Increasing order : The difference b/w the ~~item~~ successive elements are constant.
- * Ascending order : No such boundation of common diff.
- Decreasing order : The diff b/w successive element is constant
- * Descending order : No boundation of common diff.

Bubble sort is the worst technique because no. of comparision is maximum as compared to other sorting technique.

```
ALGORITHM BubbleSort ( A[], n )
// Input : List of n data items
// Output : Sorted list in ascending order.

for i = 0 to n-1 do :
    for j = 0 to n-i do :
        /* compare the adjecent element */
        if A[j] > A[j+1] then
        /* swap them */
            swap ( A[j], A[j+1] )
        end if
    end for
end for.
```

5 4 3 2 1 — c = 4, s = 4
4 5 3 2 1
4 3 5 2 1
4 3 2 5 1
4 3 2 1 [5]

4 3 3 2 1 — c = 3, s = 3
3 4 2 1
3 2 4 1
3 2 1 [4]

3 2 1 — c = 2, s = 2
2 3 1
2 1 [3]

2 1 — c = 1, s = 1
1 2

$\{ \Sigma c = 10 \; \Sigma s = 10 \}$ i.e total $= 2\sum (1+2+3+\ldots n-1)$

$$T(n) = 2 \sum 1+2+3+\cdots (n-2)(n-1)$$
$$= 2 \frac{n(n-1)}{2}$$
$$= n^2 - n = O(n^2).$$

General method:

$$T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-i} 1$$

$$T(n) = \sum_{i=0}^{n-1} (n-i+1)$$
$$= n \sum_{i=0}^{n-1} i - \sum_{i=0}^{n-1} 1 + n \sum_{i=0}^{(n-1)} 1.$$
$$=$$

$$= -\frac{n(n-1)}{2} + [(n-1)-0+1] + n[(n-1)-0+1]$$

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

$$= -\frac{n^2}{2} + \frac{n}{2} + n + n^2 + \cdots = \frac{n^2}{2} + 3n/2$$

$$= O(n^2)$$

Write the algorithm and derive Time complexity expression to arrange the data in ascending order using selection sort technique.

```
ALGORITHM selectionsort ( list[], n )
// Input : list of n data
// Output : sorted

for i=1 to n-1
   min = i  /* set current element as min */
            /* check with the element to be min */

   for j = i+1 to n
      if list[j] < list[min] then
         min = j ;
      end if
   end for
   /* swap the min element with current
      element */
   if indexmin != i then
      swap list[min] and list[0]
   end if
endfor
```

5 4 3 2 1 — c = 4, s = 1

[1] 4 3 2 5

4 3 2 5 — c = 3, s = 1

[2] 3 4 5

c = 2, s = 1

[3] 4 5

c = 1, s = 1

[4] 5

$\Sigma c = 10$

$\Sigma s = 4.$

$$T(n) = \sum 1+2+3+\cdots(n-1)$$

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

$$= O(n^2).$$

Standard step.

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1$$

$$= \sum_{i=1}^{n-1} (n - (i+1) + 1)$$

$$= \sum_{i=1}^{n-1} n - i - 1 + 1$$

$$= n \sum_{i=1}^{n-1} 1 - \sum_{i=1}^{n-1} i$$

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

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

$$= \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = O(n^2).$$

Write an algorithm and derive time complexity expression to implement insertion sort.

→ (Decrease and conquer designing technique).

ALGORITHM Insertion (A[], n):

// Input: List of n data

// output: sorted elements in ascending order.

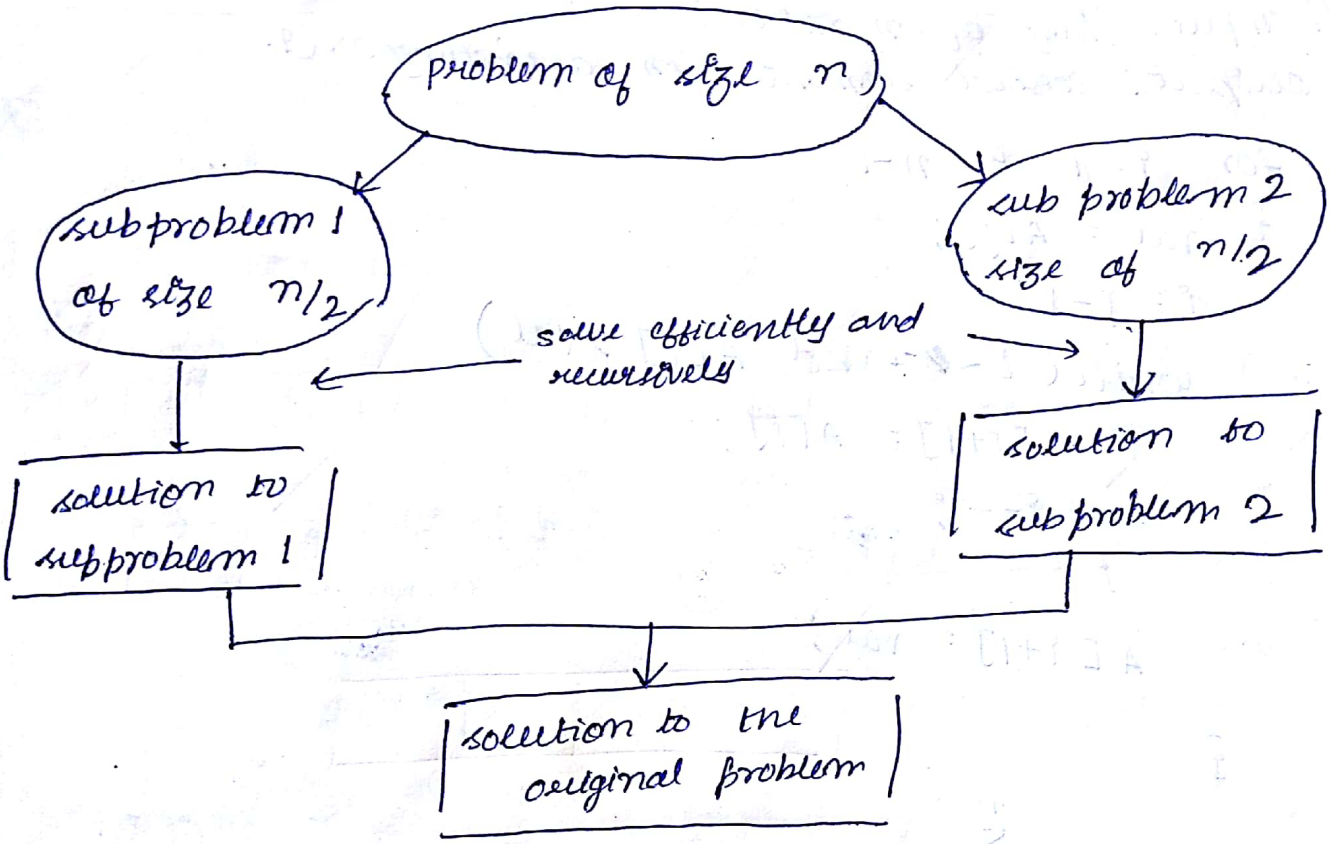
```
for j = 1 to n-1
{  val = A[j];
   i = j-1;
   while ( i<=0-1 and A[i] >val )
     {  A[i+1] = A[i];
        i--;
     }
     A[i+1] = val;
}
```

$$T(n) = \sum_{j=1}^{n} .$$

Time complexity
space complexity
simplicity
generality

UNIT-2

DIVIDE AND CONQUER DESIGNING TECH.



*. For divide and conquer iterative approch is more complex as compared to recursive approch

- Divide : the problems into number of sub problems.
- conquer : the sub problem by solving them recursively. If the subproblem size are small enough, however just solve the subproblem in brute force manner
- combine : the solution to the subproblem into the solution for the original problem.

Merge Sort : (Merge two sorted array).

ALGORITHM merge (B[0...p-1], C[0...q-1], A[0...p+q-1])

// Merge two sorted array into one sorted array.

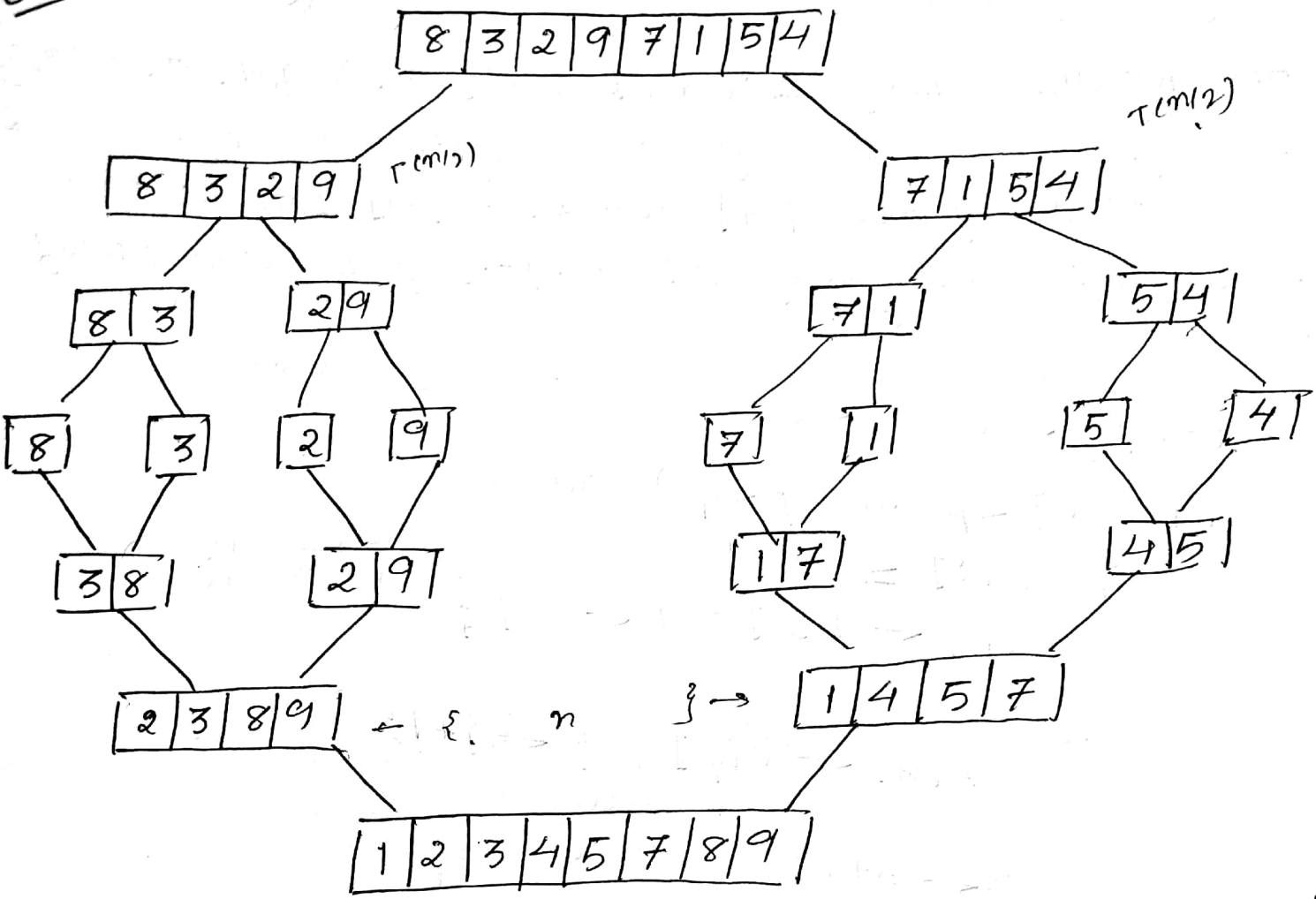
// Input : Array B[0...p-1] and C[0...q-1] both sorted.

// Output : Sorted array A[0...p+q-1] of the elements of B and C

```
i=0; j ← 0; k ← 0
while (i < p and j < q) do
    if B[i] ≤ C[j]
        A[k] ← B[i]; i ← i+1;
    else
        A[k] ← C[j]; j ← j+1;
    k ← k+1;
if i = p
    copy C[j...q-1] to A[k...p+q-1]
else
    copy B[i...p-1] to A[k...q+p-1].
```

* Total no. of comparison will be equal to the no. of elements in B or C (which ever is have smaller size).

Example



ALGORITHM MergeSort (A[0...n-1])

// sort array A[0...n-1] by recursive mergesort.

// Input: An array A[0...n-1] of orderable elements.

// output: Array A[0...n-1] sorted in non decreasing order

if $n > 1$

copy $A[0 \ldots \lfloor n/2 \rfloor - 1]$ to $B[0 \ldots \lfloor n/2 \rfloor - 1]$

copy $A[\lfloor n/2 \rfloor \ldots n-1]$ to $C[0 \ldots \lfloor n/2 \rfloor - 1]$

mergesort ($B[0 \ldots \lfloor n/2 \rfloor - 1]$)

mergesort ($C[0 \ldots \lfloor n/2 \rfloor - 1]$)

merge (B, C, A)

mergesort (A[], b, r, n)

MS(A[], b, mid, n_2)

MS(A[], mid+1, q, $n_{1/2}$)

Merge(A[], i, j, k)

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ T(n/2) + T(n/2) + n & \text{otherwise.} \end{cases}$$

$T(n/2)$ → to solve first half of subproblem
$T(n/2)$ → to solve second half
n → to merge

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

For k^{th} recursive call $= 2^k T(n/2^k) + k \cdot n$ — (I)

When ever the input instance is in power of two we will get equal sub problems.
Which means 2^k it will take maximum time.

$$\Rightarrow n = 2^k \text{ — (II) (worst case).}$$

$$\log_2 n = k \log_2 2$$

$$\boxed{\log_2 n = k}$$

Substitute (II) in (I)
$$= n\,T\left(\frac{2^k}{2^k}\right) + n.\log_2 n$$
$$= n\,T(1) + n\log_2 n$$
$$= 0 + n\log_2 n$$

$$T(n) = O(n\log_2 n). \quad (\text{nlogn class})$$

Merge sort is better than bubble & selection.

Q) Write an algorithm to implement binary search technique using divide and conquer designing technique and analyse the efficiency of an algorithm. (Divide and conquer)

*(Basic operation: comparison) BINARY SEARCH

Here, the efficiency of algorithm depends on the position of the key element, not on the input instance.
Therefore, we will calculate Best case, Worst case and Avg case time complexity.

1) Sorted data. (Essential for Binary search).

$A = \{ \overbrace{[2, 4, 6]}^{0\ 1\ 2} \overbrace{[8, 9, 12]}^{3\ 4\ 5} \}$;

Low = 0
high = n-1

Key = 13

mid = (L+H)/2;

(0+5)/2 = ⌊2.5⌋ = 2.

compare mid element with key and then check wether it is > or < mid and change L or H accordingly.

here, L = mid+1; (stop if L = H)

mid = (L+H)/2;

(3+5)/2 = 4.

compare mid element with key and then check wether it is > or < or = to mid & change L or H accordingly.

here L = mid+1; (stop if L = H)

mid = 5;

mid = (5+5)/2 = 5.

compare mid element with key and then check wether it is > or < or = to mid

check (L = H) if true.

* data not present

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

Time to cut the main prob into sub problem

Time taken to solve the sub problem.
(selected sub problem)

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

$$= [T(n/2^2) + 1] + 1$$

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

$$\vdots$$

at k^{th} recursive call $= T(n/2^k) + k$ — (1)

When ever input instance is in power of two it will take maximum no. of recursive call or comparision as we will have 2 equal sub problems.

$$n = 2^k$$

$$\boxed{\log_2 n = k.} \quad \text{— (II)}$$

Substitute (II) in (I).

$$= T(2^k/2^k) + k$$

$$= T(1) + \log_2 n$$

$$= \log_2 n + 1.$$

$$\boxed{T(n) = O(\log_2 n)}$$

Linear Search

1. Unsorted Data
2. Best case is considered when data is at beginning of the array
3. Linear time
4. Max no. of comparision
   n
5. Linear search is slower

Binary search.

Sorted Data

Best case is considered when data is at the mid.

logarithmic

max no. of comparision
$\log_2(n) + 1$

It is faster

n	L.S	B.S
6	6	3
12	2(6)	3+1

(comparisions).

ALGORITHM Binary search (A[0...n-1], K) // Iterative Approch.
// Implement nonrecursive binary search
// Input: An Array of A[0..n-1] sorted in ascending order &
// a search key.
// Output: An index of the array's element that is equal to K.
// Or -1 if there is no such element.

```
l ← 0; r ← n-1
while l ≤ r do
    m ← ⌊(l+r)/2⌋
    if K = A[m], return m
    else if K < A[m] r ← m-1
    else l ← m+1

return -1.
```

* Quick Sort (Devide and conquer)

(There the main problem is devided into unequal sub problem.

→ It depends on what kind of data we consider, not on input instance. (Devision)

→ Here time complexity will be calculated seperately.

Binary search // Recursive Approch:

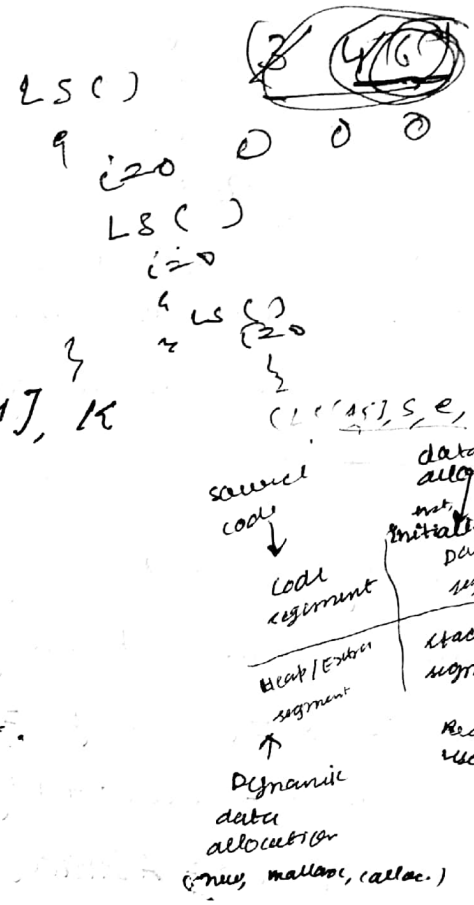
```
ALGORITHM BinarySearch_R ( A[ ], K )
    low = 0,  high = n-1
    mid = (low + high)/2
    if (K = A[mid]) then
        return mid
    else if (K < A[mid]) then
        return BinarySearch_R ( A[0---. mid-1], K )
    Else
        return BinarySearch_R ( A[mid+1---n-1], K )
    End if.
```

Linear Search Algorithm:

```
ALGORITHM linear_search ( list[ ], value )
// Input: Array of n elements
// output: success or failure.
for each item in the list
    if match item == value
        return the item's location
    end if
end for
return -1.
```

ALGORITHM LS(A[], K)

```
    i=0
    if (K == A[i]) than
        return i.
    else
        return LS(A[i+1 .... n-1], K
    return -1
```



* Recursion utilize the stack segment but Iterative approch don't use stack segment.

Recursion utilize memory space optimally

Ans.
Write an algorithm and derive the time complexity expression using of quick sort using Divide and conquer designing Technique.

0	1	2	3	4	5	6	7
5	3	1	9	8	2	4	7.

Select the pivot element.
the next element to pivot is the left pointer (L)
and last element is the right pointer (R)
if L == R (array contain 1 element)
L < R (array contain more than 1 element)

compare pivot & L if (pivot(5) > L(3)) ✓
L = L+1.

again compare pivot & L if (pivot(5) > L(1)) ✓
L = L+1

aag compare pivot & L if (pivot(5) > L(9)) (false)
stop L.

compare pivot to R if $(R^{(4)} > pivot^{(5)})$

$R = R - 1$;

compare pivot to R if $(R^{(4)} > pivot^{(5)})$

stop R.

check $L \leq R$ (True) ($\because L = 3 \quad R = 6$).

swap L and R.

$L^{(4)}$ $L = L + 1$ (4) $\quad$ $R^{(9)}$ $R = R - 1$ (5)

$L(8)$ $\quad$ $R(2)$

compare pivot & L $\quad$ $(pivot^{(5)} > L^{(8)})$ (False)

stop.

compare pivot & R $\quad$ $(pivot^{(5)} < R^{(2)})$ (False)

stop.

check $L \leq R$ (True) $\quad$ ($\because L_4 \leq R_5$.)

swap L] and R[and L++ (5) & R-- (4).

Now, $L > R$.

swap pivot element with R.

0	1	2	3	4	5	6	7
2	3	1	9	8	5	8	7

8	9	7
↑	L	R.
P		

P[] > L[] False

stop

P[] < R[] False

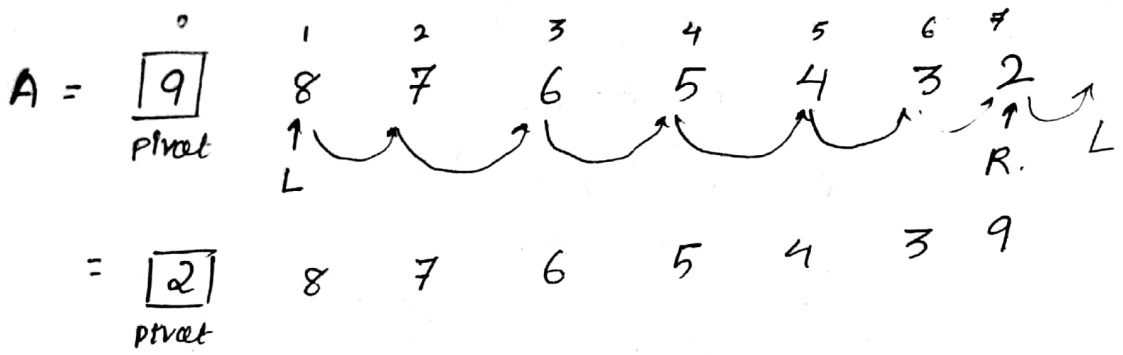
stop.

swap (L & R]

8	7	9.
↑	↑	↑
P	R	L

if (L > R)

swap (P, R).



* When we have input as sorted array then the partition of that array using quick sort, we will have only one sub problem.
* In case of unsorted the partition can be equal or unequal sub problems (2 (Two)).

→ For quick sort the partition doesn't depends on no. of value in array, it depends on the type of data we are inserting

→ Here, the complexity is depending on some other property other than input instance. Hence, we have to consider Best, worst, and Avg. case

- If we are getting equal sub problem during partition we will analyse Best case
- If we are getting unequal sub problem during partition we will analyze Avg. case.
- For sorted array i.e one sub problem. we will consider worst case time complexity.

Best Case:

$$T(n) = \begin{cases} 1 & \text{if } (n=1) \\ T(n/2) + T(n/2) + n & \text{if } (n \geq 2) \end{cases}$$

For best case we have two equal sub problems.

* For partitioning the array into two sub problem using partitioning algorith max 2 comparison n (operation).

$T(n/2)$ → solve left sub problem $T(n/2)$ → right sub proble

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

$$= 2[2T(n/4) + n/2] + n$$

$$= 2^2 [T(n/4)] + 2n$$

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

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

$$= 2^3 [T(n/2^3)] + 3n$$

$$= 2^k [T(n/2^k)] + 8kn$$

worst max no. of comparison were

$$n = 2^k$$

$$\log_2 n = k.$$

$$= n(1) + n \log_2(n)$$

$$= O(n \log_2(n)).$$

Worst case:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + n & \text{if } n \leq 2 \end{cases}$$

$$T(n) = T(n-1) + n$$
$$= T(n-2) + (n-1) + n$$
$$= T(n-3) + (n-2) + (n-1) + n$$
$$\vdots$$
$$= T(n-k) + (n-(k-1)) + [n-(k-2)] + \cdots n$$
$$= T(n-(n-1)) + n - (n-1-1) + [n-((n-1)-2)] + \cdots$$
$$= T(1) + 2 + 3 + \cdots n.$$
$$= 1 + 2 + 3 + \cdots n$$
$$= \frac{n(n+1)}{2} = \frac{n^2+n}{2}$$
$$= O(n^2).$$

Average case:

$$T(n) = \begin{cases} 1 & \text{if } (n = 1) \\ T(k-1) + T(n-k) + n. & \text{if } (n \geq 2) \end{cases}$$

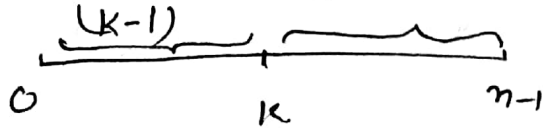
partition position is not predictable. ∴ partition position will be k $(0 \leq k \leq n-1)$.

$n \rightarrow$ atmost comparision to perform partition.

$T(k-1) \rightarrow$ complexity to find this sub problem

$T(n-k) \rightarrow$ " " second " "

$(n-k) = [\because (n-1-(k-1)]$
$= n-k.$



$$T(n) = \sum_{K=0}^{n-1} \left[\frac{T(K-1) + T(n-K)}{n} \right] + n$$

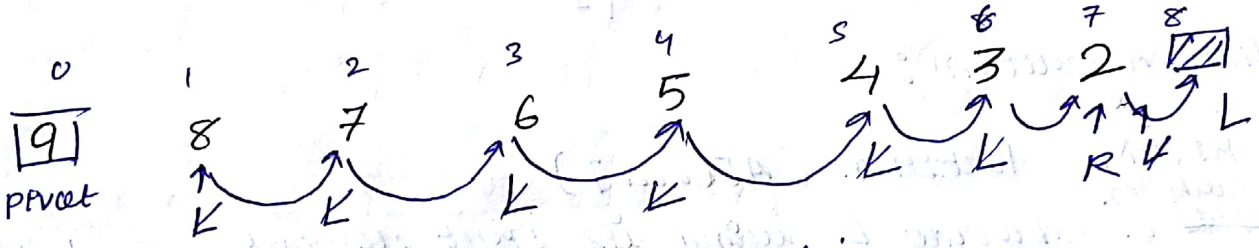
$$T(n) = \frac{1}{n} \sum_{K=0}^{n-1} \left[T(K-1) + T(n-K) \right] + n$$

⇒ Solve H.W.

* We are taking mean of (K to n-1) for analysing Avg. Time complexity.

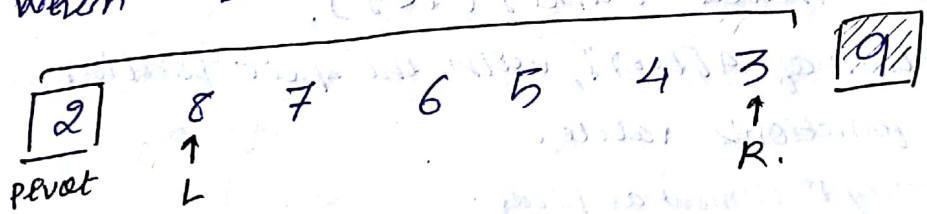
$$T(n) = 1.38\, n \log_2 n.$$ Avg.

⇒ Avg. case is 38% more than Best case.



L = L+1 till arr[pivot] > arr[L];

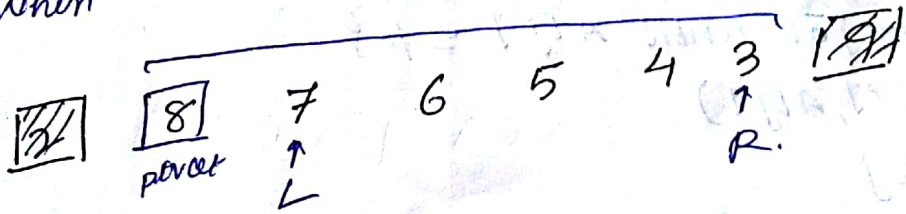
when L > R swap (arr[pivot], arr[R].



arr[plot] < arr[L]. stop.

R = R-1, till arr[pivot] < arr[R].

when R < L swap (arr[R], arr[pivot].



Quick sort :

$l > r$ (No element)
$l = r$ (one element)

ALGORITHM Quicksort (A[l...r]

// sort a subarray by quicksort.
// Input : A subarray A[l..r] of [A[0..n-1], defined by its
// left and right indices l and r
// Output : Subarray A[l...r] sorted in nondecreasing order

if $l < r$

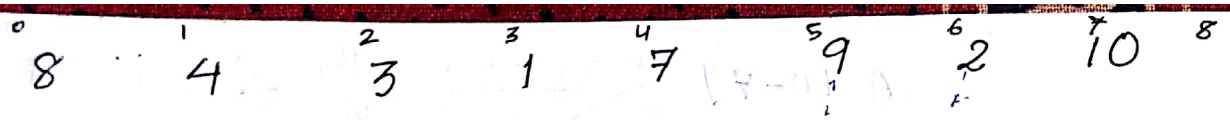
s ← Partition (A[l...r]) // s is split position.
Quicksort (A[l... s-1])
quicksort (A[s+1.... r])

Partition Algorithm :

ALGORITHM Partition (A[l...r])
// ~~sorts~~ Partition a subarray by using its first element as a pivot.
// Input : A subarray A[l...r] of A[0...n-1], defined by
// its left and right indices l and r (l<r).
// Output : A partition of A[l..r], with the split position
// returned as this function's value.

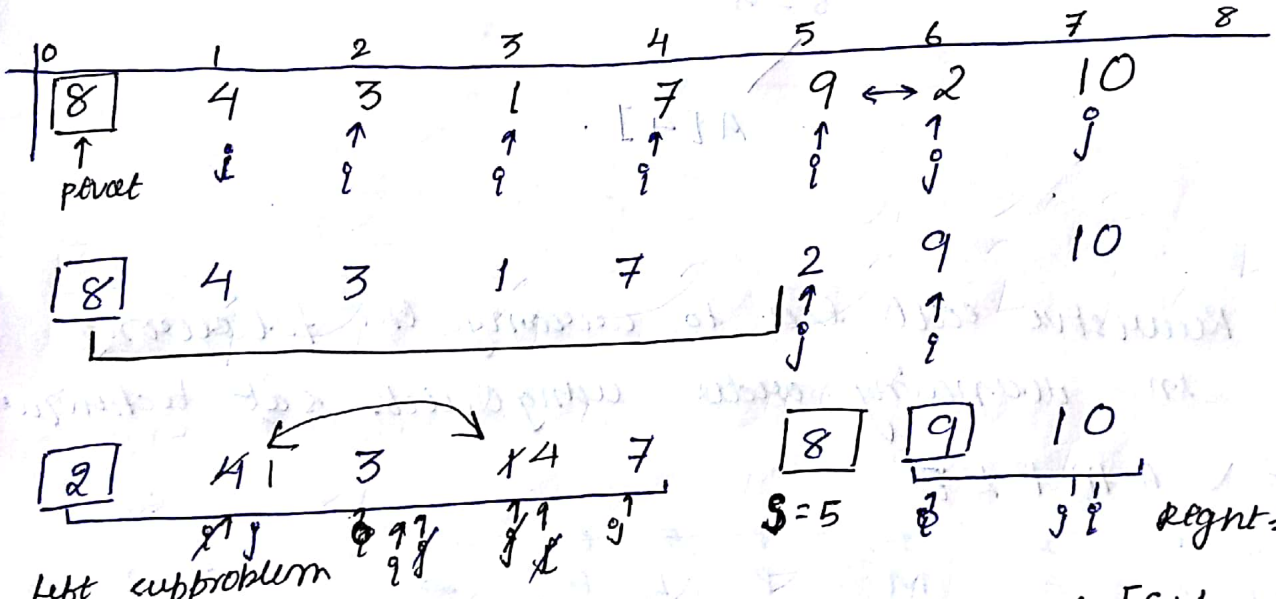
p ← A[l] // selecting 1st element as pivot.
i ← l ; j ← r+1 ;
repeat
  repeat i ← i+1 until A[i] ≥ p }
  repeat j ← j-1 until A[j] ≤ p }
  swap (A[i], A[j])
until i ≥ j
swap (A[i], A[j]) // undo last swap when i ≥ j
swap (A[l], A[j])
return j

loop.



We are initializing $j = n+1$ bcoz inside the loop. we are first incrementing i (left pointer) and decrementing j (right pointer).

Write a recursive call binary tree to arrange the following data in ascending order using Quick sort algorithm and find the type of efficiency class.



S = 5

Left subproblem

A [0 . . . S-1]

A [0 . . . 4]

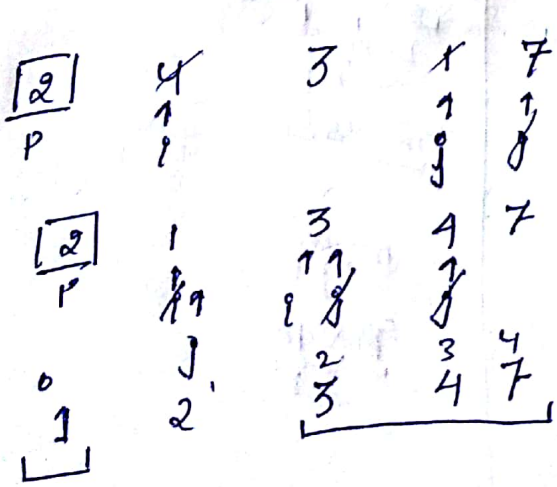
S = 1

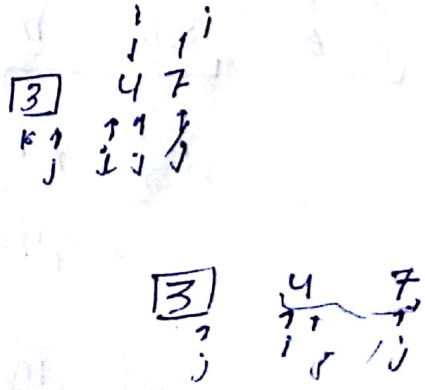
Right subproblem

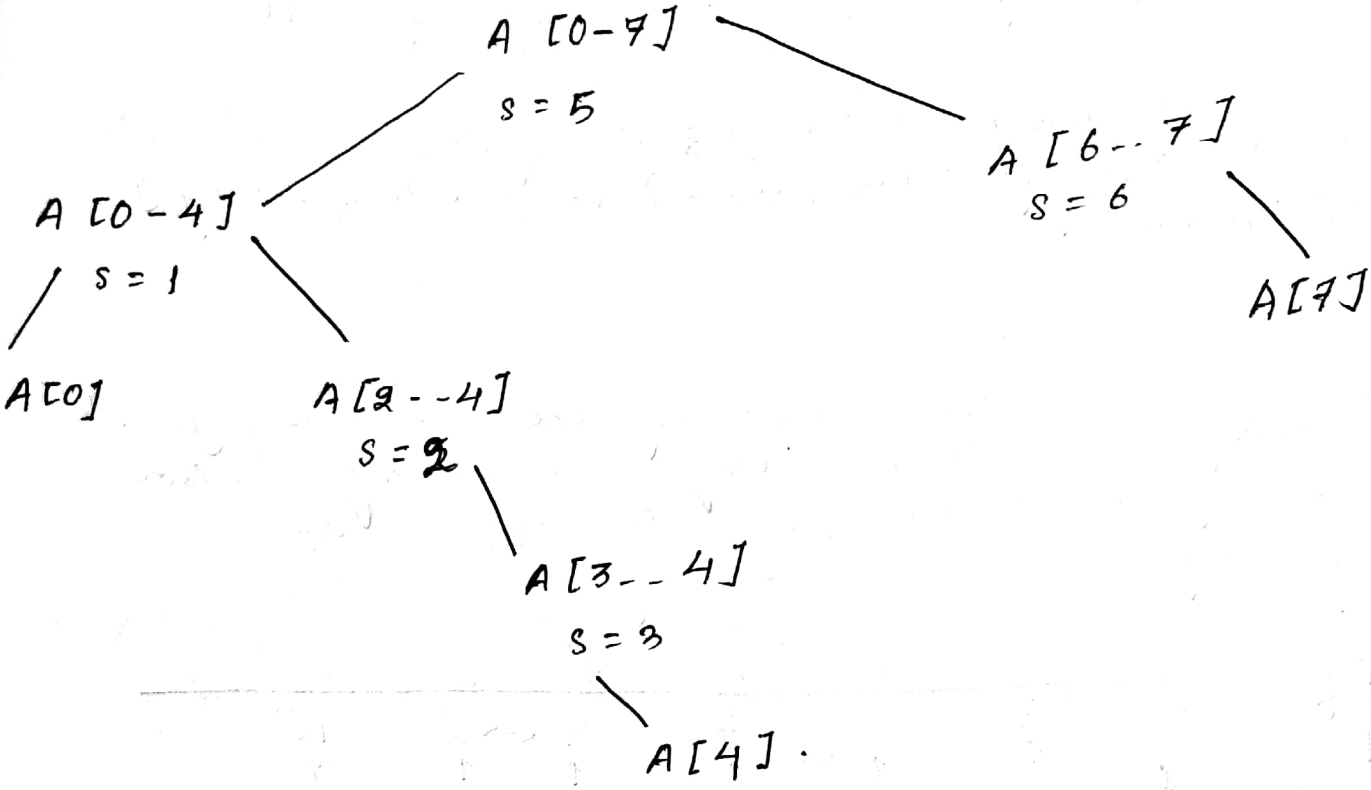
A [S+1 . . . (n-1)]

A [6 . . . 7]

S = 6.

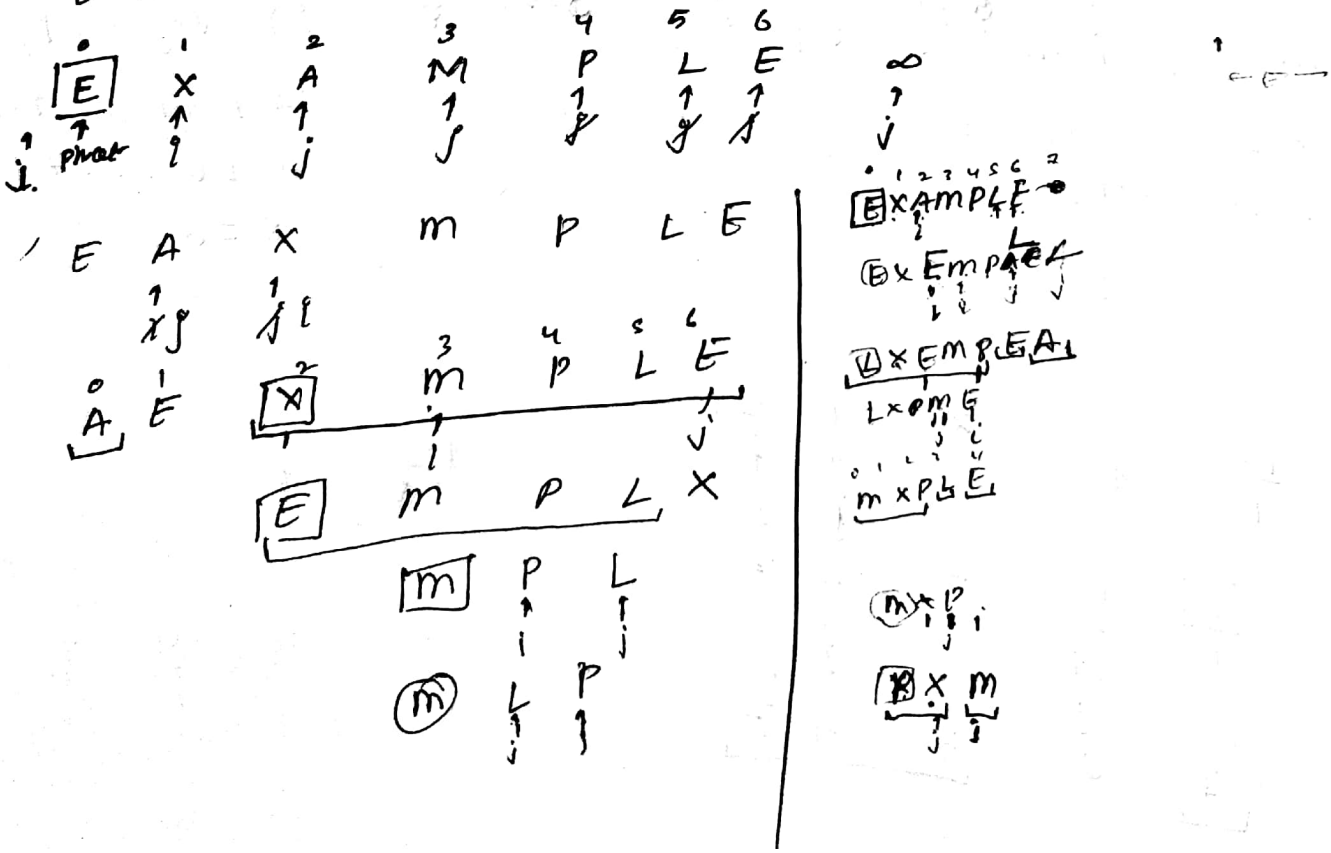


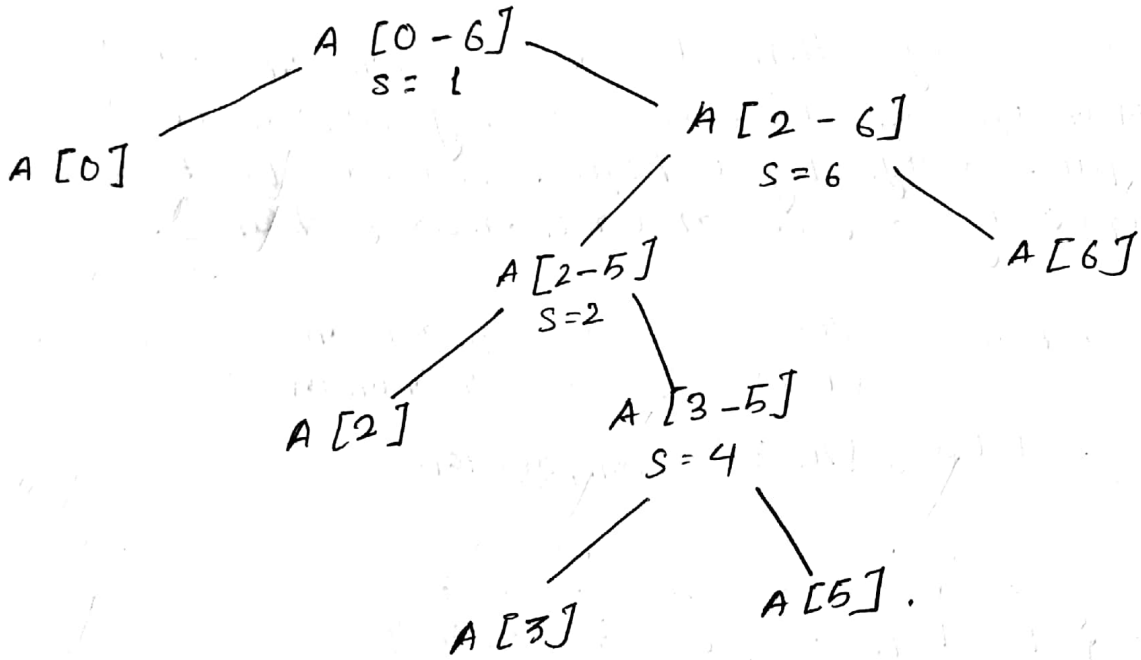


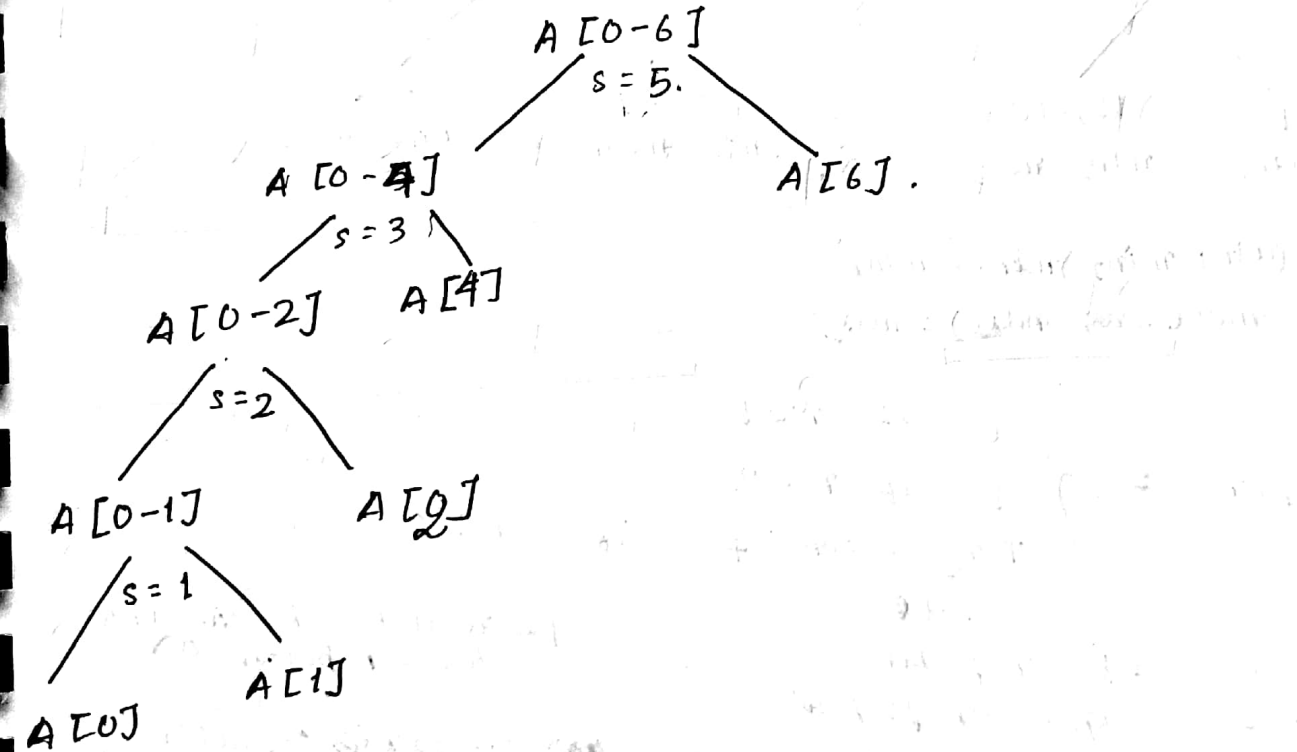


Q) Recursive call use to arrange the following data in decending order using Quick sort technique.

E X A M P L E



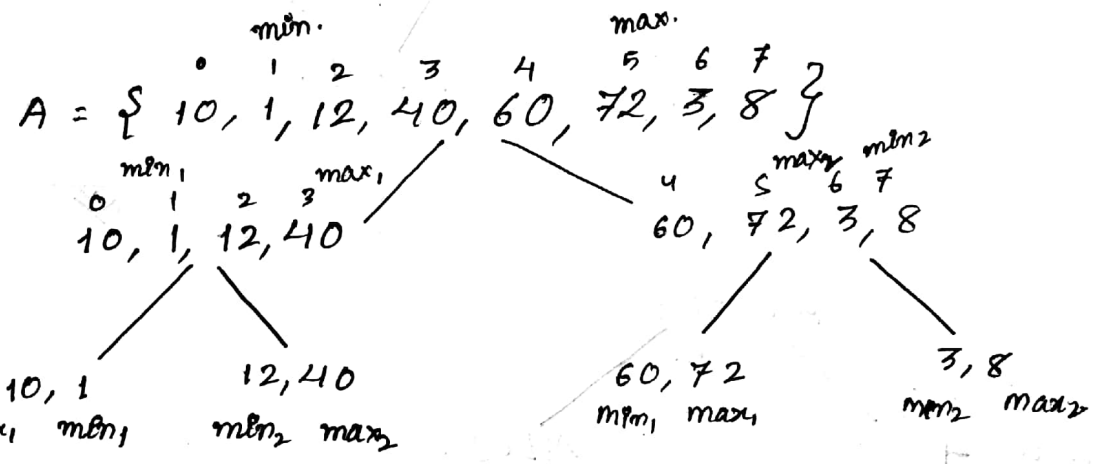




Qs. Write an algorithm to find maximum and minimum of 'n' data using divide and conquer designing technique and derive time complexity expression (min-max problem).

Brute force: $(n-1)$ To find maximum } or vice versa
$(n-2)$ to find minimum

Total $(2n-3)$ comparisions



compare (min_1, min_2) $min = min_1$

compare $max(max_1, max_2) = max_2$

$T(n/2) + 1$

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

$T(n/2)$ → left sub problem

$T(n/2)$ → right sub problem

$1 + 1$ → To find min & max of ~~left~~ right sub problem (smaller instance)

1 → To subdivide problem into two sub problems or

$$T(n) = 2T(n/2) + 2$$
$$= 2[2T(n/4) + 2] + 2$$
$$= 4T(n/4) + 4 + 2$$
$$= 2^2[2T(n/8)] + 8 + 4 + 2$$
$$= 2^3[T(n/8)] + 2^3 + 2^2 + 2^1$$
$$=$$

$$2^K [T(n/2^K)] + 2^K + 2^{K-1} + \cdots 2^1$$

maximum no. of comparisons will take place

$$\Phi \quad n = 2^K$$

$$\log_2 n = K \quad \text{——} (1)$$

$$= n[0] + n + n/2 + n/4 + n/8 + \cdots n/2^{\log n}$$

$$= 0 + n[1 + 1/2 + 1/4 + 1/8 + \cdots]$$

$$=$$

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

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

$$= 2[2T(n/4) + 1] + 1$$

$$= 2^2 [T(n/4)] + 2^2 + 2 + 1$$

$$= 2^2 [2(T(n/8) + 1] + 2$$

$$= 2^3 (T(n/8)) + 2^3 + 2^2 + 2 + 1$$

$$= 2^K (T(n/2^K)) + 2^K + \cdots 2^3 + 2^2 + 2 + 1$$

$$= n[T(1)] + n + (n-1) + \cdots$$

```
begin
    if  i = j
    then  begin  lo := a[i] ;  hi := a[i] end.
    else if  i = j - 1
        then begin if  a[i] < a[j]
                then begin  lo := a[i] ; hi := a[j] en
                else begin  lo := a[j] ; hi := a[i] en
             end
        else begin
                m := (i+j) div 2 ;
                mm (i, m, min1, max1) ;
                mm ( m+1, j , min2 , max2 ) ;
                lo := MIN ( min1, min2 ) ;
                hi := MAX ( max1, max2 )
             end
end.
```

MASTER THEOREM:

(To solve recurrence relation)

In calculating the complexity of recursive problem time required for breaking the main problem and combining the sub problem is of order $\Rightarrow O(n^d)$

↳ no. input instances

a = no. of instance used to solve the sub problem.

$$T(n) = a.T(n/b) + O(n^d).$$

b = total no. of divisions (No. of parts main problem is breaked)

Ex. in mergesort $n = 2$. (No. of sub problem in each iteration)

$T(n/b)$ ⇒ Time to solve one sub problem. problem.

Ex. in case of MS $T(n/2)$ to solve one sub problem $T(n/2)$ $T(n/2)$

$$T(n) = \begin{cases} O(n^d) & \text{if } a < b^d. \\ O(n^d \log_b n) & \text{if } a = b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

For Binary search:

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

here, $a = 1$; $b = 2$; $d = 0$.

It satisfy 2nd condition $a = 2^0$ } $1 = 2^0$ }.

$$T(n) = O(n^0 \log_2 n)$$

$$= O(\log_2 n).$$

Merge sort: $T(n) = 2T(n/2) + n$

$a = 2 ; b = 2 ; d = 1.$

$2 = 2^1$

$T(n) = O(n^d \log_b n)$

$= O(n \log_2 n).$ ✓

Quick sort:

Best case: $T(n) = 2T(n/2) + n.$

$a = 2; b = 2; d = 1.$

$T(n) = O(n^d \log_b n)$

$= O(n \log_2 n)$

worst case: $T(n) = T(n-1) + n.$

$a = 1 ; b = 1; d = 1$

$1 = (1)^1$

$= O(n^d \log_b n)$

$= O(n \log_1 n)$

Ans: 1) $T(n) = 4T(n/3) + n^2$

$a = 4; \ b = 3; \ d = 2.$

$\Rightarrow 4 > (3)^2$

$\Rightarrow 4 < 9$

$T(n) = O(n^2)$

2) $T(n) = 3T(2n/3) + n$

$a = 3; \ b = 3/2; \ d = 1$

$3 > (3/2)^1$

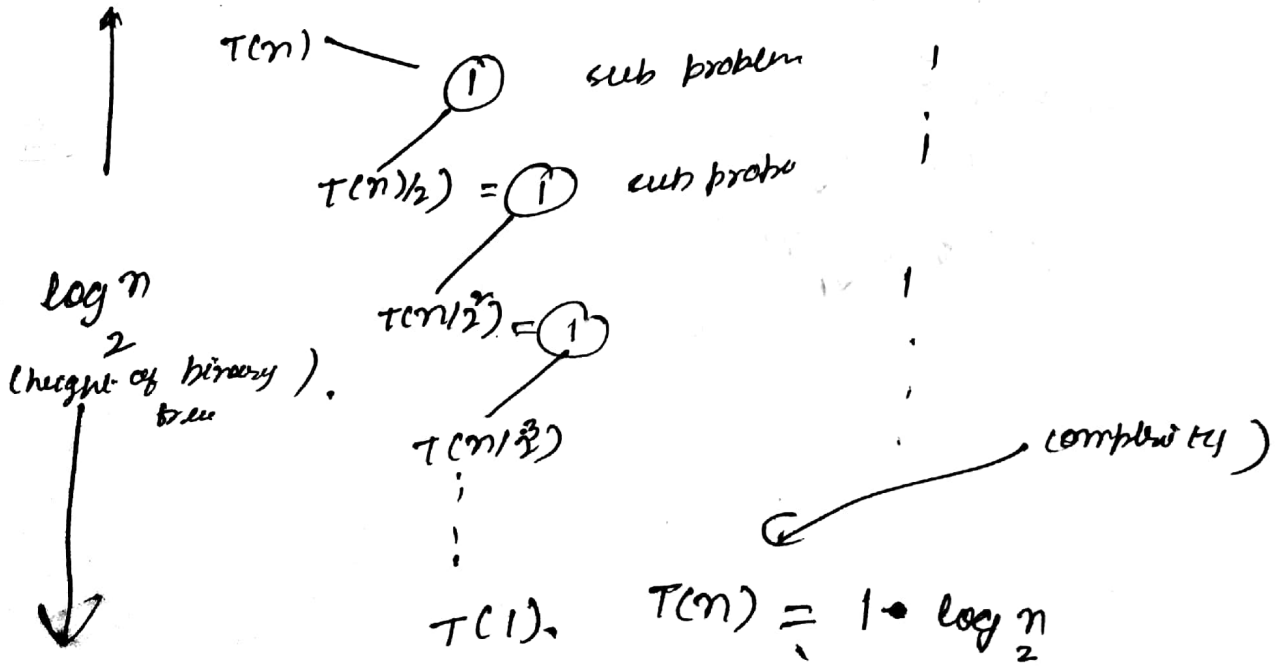
$3 > 3/2$

$= O(n^{\log_{3/2} 3})$

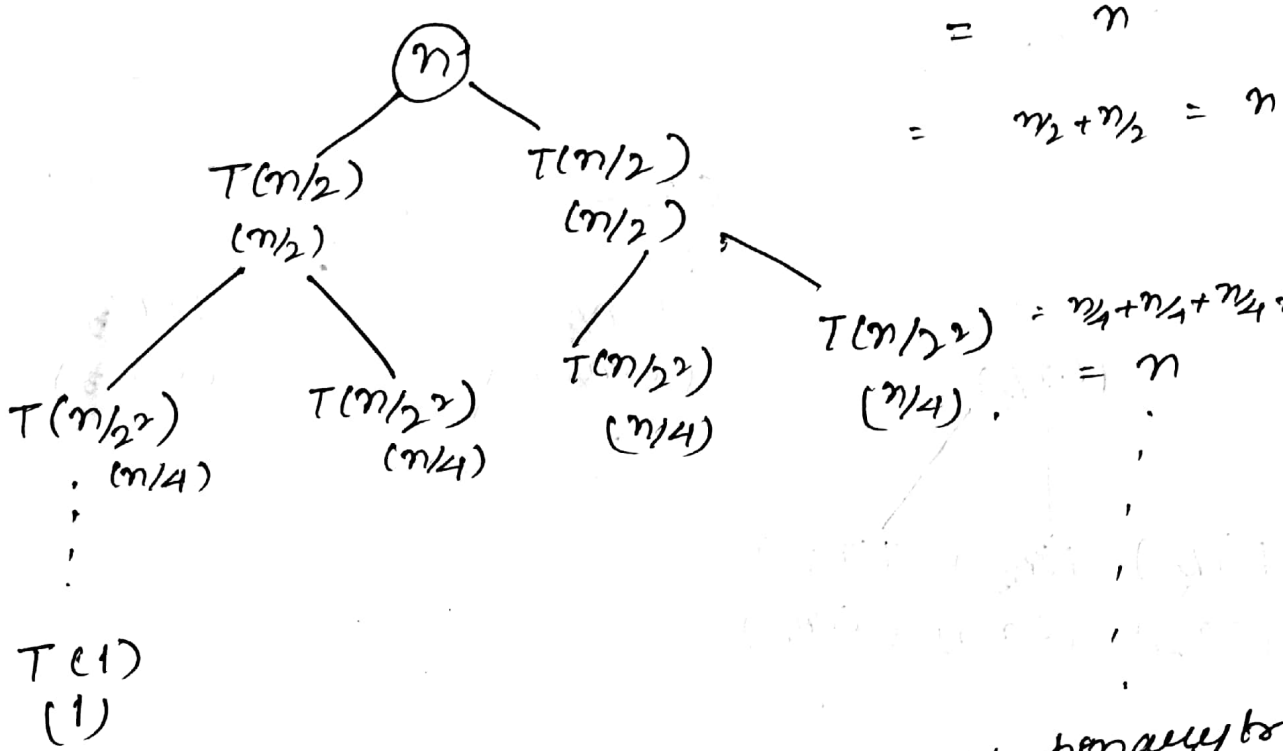
Recursive Tree Method:

When level sum is equal. take its sum (of one level) * height of binary tree. is the complexity.

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



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



$T(n)$ = sum of one level * height of binary tree

$= n \log_2 n$. ✓✓

Ans. Solve the following Recurrence Relation using a Recursive tree method.

(1) $T(n) = 3T(n/3) + c.n$

where, c is the constant and, n is the instance.

$c.n \rightarrow$ 'c' is constant
'n' is ords to combine and divide the problem into sub prob.
growth

$3.T(n/3) \rightarrow$ we are cutting the problem into 3 sub problem.

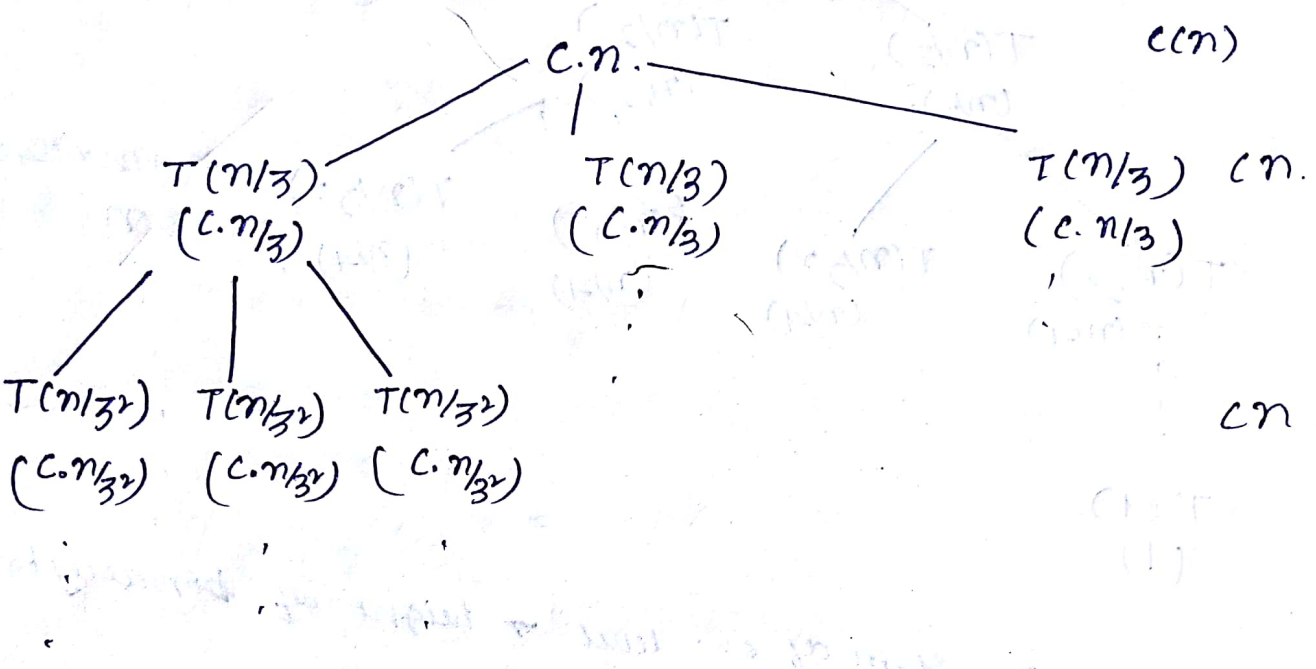
↘ To solve one sub problem

↓ No. of sub problem

(n)
T(n)

T(n/3)

$n \log_3 n$



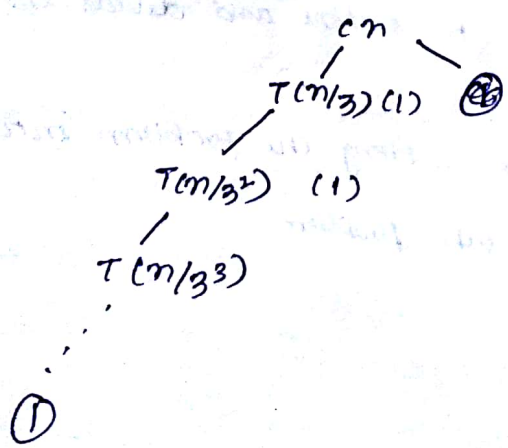
Level sum = c.n ; maximum height of Tree is $\log_3 n$

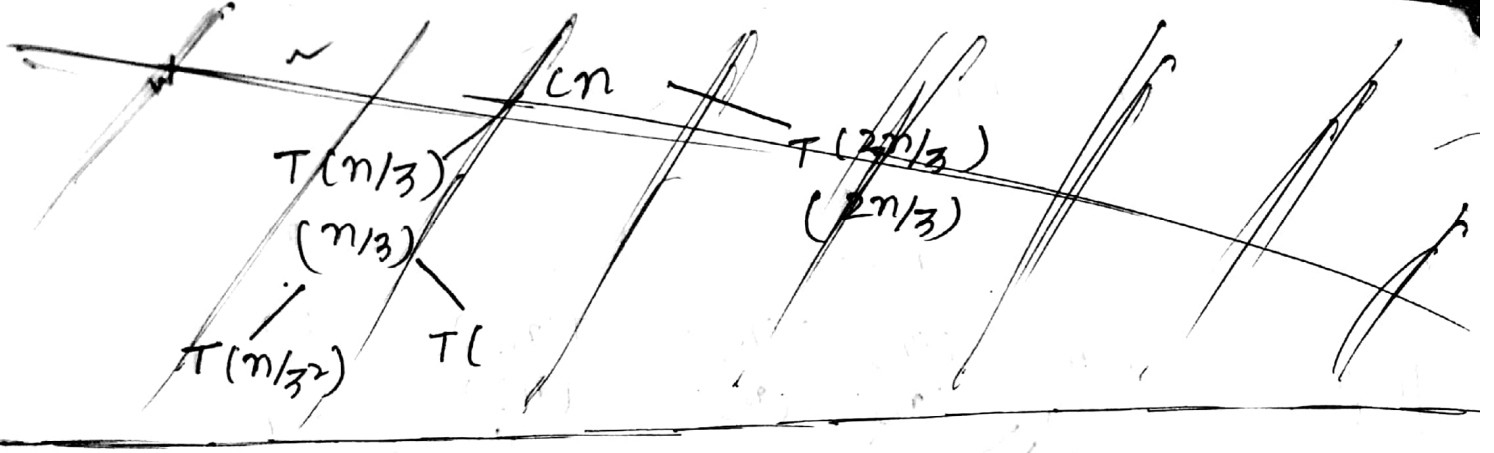
$T(n) = c.n * \log_3 n$

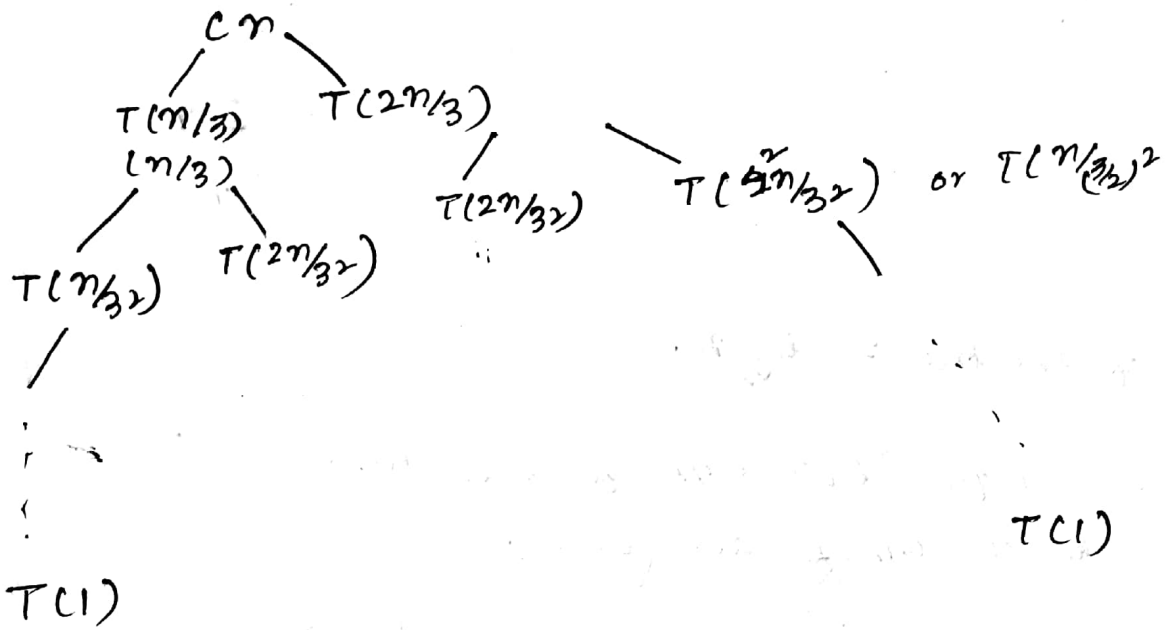
$= O(n \log_3 n)$.

Ex. $T(n) = T(n/3) + T(2n/3) + cn$.

for $T(n/3)$







max height of left sub tree = $\log_3 n$.

max height of right sub tree = $\log_{3/2} n$

Level sum = $c.n$.

When tree has different heights for right & left sub tree we will consider the tree which have more no. of sub problem. i.e more height

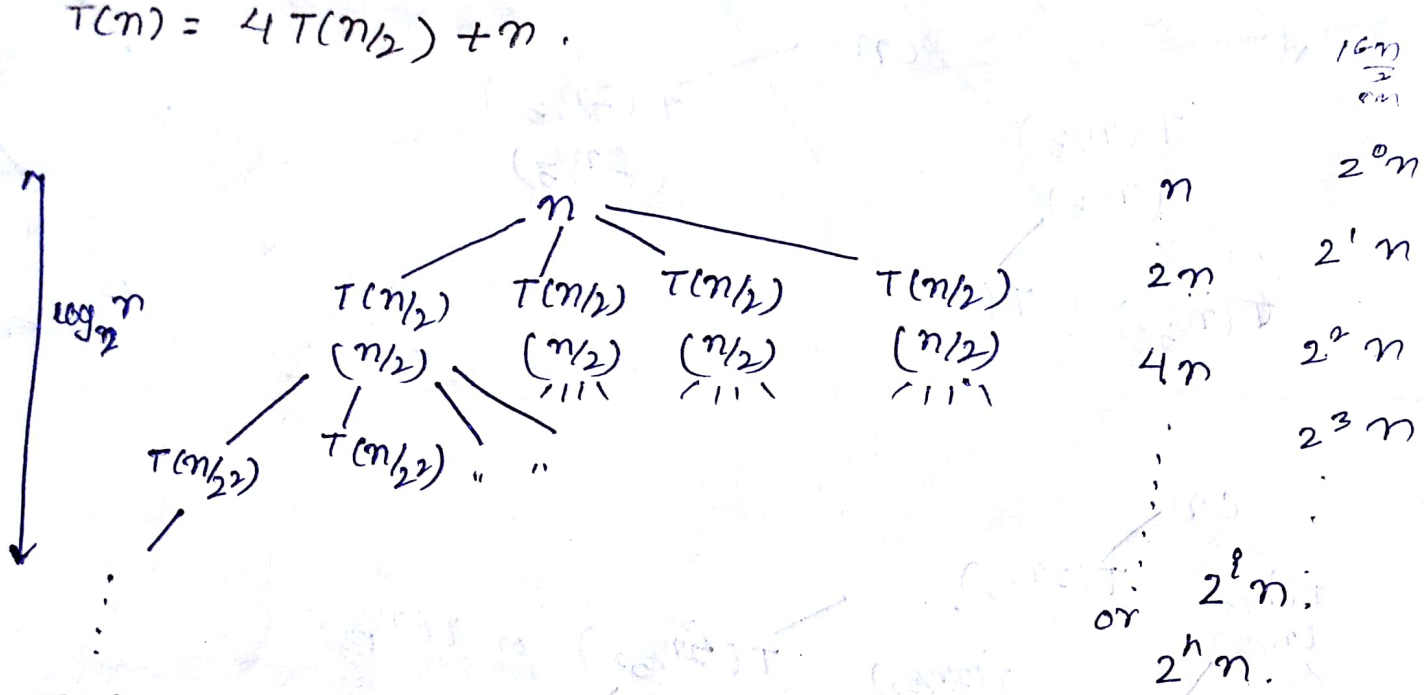
In this case $\log_{3/2} n > \log_3 n$.

$$T(n) = c * n \log_{3/2} n.$$

$$= O(n \log_{3/2} n).$$

Q.

$T(n) = 4T(n/2) + n$.



max. height of the tree = $\log_2 n$.

Here, we have unequal level sum for each level

∴ we will take the sum of all levels.

$$T(n) = (2^0 n + 2^1 n + 2^2 n + 2^3 n + \dots + 2^h n)$$

$$= n(1 + 2^1 + 2^2 + 2^3 + \dots + 2^h)$$

$$= n \cdot 2^{h-1}$$

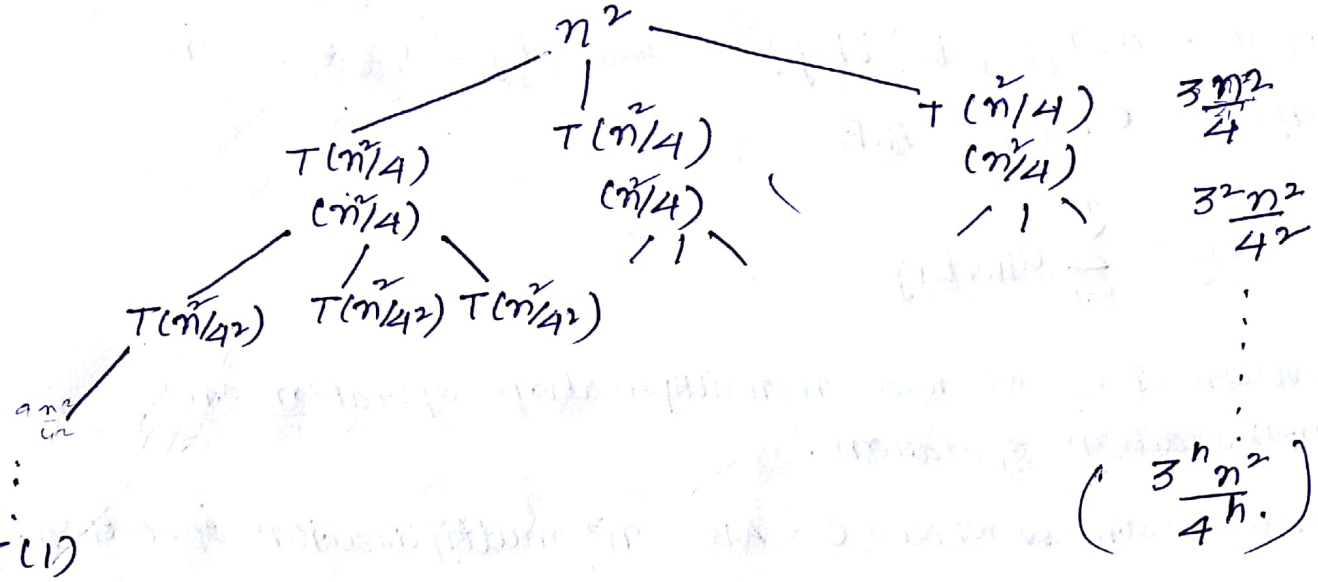
$$= n \cdot 2^{\log_2 n - 1}$$

$$= n \cdot \frac{2^{\log_2 n}}{2} = \frac{n}{2} n^{\log_2 2}$$

$$= n^2/2.$$

$$O(n^2).$$

4. $T(n) = 3T(n/4) + n^2$



maximum height = $\log_4 n$.

$$T(n) = n^2\left(1 + \frac{3}{4} + \frac{3^2}{4^2} + \cdots \frac{3^h}{4^h}\right).$$

$$= n^2 \cdot (3/4)^{h-1}$$

$$= n^2 (3/4)^{\log_4 n - 1}$$

$$= n^2 \frac{(3/4)^{\log_4 n}}{3/4}$$

$$= \frac{4}{3} n^2 \frac{3^{\log_4 n}}{4^{\log_4 n}}$$

$$= \frac{4}{3} n^2 \frac{3^{\log_4 n}}{n} = \frac{4}{3} n \cdot 3^{\log_4 n}$$

Matrix Multiplication :

Input = $A[a_{ij}]$, $B = [b_{ij}]$ where $i, j = 1, 2, 3, \ldots n$

Output = $C = [c_{ij}] = A.B$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

* To obtain 1 c_{ij} we need n multiplication operation and $(n-1)$ addition operation.

i.e. to obtain whole $C = A.B$ n^3 multiplication operation.

ALGORITHM : (Brute Force) .

```
for i ← 1 to n
  do for j ← 1 to n
       do  cij = 0
           for k ← 1 to n
               do  cij ← cij + aik · bkj
```

$$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} 1 \qquad (1 \rightarrow \text{standard time to solve one multiplication})$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} (n-1+1)$$

$$= n \sum_{i=1}^{n} \cdot (n-1+1)$$

$$= n^2 \sum_{i=1}^{n} (1)$$

$$= n^2 (n-1+1)$$

$$= n^3 .$$

$$= O(n^3).$$

Divide and conquer algorithm :

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix}_{2\times2} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}_{2\times2} \begin{bmatrix} e & f \\ g & h \end{bmatrix}_{2\times2}$$

$$C = A \cdot B.$$

$r = ae + bg$; $s = af + bh$

$t = ce + dg$; $u = cf + dh$

} 8 multiplication of $\frac{n}{2} \times \frac{n}{2}$ matrices
4 addition.

8 multiplications of $(n/2) \times (n/2)$ submatrices
4 addition of $(n/2) \times (n/2)$ submatrices.

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

- 8 → No. of submatrix
- $n/2$ → submatrix size
- $O(n^2)$ → work on addition of sub matrix

$a = 8$; $b = 2$; $d = 2$

$a > b^d$

$8 > 2^2$

$8 > 4$

$\Rightarrow O(n^{\log_2 8})$

$= O(n^{\log_2 2^3})$

$= O(n^3)$.

Strassen's Idea :

multiply 2×2 matrix with 7 recursive mults.

$P_1 = a.(f-h)$

$P_2 = (a+b).h$

$P_3 = (c+d).e$

$P_4 = d.(g-e)$

$P_5 = (a+d).(e+h)$

$P_6 = (b-d).(g+h)$

$P_7 = (a-c).(e+f)$

No. of multiplication = 7.

no. of add & sub = 18.

$r = P_5 + P_4 - P_2 + P_6$

$s = P_1 + P_2$

$t = P_3 + P_4$

$u = P_5 + P_1 - P_3 - P_7.$

$$T(n) = 7\,T(n/2) + O(n^2).$$

$a = 7 ; \; b = 2 ; \; d = 2.$

$a \quad b^d$

$7 > 2^2$

$7 > 4$

$$T(n) = O(n^{\log_2^7}) = O(n^{2.81})$$

Using strassens method no. of multiplication is reduced which makes time complexity from $O(n^3)$ to $O(n^{2.81})$.

ALGORITHM

```
ALGORITHM Strassen (n, A, B, C)
// n → matrix size, A and B are input matrices
// C → Resultant matrix
begin
    if n = 2 then
        Matrix-multiplication (A, B, C) // strassens method for
                                        // 2×2 multiplication.
    else
    begin
    Strassen (n/2, A11, B12 − B22, M1);
    strassen (n/2, A11 + A12, B22, M2);
    strassen (n/2, A21 + A22, B11, M3);
    strassen (n/2, A22, B21 − B11, M4);
    strassen (n/2, A11 + A22, B11 + B22, M5);
    strassen (n/2, A12 − A22, B21 + B22, M6);
    strassen (n/2, A11 − A21, B11 + B12, M7);
    end;
```

Imp:
** Numericals on Strassen formula //

19/sep/2018 → Last date 30/sept 2018

ARTICE NAME (TITLE)

→ AUTHOR(s) (YEAR of Publication)

Name :

Roll no :

SAP :

Branch :

SEM :

As 1. Define the area of problem considered in the article.

Ans: 4-5 statment (problem section)

As.2. Identify atleast 4-5 objectives considered in the article and explain.

Ans Each objective explained in 1-2 statment

As. 3. Explain the design methodology defined in the article (any one method).

Ans ALGORITHM

As 4. Identify any four performance parameter and explain in details. and discuss each performance parameter.

Ans. (In Result section)

As. 5. Explain any two related work.

Each related work → 5 statmen

1→ what problem they consider

2→ what methods they have proposed.

3→ Wat are the performance parameter they have consider

4→ Advantages

5→ Issues (min two).

GREEDY METHOD

When a problem contains a set of objects from that object finding optimal feasible solution.

GENERAL APPROCH:

→ Given a set of n inputs / objects.

→ Find a subset, called feasible solution, of the n input subject to some constraints and satisfying a given objective function.

→ If the objective function is maximized or minimized the feasible solution is optimal.

→ It is a locally optimal method.

ALGORITHM:

Step 1: choose an input from the input set, based on some criterion. If no more input exist.

Step 2: check whether the chosen input yields to a feasible solution. If no discard the input & go to step 1.

Step 3: Include the input onto the solution vector & update the objective function. Goto step 1.

```
Algorithm Greedy (C,S).
C → set of candidates // set of object / problem set.
S → solution set. // set of object that satisfy all feasible constraint
S = { }
while ( C != { } not solution(S) ) do
    X ← select(C)
    C ←→ C - {x}
    If (feasible (S U {x})) then
        S ← S U {x}
```

```
if (solution(x)) then
    return S;

Return "There is no solution"
end if
end if
end while.
```

Ans. Write an algorithm to design optimal file merging operation.

- Merge two files each has n and m elements, respectively:
  ⇒ takes $O(n+m)$.

- Given n files
  What's the minimum time needed to merge all n files?

Ex. $(f_1, f_2, f_3, f_4, f_5) = (20, 30, 10, 5, 30)$

$M_1 = f_1 \& f_2 \Rightarrow 20+30 = 50$
$M_2 = m_1 \& f_3 \Rightarrow 50+10 = 60$
$M_3 = M_2 \& f_4 \Rightarrow 60+5 = 65.$
$m_4 = m_3 \& f_5 \Rightarrow 65+30 = 95$
270 operations

} by brute force approach.

- Optimal merge pattern: (Greedy Method)

  Sort the listed file in ascending order.
  (5, 10, 20, 30, 30)

  merge first two files
  (5, 10, 20, 30, 30) → (15, 20, 30, 30) (15)

  merge next two files
  (15, 20, 30, 30) → (30, 30, 35) (35)

merge next two files

(30, 30, 35) ⟶ (35, 60) (60)

merge next two files

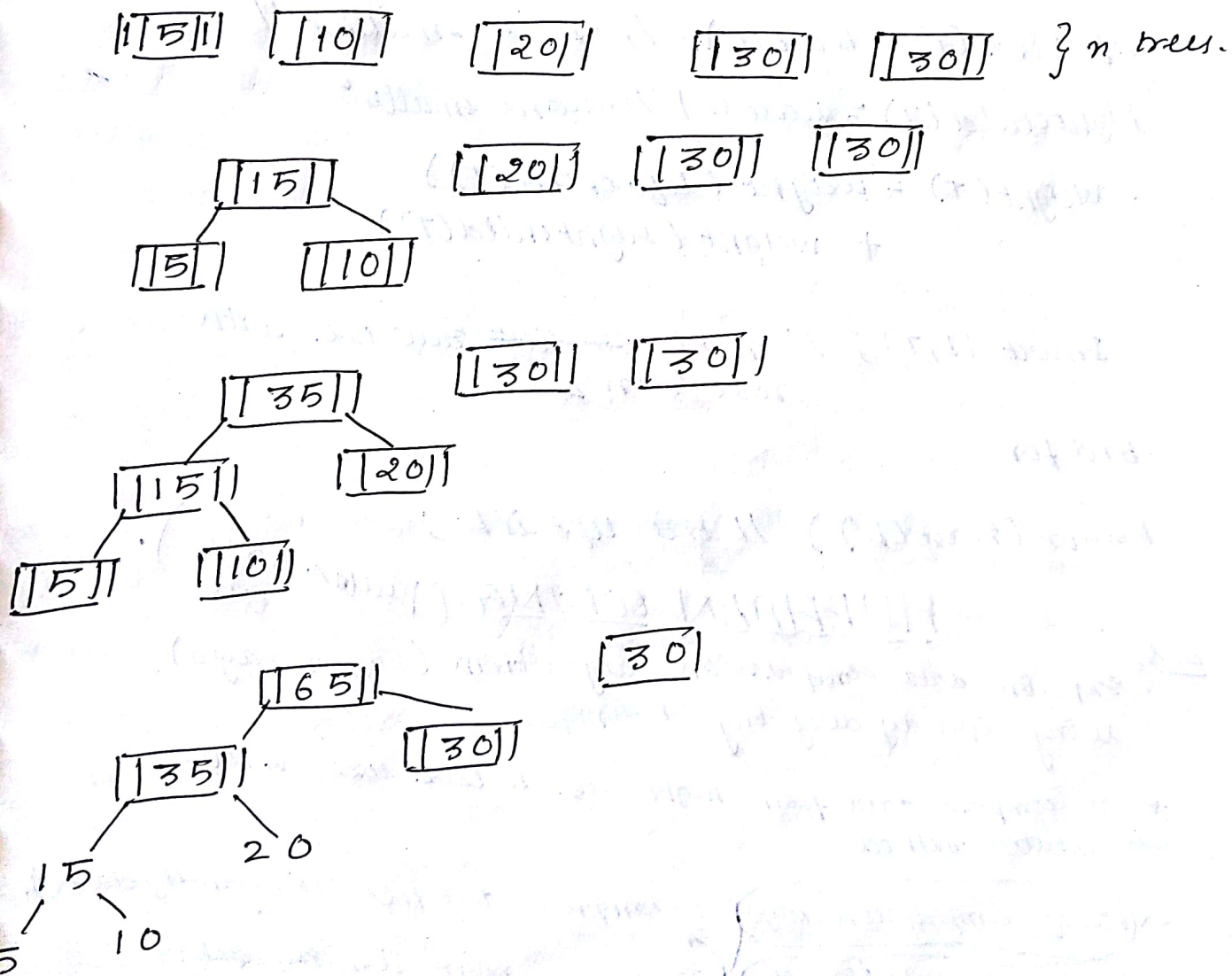
(35, 60) ⟶ (95) (95)

Total Time = 15 + 35 + 60 + 95

= 205. (combining time taken in each situation)

This is called 2-way merge.

Ex:



ALGORITHM :

- Least(L) : find a tree in L whose root has the smallest weight.

- Function : Tree (L, n)

```
Integer i ;
Begin
  For i=1 to n-1 do
  Getnode (T)   // create a node pointed by T
  Leftchild (T) = Least (L)  // first smallest //
  Rightchild (T) = Least (L) // second smallest
    Weight (T) = weight ( left child (T))
                + weight ( right child (T))

  Insert (L, T) ; // insert smallest new tree with
                     root T in L
  End for
  Return (least (L)) // tree left is L ;
```

HUFFMAN CODING (prefix free code).

Ans. Explain data compression algorithm (huffman algo) using Greedy designing technique.

* To compress data from higher size to lower size we have certain methods.

→ Lossy compression Algo — To compress text file (low density data).

→ Lossless compression Algo — To compress high density data (binary file) (audio, video, image etc)

* Memory optimization.

Huffman coding is a lossless data compression algorithm.

The idea is to assign variable length codes to input characters, length of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.

(less no. of bits for more frequent character and more no. of bits for less frequent character.)

It is a variable length encoding technique.

The variable length codes assigned to input characters are Prefixed codes, means the code (bit sequence) are assigned in such a way that the code assigned to one is not prefix of code assigned to any other character.

* In fixed length encoding technique. there may be circumstances where we can loose data. if a character is occuring again and again.

* When we assign any code to any character then it can't be a prefix of any other number

* We have to find prefix free code.

To generate prefix free code.

1. We can construct a Binary Tree of the Text (Huffman Tree)
2. We need to identify the code word for every characters.
3. In Huffman tree leaf node contain codes for a corresponding character

Tree construction
Traversal of tree from Root to leaf } Steps for encoding.

Ex.

character	Freq.
a	5
b	9
c	12
d	13
e	16
f	45

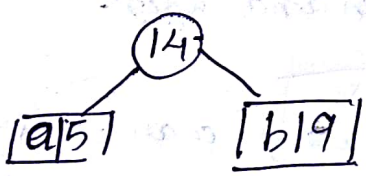
* before encoding. we have to $ the data will take 100 bytes of memory.

Step 1 : Build a min heap that contain 6 nodes were each node represents root of a tree with singe node.

(a) 5 (b) 9 (c) 12 (d) 13 (e) 16 (f) 45.

2 : Extract two min freq. heap tree add a new internal node with freq: 5+9 = 14

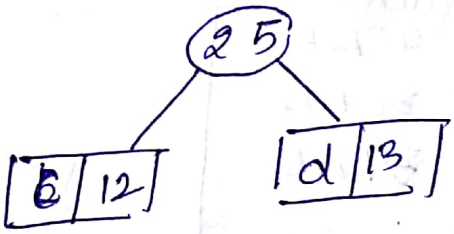
save sum of freq. of 2 min heap node as root of those 2.



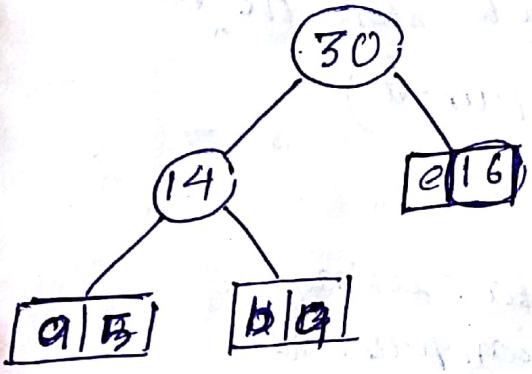
Now min heap contains 5 nodes where 4 are roots of. trees with sing element.

c 12
d 13
internal node 14
e 16
f 45.

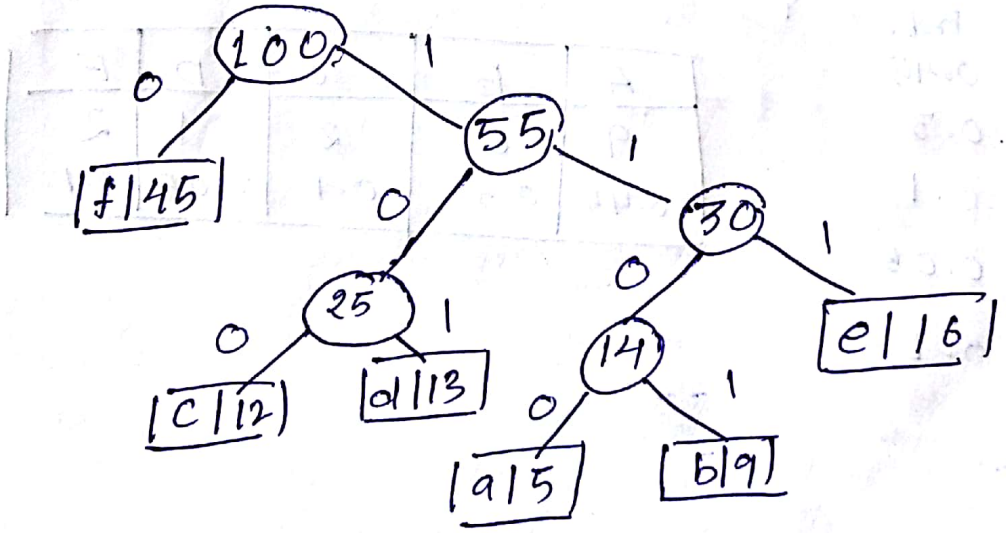
Extract next two min heapp.



Internal Nodes 14
e 16
Internal Nodes 25
f 45.



Repeat the process till we get a single tree.



(put 0 for left sub tree and 1 for right sub tree of every nodes in a tree.)

codes are as follows.

char	codes	bits required	No. of bits X frequency
f	0	1 x 45 = 45 bits	
c	100	3 x 12 = 36 bits	
d	101	3 x 13 = 39 bits	
a	1100	4 x 5 = 20 bits	
b	1101	4 x 9 = 36 bits	
e	111	3 x 16 = 48 bits	
		224 bits.	

are required to store the complete file:

(before encoding 800 bits were required)

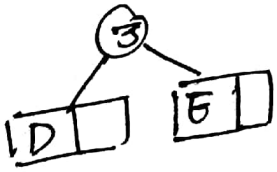
Ex.
Encode and decode the following data using Huffman algo. Find the data compression ratio to encode given data.

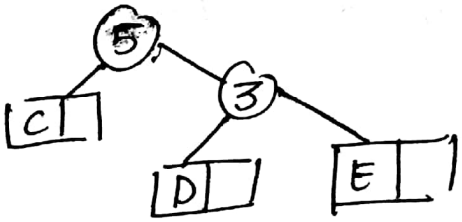
AABBCAADEEAAABBBAACBB

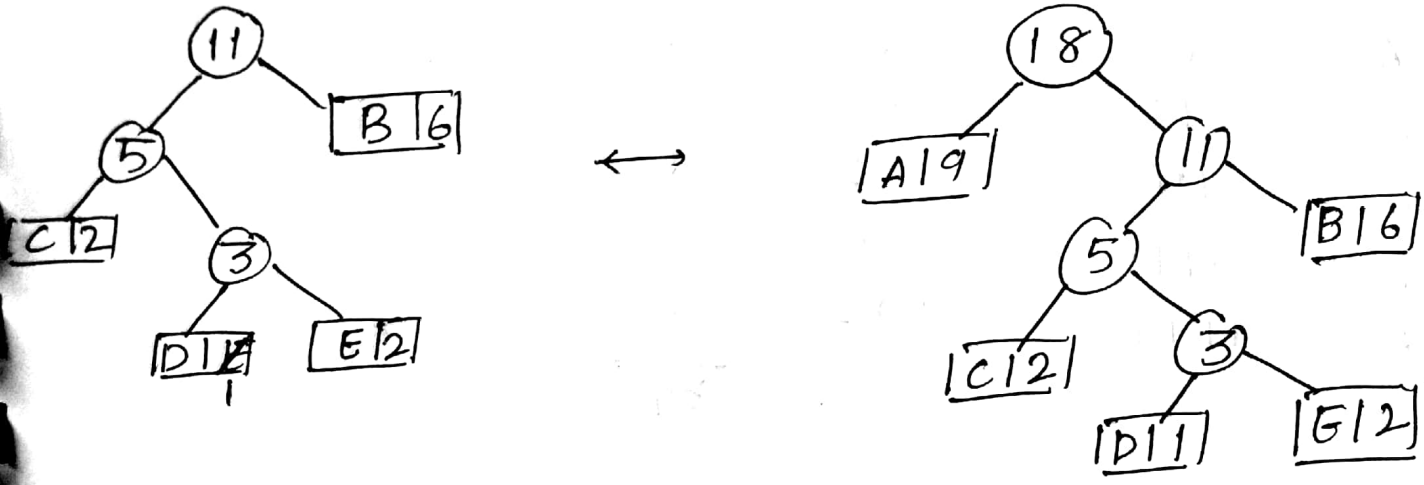
		Prob.
A	9	0.45
B	6	0.3
C	2	0.1
D	1	0.05
E	2	0.1
	160 bits.	

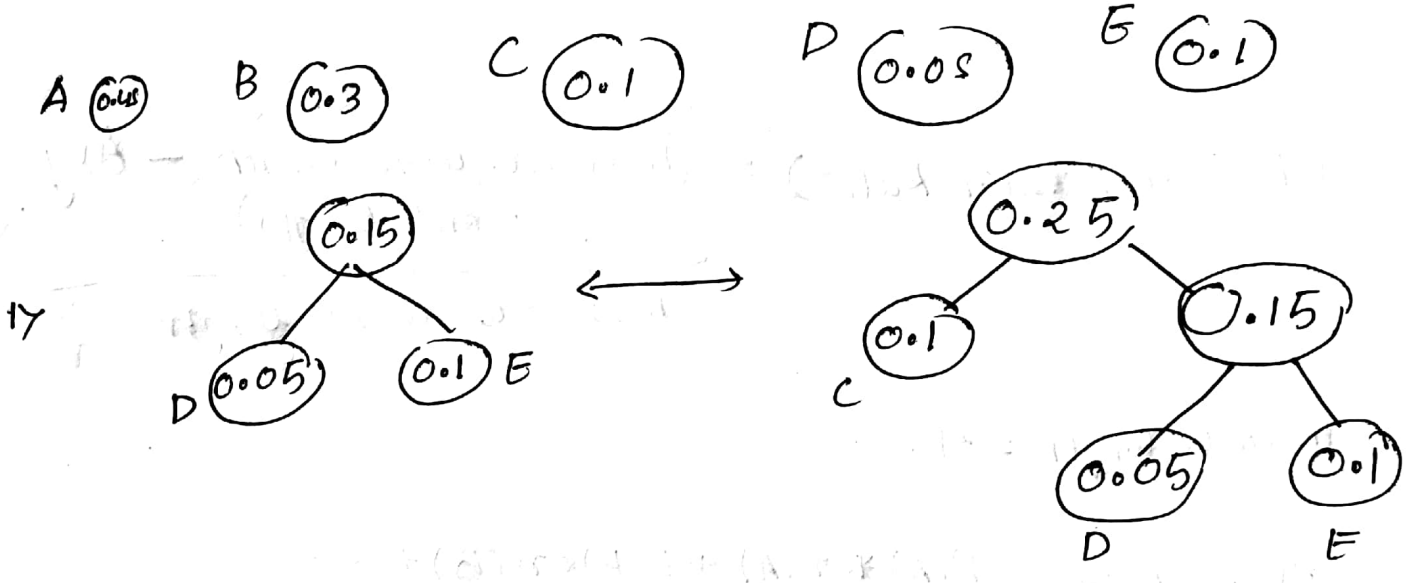
A	B	C	D	E
9	6	2	1	2
0.45	0.3	0.1	0.05	0.1

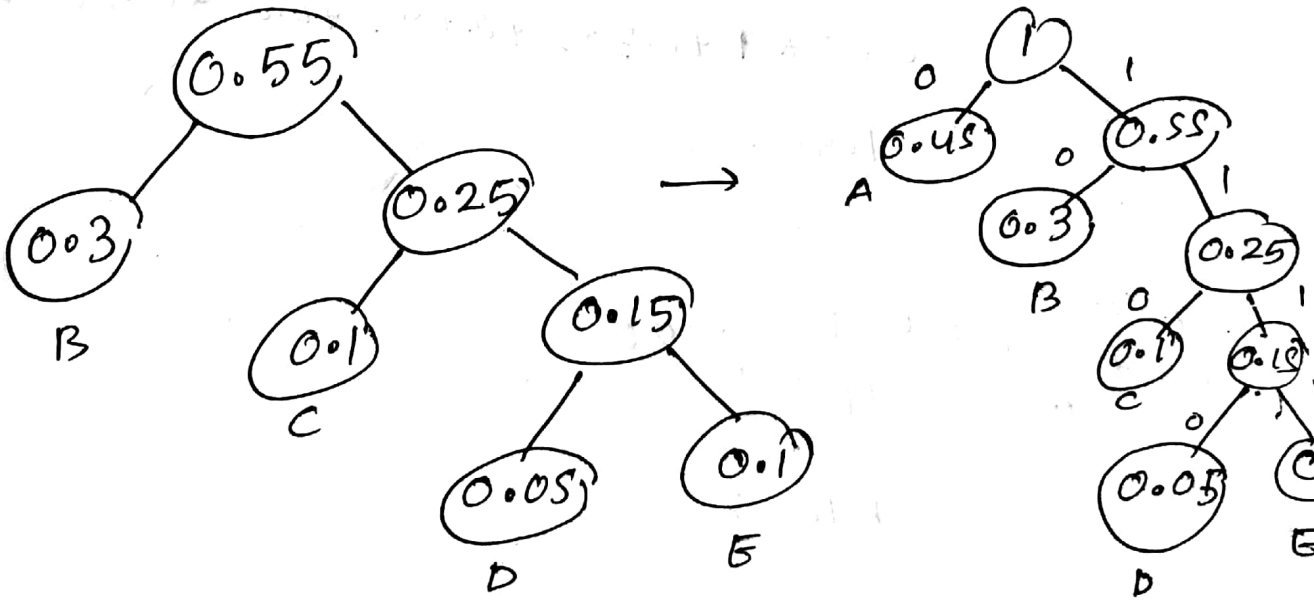
(D)$_1$ (E)$_2$ (C)$_2$ (B)$_6$ (A)$_9$.











	Code	Total Bits	
A	0	1x9 =	9
B	10	2x6 =	12
C	110	3x2 =	6
D	1110	4x1 =	4
E	1111	4x2 =	8
			39

A A B B C A A D E E A A A B B B A A C B

0 0 10101100011101111111000101010100111010.

$$CR\ (\text{Compression Ratio}) = \frac{(\text{max code word length}) - (\text{Avg code word length})}{\text{max code word length}}$$

max code length = 4.

$$\text{Avg. length} = P(A)*n(A) + P(B)*n(B) + \cdots$$

$$= 0.45*1 + 0.3*2 + 0.1*3 + 0.05*4 + 0.1*4$$

$$= 1.95.$$

$$CR = \frac{4-1.95}{4} \times 100$$

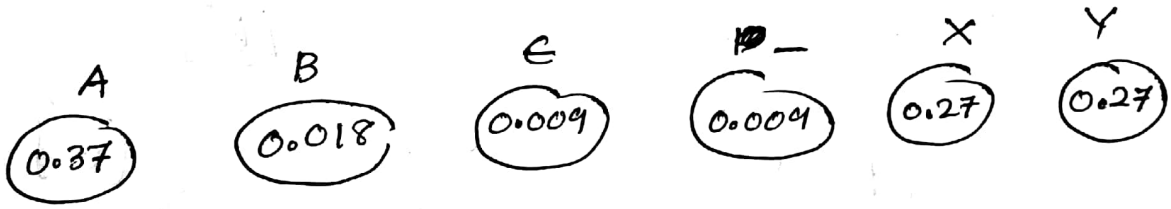
$$= 51.25\%.$$

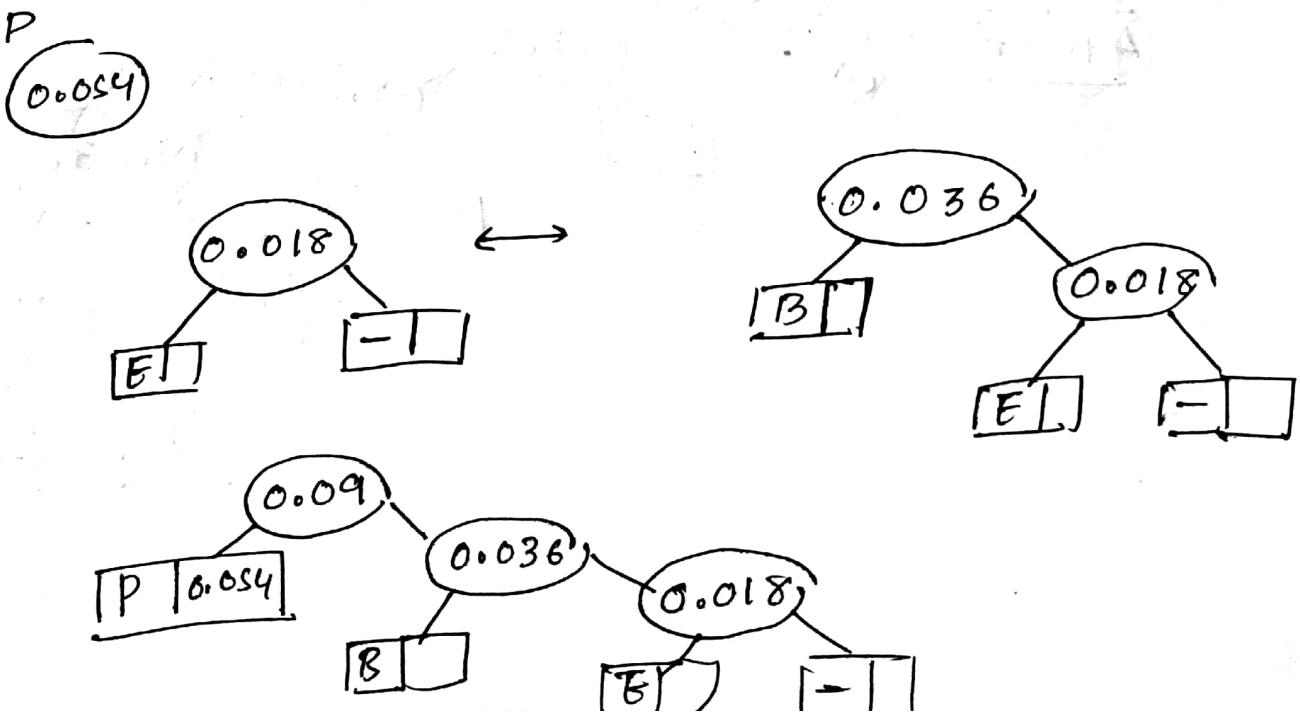
4° Find compression ratio for following instance of problem using Huffman coding algorithm

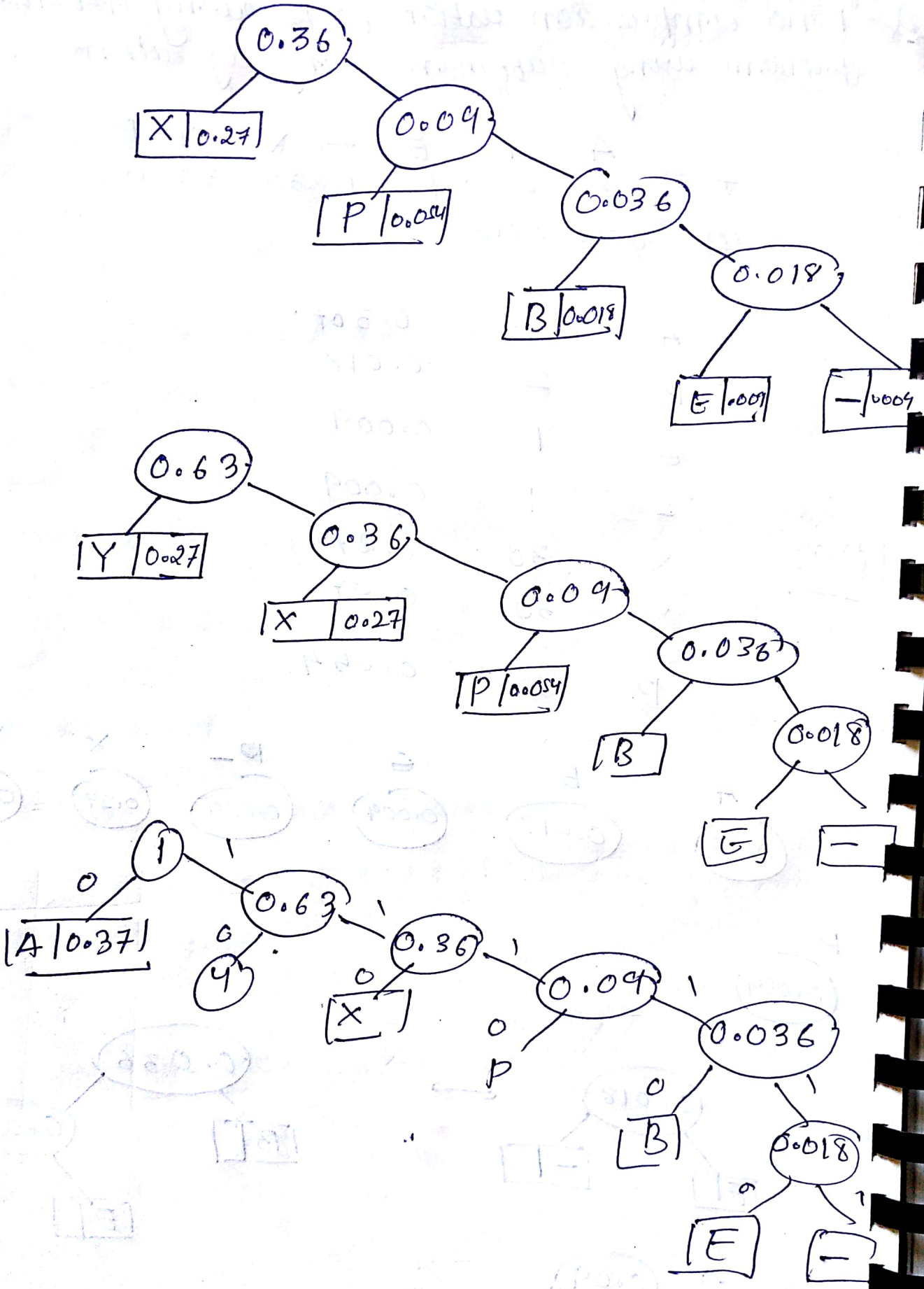
	A	B	E	–	X	Y	P
f	40	2	1	1	30	30	6
P(f)	0.36	0.018					

A	40	0.367
B	2	0.018
E	1	0.009
–	1	0.009
X	30	0.27
Y	30	0.27
P	6	0.054

100







A → 0	1 × 40	40
Y → 10	2 × 30	60
X → 110	3 × 30	90
P → 1110	4 × 6	24
B → 11110	5 × 2	10
E → 111110	1 × 6	6
— → 111111	1 × 6	6
		236

max bit length = 6.

Avg. = 0.37 * 1 + 0.27 * 2 + 0.27 * 3 + 0.054 * 4
+ 0.018 * 5 + 0.009 * 6 + 0.009 = 6.

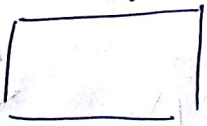
= 2.134.

$$CR = \frac{6 - 2.134}{6} \times 100$$

= 64.43%

4) Solve the following instance of knapsack problem using Greedy designing Technique.

(knapsack : memory space)



knapsack Problem:

n → list of objects (0 - - n-1) or (0 - - - n).

Profit of i^{th} object P_i

weights of i^{th} object w_i

(P_i, w_i)

(P_1, w_1)	10	2
(P_2, w_2)	30	2
(P_3, w_3)	10	2

Total capacity = 5

Aim is to include the objects (subset) in the given knapsack capacity. which gives (subset) gives maximum profit.

If we use brute force technique. we have to check max profit of each subset (possible subset) of the given problem. i,e 2^n comparisons. (Not efficient)

Knapsack :

Fractional

Objects can be included partially.

Discrete

If object is completely considered in the final solution. (0/1 knapsack)

if weight of object is > the remaining knapsack, then we can't include it in knapsack hence there is wastage of memory.

→ In dynamic technique

Ex. $n = 5$, $W = 100$.

n	w	v
1	10	20
2	20	30
3	30	66
4	40	40
5	50	60

We have to find most feasible solution where we have maximum profit.

0 1) $\sum_{i=1}^{n} w_i > W$

It means their are some set of solutions we have to find most efficient solution or subset.

if $\sum_{i=1}^{n} w_i \leq W$ all objects can be included)

11) $\sum_{i=1}^{n} x_i v_i$ here x_i is solution vector / bit vector

where, $x_i = 0$ or 1

$x_i = 0 \rightarrow i^{th}$ element/object not included

$x_i = 1 \rightarrow i^{th}$ element/object is included.

Step 1: Find the ratio V/W or V:W (profit/value : weight)

n	w	v	v/w
1	10	20	2
2	20	30	1.5
3	30	66	2.2
4	40	40	1
5	50	60	1.2

Step 2: Arrange V/W ratio in descending order.

n	w	v	v/w
3	30	66	2.2
1	10	20	2
2	20	30	1.5
5	50	60	1.2
4	40	40	1

Step 3: Select the object and test which includes in the final solution or not ($w_i \leq W$).

$X' = \{1, 1, 1, 0.8, 0\}$.

$w_3 = 30 < W_{(100)}$ ✓

$w_1 = 10 < W_{(70)}$ ✓

$w_2 = 20 < W_{(60)}$ ✓

$w_5 = 50 \not< W_{(40)} \Rightarrow \frac{40}{50} = 0.8$ { it means only 80% of 5th object is considered.

arrange X' in correct order.

$X = \{1, 1, 1, 0, 0.8\}$

$$P = \sum_{i=1}^{5} X_i V_i$$

$= (20 \times 1 + 30 \times 1 + 66 \times 1 + 0 \times 40 + 0.8 \times 60)$

$= (20 + 30 + 66 + 48)$

$= (164) \rightarrow$ max possible profit.

4a. Find the optimal solution for the following instance of knapsack problem using greedy D.T.

$n = 5; \quad W = 7.$

$w_i = \{1, 2, 3, 1, 2\}$ $\quad$ $v_i = \{30, 20, 10, 40, 50\}$

n	w	v	v/w
1	1	30	30
2	2	20	10
3	3	10	3.33
4	1	40	40
5	2	50	25.

n	w	v	v/w.
4	1	40	40
1	1	30	30
5	2	50	25.
2	2	20	10
3	3	10	3.33

$x_i' = \{1, 1, 1, 1, 0.33\}$

$x = \{1, 1, 0.33, 1, 1\}$

$$P = \sum_{i=1}^{5} x_i v_i$$

$= 1 \times 30 + 1 \times 20 + 0.33 \times 10 + 1 \times 40 + 1 \times 50$

$= 30 + 20 + 3.3 + 40 + 50$

$= 143.3$

4s. $n = 3$; $w = \{18, 15, 16\}$ $p = \{25, 24, 15\}$ $W = 20$.

n	w	p	p/w
1	18	25	1.39
2	15	24	1.6
3	16	15	0.9375

n	w	p	p/w
2	15	24	1.6
1	18	25	1.39
3	16	15	0.9375

$x_i' \{1, 0.27, 0\}$ $x = \{0.27, 1, 0\}$

20 - 15

5

$\frac{5}{16} \times 100$

$$P = \sum_{i=1}^{3} p_i x_i$$

$= 0.27 \times 25 + 24 \times 1 + 0$

$= 6.75 + 24$

$= 30.75$

```
ALGORITHM Knapsack(wi, Vi, W)
// wi → weights, Vi → Profits, W → max capacity.
  for(i ← 1 to n) do
     x[i] = 0, weight = 0
     while (weight < W) do
        if (weight + w[i] ≤ W) then
              x[i] = = 1 ;
              weight = weight = w[i];
              else
                   x[i] = ((W - weight)/w[i]);
                   weight = w;
              // weight = weight + w[i];
           end if
       end while
  end for

  p = 0
  for (i ← 1 to n) do
       p = p + x[i] * W[i] ;
  end for.
  return p
```

Dynamic Programming Technique :

To solve overlapped subproblem we have to call a sub problem many time; so it requires a lot of time. ∴ we store the solution of sub problems in memory and retrive it when required.

We need extra memory to store the solution of sub problems.

4s. Find max profit using dynamic programming technique.

$n = 4$; $W = 6$

n	W	V
1	3	20
2	2	30
3	2	40
4	3	10

We have to build a matrix with $[P(i,j)]$.

n (colums) = $(W+1)$ & no. (rows) = $n+1$.

(i) ↓	n	w	V	0	1	2	3	4	5	6. (j) →
	0			0	0	0	0	0	0	0
	1	3	20	0	0	0	20	20	20	20
	2	2	30	0	0	30	30	30	50	50
	3	2	40	0	0	40	40	70	70	70
	4	3	10	0	0	0	40	70	70	70.

$$P(i,j) = \begin{cases} 0 & \text{if } i=0 \text{ OR } j=0. \\ P(i-1,j) & \text{if } j < w(i) \\ V_i + \max\{P(i-1,j), P(i-1, j-w_i)\} & \text{if } j >= w(i). \end{cases}$$

$$\max\{P(i-1,j),\ V_i + P(i-1, j-w_i)\}.$$

$40 + P[3,$

i → object index

j → max. knapsack capacity

$P(1,3) = V_1 + \max\{P(0,3) - P(0,0)\}$
$= 20 + \max\{0, 0\}$
$= 20.$

$P(1,4) = V_1 + \max\{P(0,4), P(0,1)\}$
$= 20 + 0$
$= 20$

$P(2,2) = V_2 + \max\{P(1,2) - P(1,0)\}$
$= 30 + \max\{0, 0\} = 30$

$P_{2,3} = V_2 + \max\{P(1,3), P(1,1)\}$

$= 30 + \max\{20, 0\}$

$= 30 + 20 = 50.$

$P(2,4) = V_2 + \max\{P(1,3), P(1,2)\}$

$= 30 + \max\{20, 0\}$

$= 50$

$P(2,5) = V_2 + \max\{P_{1,}$

for $P(3,3) = \max\{P(2,3), V_3 + P(2,1)\}$

$= \{30, 40+0\}$

$= 40$

$P(3,4) = \max\{P(2,4), V_2 + P(2,2)\}$

$= \{30, 40+30\}$

$= 70$

$P(3,5) = \max\{P(2,5), V_3 + P(2,3)\}$

$= \{50, 40+30\}$

$P(4,3) = \max\{P(3,3), 10 + P(3,0).$

$= \{40, 10\}$

$= 40$

$P(4,4) = \max\{P(3,4), 10 + P(3,1)\}$

$= \max\{70, 10+0\}$

$= 70$

$P(4,5) = \max\{P(3,5), 10 + P(3,2)$

$\{70, 10+40\}$

$= 70$

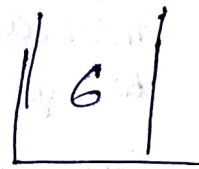
$P(4,6) = \max\{P(3,6), 10+P(4,3)$

$70, 10+40$

$= 70.$

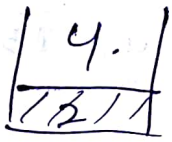
Compare P(last row, last colum) and P(last row, last col - 1) if they are same.
Don't include the object.

P(4,6) = P(3,6) [both are equal] ✗



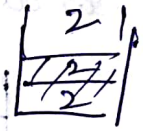
Dont include 4th object.

P(3,6) ≠ P(2,6) [Not equal] ✓



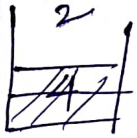
include 3rd object

P(2,4) ≠ P(1,4) [Not equal] ✓



include 2nd object.

P(1,2) = P(0,2) [Not equal] ✗



don't include object 1st.

$$X_p = \{0, 1, 1, 0\}$$

Disadvantage:

→ memory wastage
→ If knapsack prob have higher WI the matrix will be difficult to built.
→ In some instances we are saveing same values in subsequent columns i.e wastage of memory.

Find the optimal solution for the following instance of knapsack problem using discrete and fractional methods outlining method & find the best method to get the optimal solution.

$n = 4$; $W =$

$w_i = \{3, 2, 4, 1\}$ $\quad$ $V_i = \{100, 20, 60, 40\}$

Fractional method:

n	w_i	V_i	V_i/w_i
1	3	100	33.33
2	2	20	10
3	4	60	15
4	1	40	40

n	w_i	V_i	V_i/w_i
4	1	40	40
1	3	100	33.33
3	4	60	15.
2	2	10	10.

$x_i' = \{1, 1, 0.25, 0\}$ $\quad$ $x_i = \{1, 0, 0.25, 1\}$.

$P_i = \sum x_i V_i = (1 \times 100 + 0 \times 20 + 0.25 \times 60 + 40 \times 1)$

$= 100 + 0 + 15 + 40$

$= 155$

Descrete methods :

n	wi	Vi	Vi/wi	0	1	2	3	4	5
0				0	0	0	0	0	0
1	3	100	33.33	0	0	0	100	100	100 ✓
2	2	20	10	0	0	20	100	100	120
3	4	60	15	0	0	20	100	100	120
4	1	40	40	0	40	40	100	140	140 ✓

$i = p + 3$

$j = x$

$P(3,4) = \max\{P(2,4),\ V_i + P(2,0)\}$

$P(3,5) = \max\{P(2,5),\ 60 + P(2,1)\}$

$P(4,1) = \max(P(3,1),\ 40 + P(3,0))$

$P(4,2) = \max\{P(3,2),\ 40 + P(3,1)\}$

$\{20, 40\}$

$P(4,3) = \max\{P(3,3),\ 40 + P(3,2)\}$

$100,\ 40 + 20$

$P(4,4) = \max\{P(3,4),\ 40 + P(3,3)\}$

$100,\ 40 + 100$

$P(4,5) = \max\{P(3,5),\ 40 + P(3,4)\}$

$100,\ 40 +$

$P(2,5) = \{P(1,5),\ 20 + P(1,3)\}$

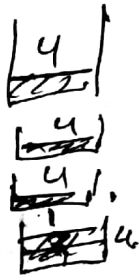
$P(3,5) = \{P(2,5),$

$P(4,5) \neq P(3,5)$

$P(3,4) = P(2,4) \times$

$P(2,4) = P(1,4)$

$P(1,4) \neq P(0,4)$



40 ✓ $x_i = \{1, 0, 0, 1\}$.

$P_i = \Sigma x_i \times v_i = (1 \times 100 + 0 + 0 + 1 \times 40)$

$= 140.$

fractional is more efficient. at Periodic it larger and no wastage of knapsack capacity.

ALGORITHM Knapsack (Vi, Wi, n, W)

// Vi → profits, Wi → weight, n → no. of objects

// W → max knapsack capacity.

```
for (j ← 0 to W) do
    C[0,j] = 0 // for first row to be zero
for (i ← 0 to n) do
    C[i,0] = 0 // to make first column zero.
for (j ← 1 to W) do
    if (Wi > j) Then
     C[i,j] = C[i-1,j]
    else
    C[i,j] = max{ C[i-1,j], Vi + C[i-1,j-Wi] }
  end if
end for
end for
    return C[i,j]
```

$$T(n) = \sum_{i=0}^{n} \sum_{j=0}^{W} 1$$

$$= T(n \cdot W$$

$$P(i,j) = \begin{cases} 0 & \text{if } i=0 \text{ || } j=0 \\ P(i-1,j) & \text{if } (w_i > j) \\ \max\{P(i-1,j), P_i + P(i-1, j-w_i)\} & \text{if } (w_i > j) \end{cases}$$

4. construct minimum cost spanning tree (MST) using greedy designing technique.

Graph (V, E)

① No. of components

Dense (no. V < no. E)

Sparse (no. V > no. E)

② Direction

Directed (specific direction from one node to another)

undirected (-do-) (directions is not specified.)

← || — do —

— do —

③ weight

weighted (when weight is assigned to any edge)

unweighted -do- (Not assigned)

— do —

④ connectivity.

connected (when we have a node which is connected to every remaining node)

disconnected (-do)

— do —

⑤ cycle.

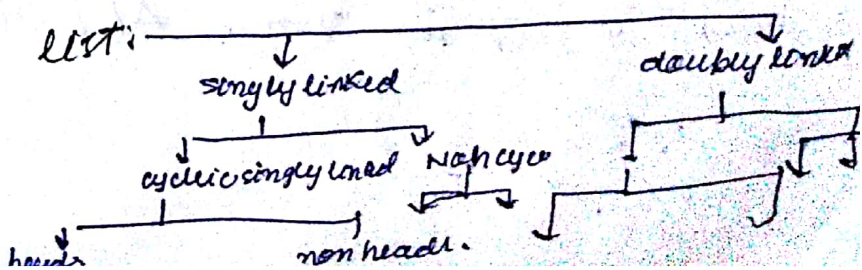
cyclic (when we start from one node and come back to same node after traversing some set of edges)

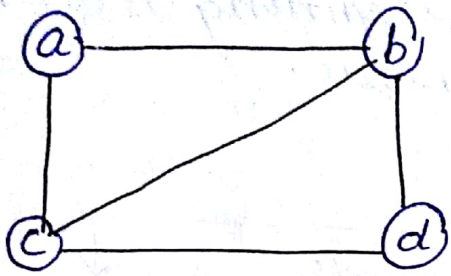
Acyclic.

To represent any type of graph we have two methods.

1) Adjacency matrix.

2) Adjacency Linked list:

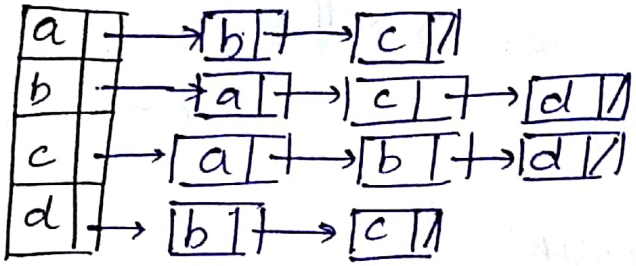




cyclic Unweighted Undirected connected Dense Graph.

or

No. nodes = 4 so, create 4 heads nodes



$4 \times 4 = 16$ B

16 B

$2 \times 10 + 2 \times 10$

$20 + 20$

66

or

$14 \times 2 + 14 \times 2$

$= 56.$

$4 \times 4 = 16$ (for heads nodes)

$(2 \times 10 + 4 \times 10) = 60$ (for remaing nodes)

$= 76$ B assuming int → 2

" → 4

$= 66$

Adjacency ~~matrix~~ matrix :

	a	b	c	d
a	0	1	1	0
b	1	0	1	1
c	1	1	0	1
d	0	1	1	0

memory = $(4 \times 4) \times 2$

$= 16 \times 2$

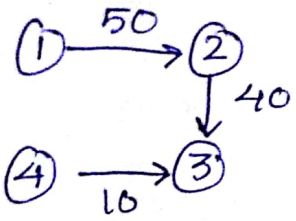
$= 32$ bytes.

When graph is dense : AM (is efficient)

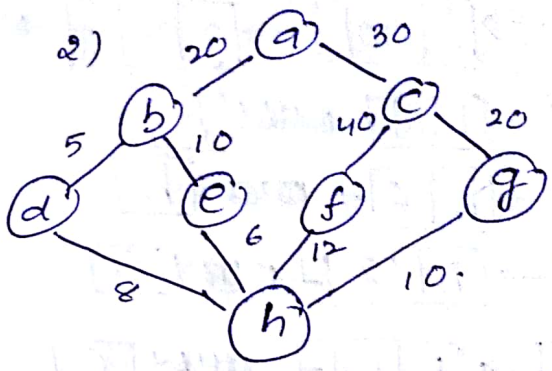
when a grap is sparse: ALL (is efficie

Represent the following graph in computer memory using adjacency linked list and adjacency matrix, and find the efficient data structure.

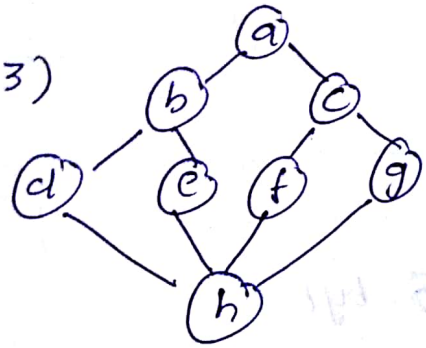
1)



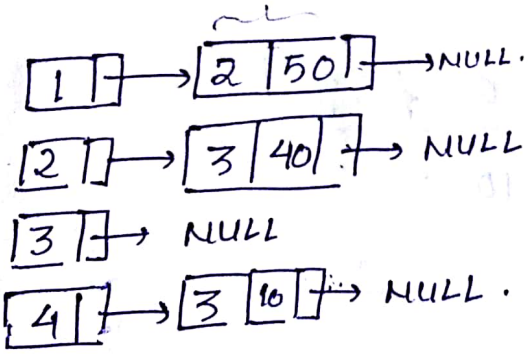
2)



3)



1)



$2 \times 4 + 4 \times 3 + 2 \times 3$

$= 8 + 12 + 6$

$= 26$

$16 + 12 + 6$

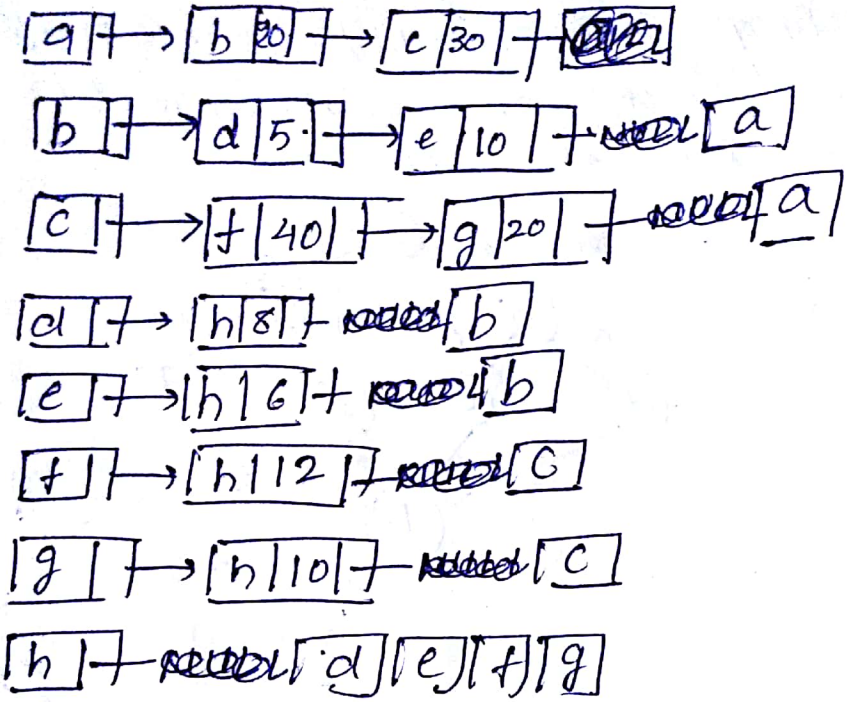
$= 28 + 6$

$= 34$

	1	2	3	4
1	∞	50	0	0
2	0	∞	40	0
3	0	0	∞	0
4	0	0	10	∞

$(4 \times 4) \times 2$

$= 32$ bytes



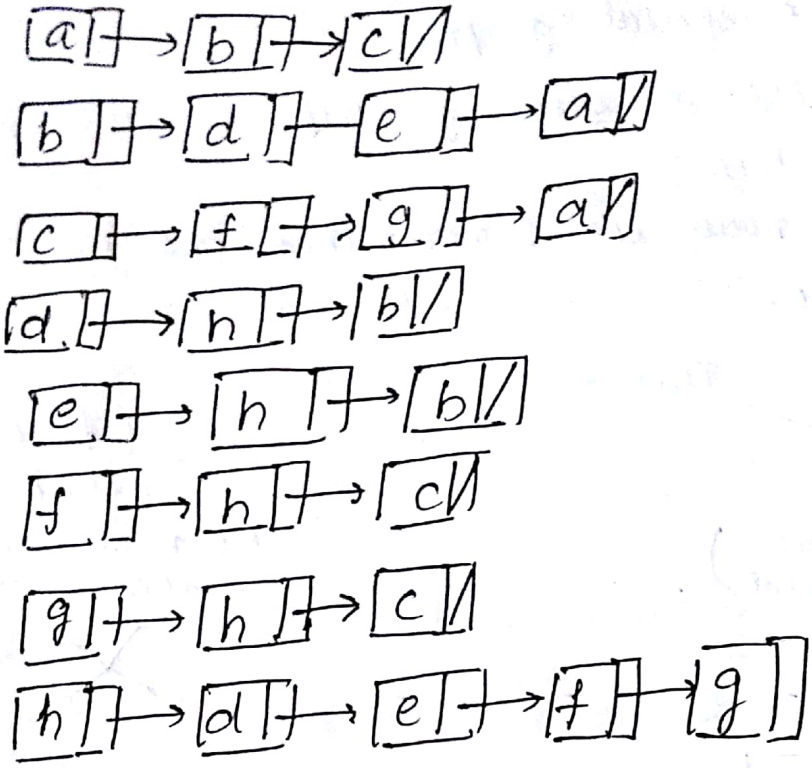
$4 \times 8 + 6 \times 20 = 32 + 120 = 152$ bytes

	a	b	c	d	e	f	g	h
a	0	20	30	-	-	-	-	-
b	20	-	-	5	10	-	-	-
c	30							
d								
e								
f								
g								
h.								

(9×64) bytes

$= 128$ bytes.

(3)



$28 \times 4 = 112$ bytes.

a b c d e f g h

a

b

c

d

e

f

g

h.

$(8 \times 8) \times 2 = 64 \times 2$

$= 128$ bytes.

TREES : a special graph

- It is a directed a cyclic graph
- Properties of tree :
  1) No. of node is 1 more than no. of edges.
  2) acyclic.

Tree

Rooted
(If a node is specified as root then it is rooted).

Ex.  ← Root

1

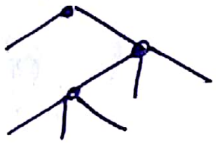
(Binary)
{Root/node have parent 0, 1, or 2 sub trees}.

{if node doesn't have any child then it is leaf node}

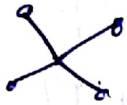
root → no parent
Intermediate → both parent & child

Ex. 

(Non-Binary)
{more than 2 sub tree to parent}



(free)
(UnRooted)
(No node specified as rooted

Ex. 

Binary Tree

Left skewed
→ all the node are left to its parent
Ex 

Right skewed
→ all nodes are right to its parent
Ex 

Strictly Binary
each parent has 0 or 2 sub tree
Ex. 

complete Binary tree
all levels are completely filled.

No of nodes in a level = 2^n
n → level.

Cont.

↓

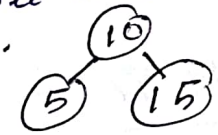
Almost complete Binary tree.
all levels are filled but last level is not completly filled & is filling from left to right

↓

Min Heap.
value of parent is < than all its sub tree value.

↓

Binary Search Tree.
if value of parent is >= left subtree and < its right sub tree then the tree is called BST.

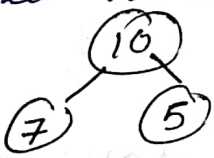


↓

AVL
In a binary tree if leaf node is at last level or last -1 level.

↓

Max Heap tree
value of parent is > is greater than all its sub tree value.



or in BST the balance factor is zero, 1, or -1.

balance factor = max height of left sub tree − max height of right sub tree.

↓

Balanced tree.
if balance factor of all nodes are zero.
Ex. complete binary tree.

↓

2-3 Tree / m-way Tree.
It is a balanced tree.
if 2 → data nodes then we have 3 subtrees.

~~Non binary~~

Spanning Tree: (can be binary or non binary).
It contains all the nodes of the original graph without having any cycle

No. of spanning tree that can be formed using n-nodes.

$= 2^n - n$.

i.e.

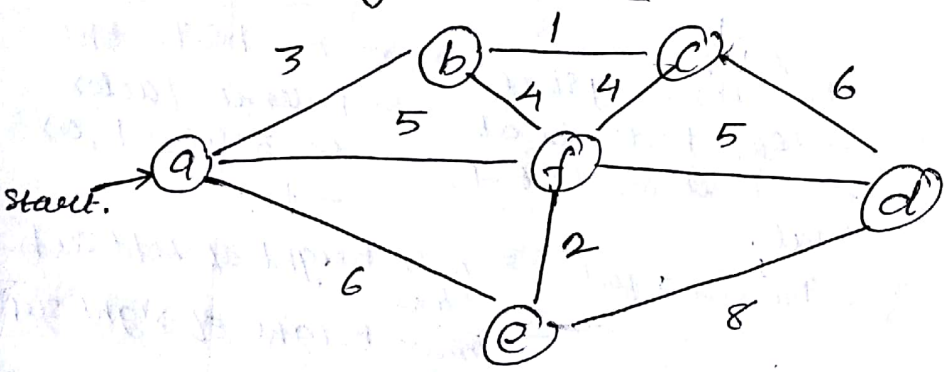
	binary No. of spanning tree.
$n = 3$	$2^3 - 3 = 5$
$n = 4$	$2^4 - 4 = 12$

Minimum cost Spanning Tree :

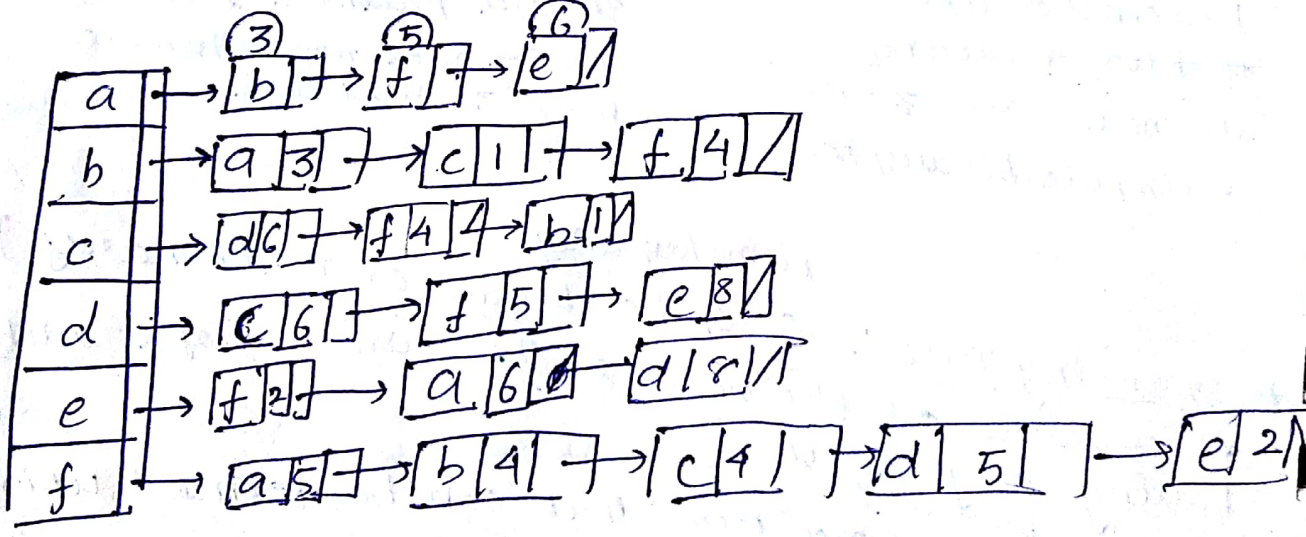
Spanning tree that can be constructed from n nodes = $2^n - n$ and the spanning tree having least cost (sum of weights to construct tree) is called minimum cost spanning tree :

Ex.

Prim's Algorithm :



Represent the given graph using linked and adjency matrix.



$= 6 \times 4 + 20 \times 6$

$= 24 + 120$

$= 144$ bytes

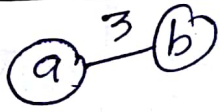
	a	b	c	d	e	f
a	0	3	∞	∞	6	5.
b	3	0	1	∞	~~4~~	4
c	∞	1	0	6	∞	4.
d	∞	∞	6	0	8	5
e	6	∞	∞	8	0	2
f	5	4	4	5	2	0

$(6 \times 6) \times 2 = 36 \times 2 = 72$ bytes.

Prim's Algorithm : (Greedy Designing Tree).

The Graph should be connected, weighted graph, with a starting nodes.

→ start → root node of spanning tree & initially all nodes is unvisited

Tree Vertices (visited nodes).	Remaining vertices (unvisited node)	Illustration (Spanning tree covi)
T(a, –)	b(a, 3) → (minimum cost).	
For visiting unvisited node b from a	c(–, –) → novisited node connected to c	
	d(–, –) – –do–	
	e(a, 6)	
	f(a, 5).	

We found minimum cost to reach every node via visited node.

Select the minimum of all from the column. visited via a.

Now, same process for (b, 3)

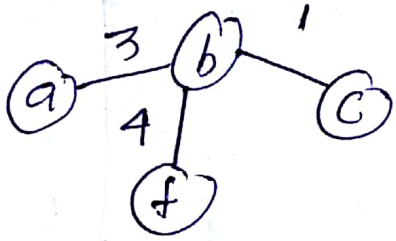
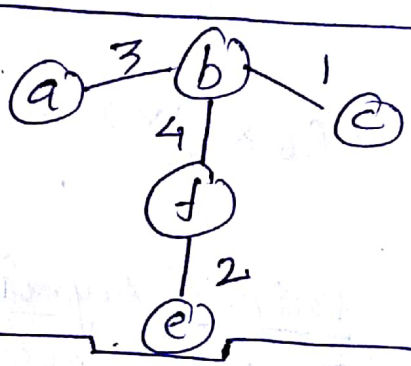
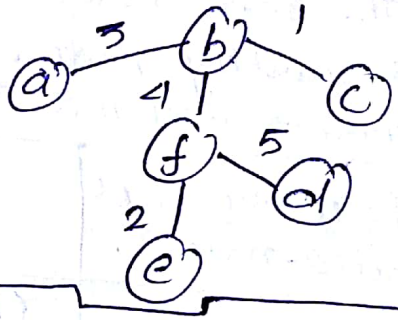
	a	b	c	d	e	f	
Visited	1	1	1	1	1	1	✓

T(a, –)

node ↑ means → have no parent.

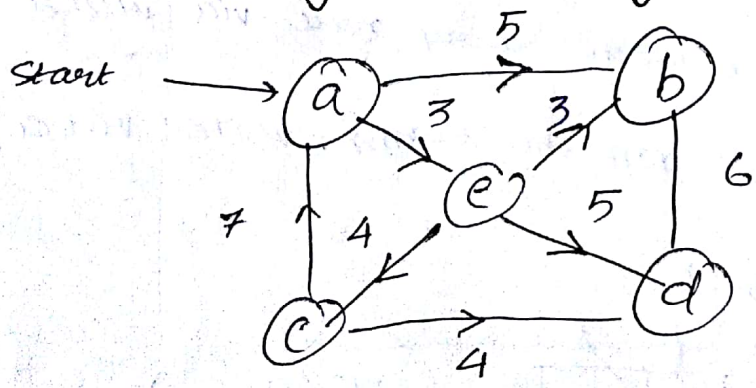
T(b,3)	c(b,1), d(–,–), e(a,6), f(b,4)	

To check the minimum cost from visited node a and b to all unvisited node.

T(c,1)	d(c,6), e(a,6), f(b,4)	
T(f,4)	d(f,5), e(f,2)	
T(e,2)	d(f,5)	

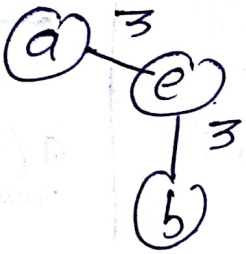
min.
Cost = 3+1+4+5+2 = 15

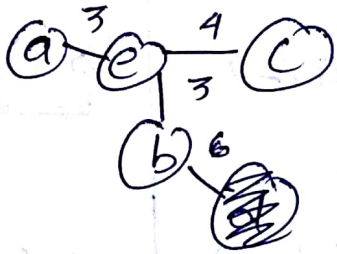
4. Construct minimum cost spanning tree for following graph using prims algorithm.

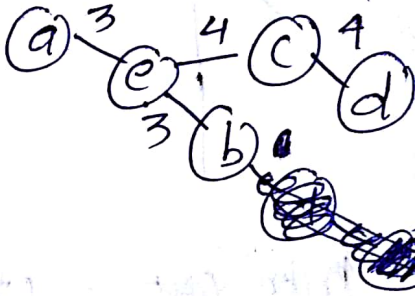


T(a, –)	b(a,5), d(–,–), c(––) e(a,3)
T(e, 3)	b(e,3), d(e,5), c(e,4) ~~c(e,4)~~
T(b, 3)	d(b,6), c(e,4)
T(c, 4)	d(c,4)

(a)

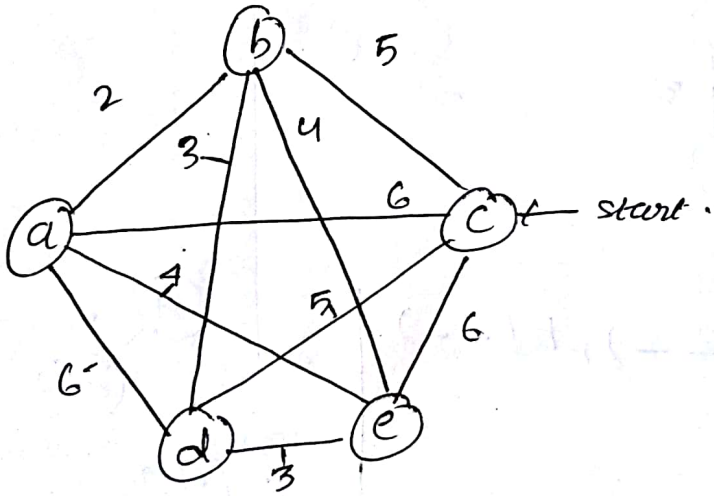




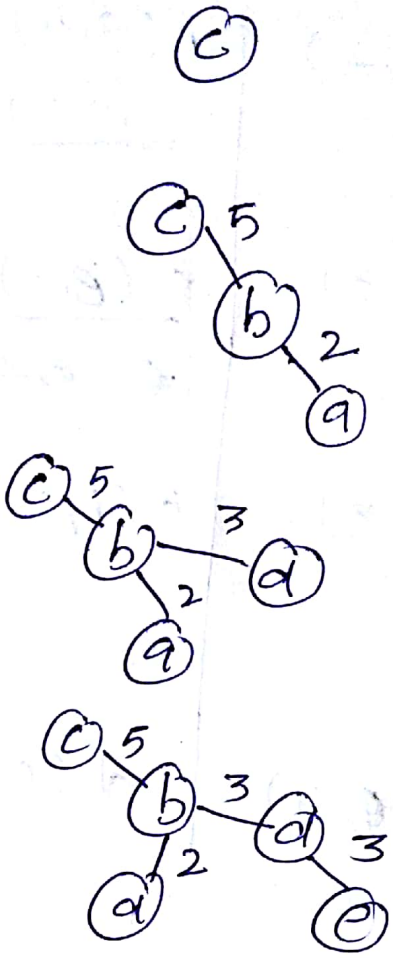


min cost = 14.

(2)

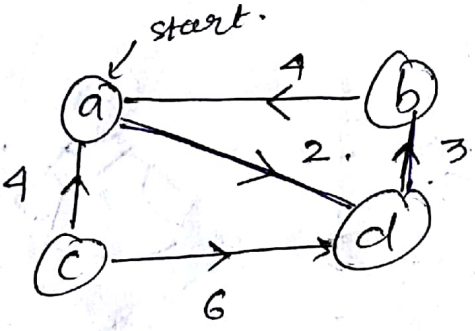


T(c,–)	a(c,6), b(c,5), d(c,5) e(c,6)
T(b,5)	a(b,2), d(b,3), e(b,4)
T(a,2)	d(b,3), e(b,4)
T(d,3)	e(d,3)

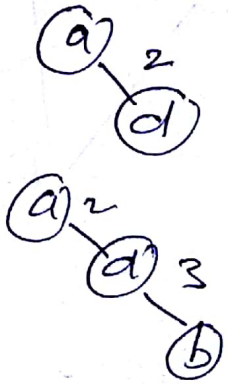


Min. cost = 13.

Ans.



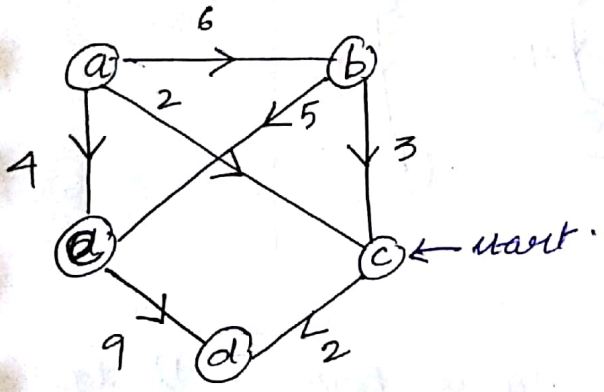
T(a,–)	d(a,2), c(––), b(––)
T(d,2)	b(d,3), c(––)
T(b,3)	c(–,–)



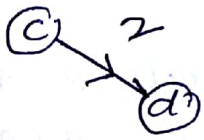
The graph is disconnected graph.
∴ we can't construct min cost spanning Tree.

To overcome this problem we have ~~Kruskal's~~ kruskal's algo.

Construct a minimum cost spanning tree for the following graph using prim's algorithm and kruskal's algorithm.



1) using prim's algorithm

T(c,-)	d(c,2)	
T(d,2)		

⇒ If there is no path from starting node to every other node (disconnected graph) then we can't construct minimum cost spanning using prim's algorithm hence we use kruskal's algorithm.

Kruskal's Algorithm.

1. Arrange all the edges in ascending order of their weights.
2. select the minimum cost edge from the list of edges.
3. select a minimum cost edge from a visited node in such a way that it doesn't make any cycle.

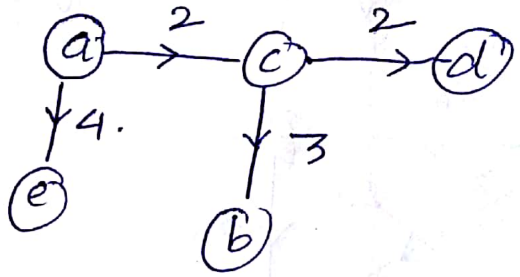
edges and cost.

(a,c) = 2 ; (c,d) = 2

(b,c) = 3 ; ~~(a,b) (b,d)~~

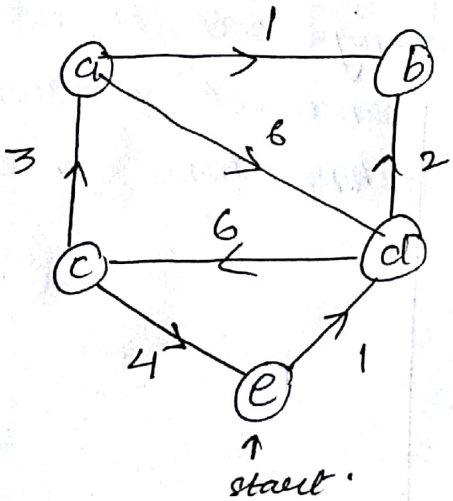
(a,e) = 4 ; (b,e) = 5

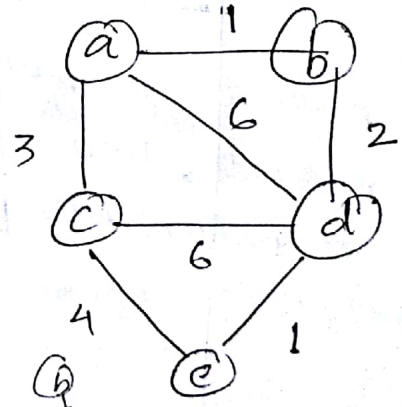
(a,b) = 6 ; (e,d) = 9

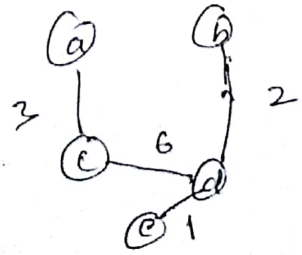


* We will repeat the following step till we have (n-1) edges for the given n-node

Ex construct a minimum cost spanning tree using Prim's and Kruskal's algorithm.

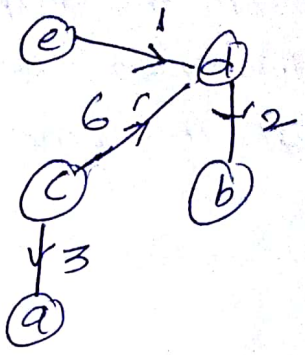






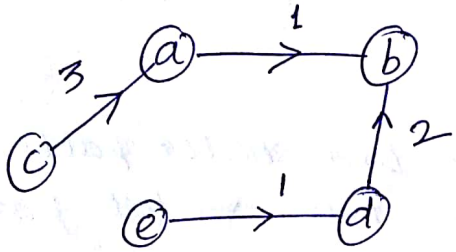
Tree nodes	Remaining nodes
T(e,–)	d(e,1)
T(d,1)	b(d,2) c(d,6)
T(b,2)	c(d,6)
~~T(c,6)~~	a(c,3)
T(c,6)	

mcst



MCST = 12

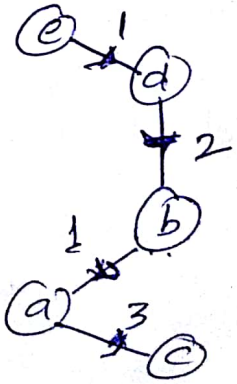
Kruskals



mcost = 7.

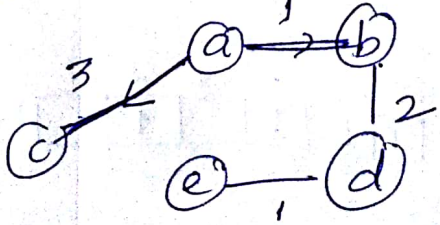
Tree	Remaining Nodes
T(e,–)	d(e,1) c(e,4)
T(d,1)	b(d,2) c(e,4) a(d,6)
T(b,2)	a(b,1) c(e,4)
T(a,1)	c(a,3)

illustration



minimum cost = 7.

Kruskals



(7)

When the graph is undirected and connected then minimum cost spanning tree is same for both. prim's and kruskal's algorithm.

Dijkstra's algorithm: (Greedy designing tech).

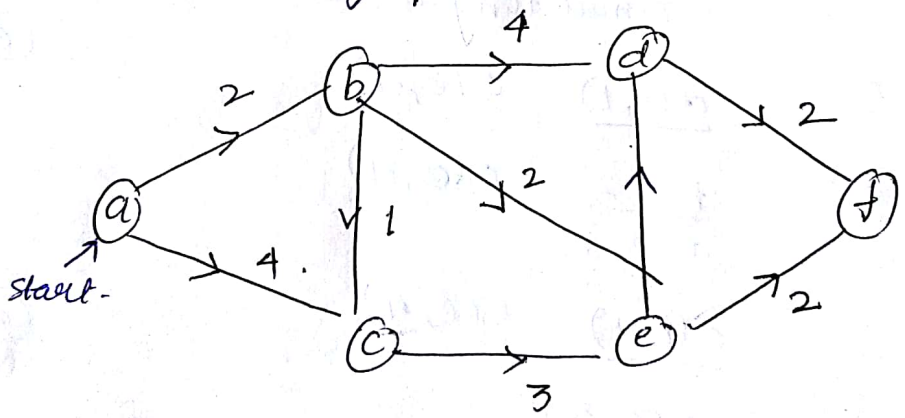
(spanning tree construction and shortest path ~~to~~ ~~to~~ ~~edges~~. to all the vertices starting from a given node.)

- To find smallest path from starting node to other remaining path.
- single source shortest path

Floyd's algo: (It is used to find shortest path blw all the nodes. / All pair shortest path.)
→ (dynamic programming.)

(1) we need a connected graph.

4s.



a	b	c	d	e	f
1	1	1		1	

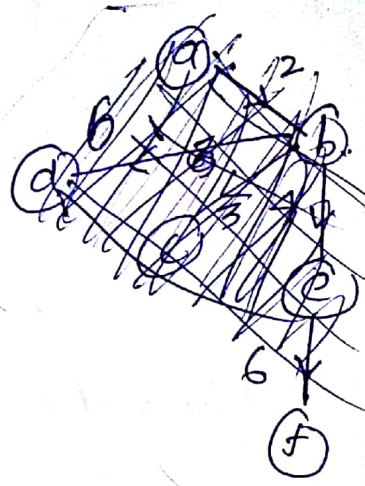
nodes,

a(–,0)

Remaining nodes

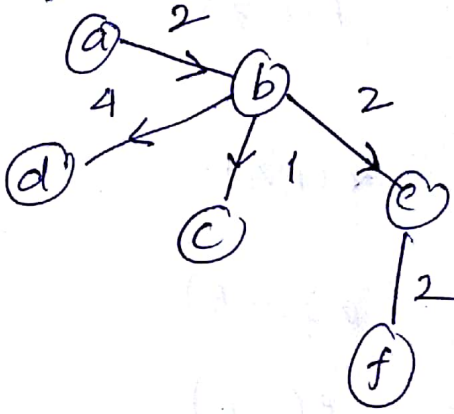
b(a,2), c(a,4), d(–,∞)
e(–,∞), f(–,∞).

illustration.



b(a,2)	c(b,3) d(b,6) e(b,4) f(–∞)
c(b,3)	d(b,6) e(b,4) f(–,∞)
e(b,4)	d(b,6) f(e,6)
d(b,6)	f(e,6)

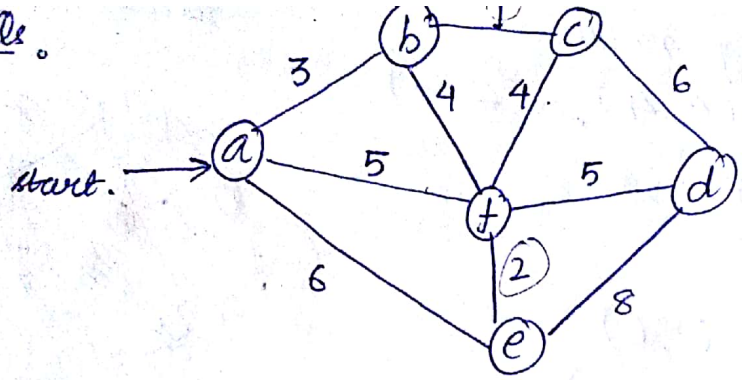
Ans



cost = 11

a → b = 2
a → c (a → b) = 3
a → d (a → b → d) = 6
a → e (a → b → e) = 4
a → f (a → b → e → f) = 6.

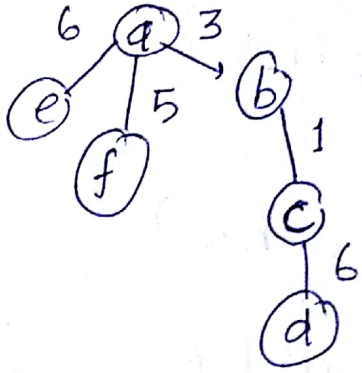
Qs.



Find

$a \to e$

$a \to d$.

Tree nodes	Remaining Node	Illustration
$a(-,0)$	$b(a,3)$, $f(a,5)$, $e(a,6)$ $c(-,\infty)$, $d(-,\infty)$,	
$b(a,3)$	$c(b,4)$, $f(a,5)$, $d(-,\infty)$ $e(a,6)$	
$c(b,4)$	$d(c,10)$, $e(a,6)$, $f(a,5)$	
$f(a,5)$	$d(c,10)$, $e(a,6)$.	
$e(a,6)$	$d(c,10)$	

$a \to e$ $(a \to e) = 6$.

$a \to d)$ $(a \to b \to c - d) = 10$.

$a \to f = 5$

$a \to c = 4$

$a \to b = 3$

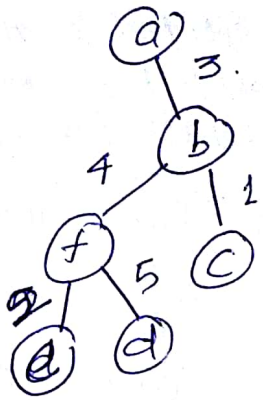
Minimum cost = 21.

✱ For minimum cost spanning tree Prim's algorithm is better than Dijakstra!

Construct spanning tree (minimum cost) using prims algorithm. Shortest using Dijkstra's algorithm.

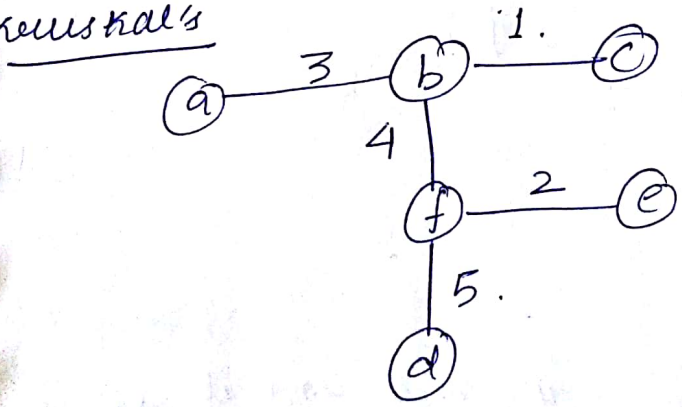
Tree node.

T (a, –)	b (a, 3), f (a, 5), e (a, 6)
T (b, 3)	c (b, 1), ~~d~~ e (a, 6), f (b, ~~4~~).
T (c, 1)	d (c, 6) e (a, 6) f (b, ~~4~~)
T (f, 4)	d (f, 5), e (f, 2)
T (e, 2)	d f (d, 5)



minimum cost = 15.

Kruskal's



minimum cost using kruskal's algo = 15.

ALGORITHM

```
ALGORITHM Prims (V, G[][], selected[])
// V is the number of nodes, G is the cost matrix and
// selected is the visited array.
// G[][] is cost adjacency matrix.
while (no_edge < V-1)
{ int min = INF; // INF is some max value.
  x = 0;
  y = 0;
  for(int i=0; i<V; i++)
  { if (selected[i])
    { for (int j=0; j<V; j++)
      { if (!selected[j] && G[i][j])
        // if not selected and ∃ an edge.
        { if (min > G[i][j])
          { min = G[i][j];
            x = i;
            y = j;
          }
        }
      }
    }
  }
}
```

we are considering a[0] node as start node. and it is already visited i,e a[0] = 1.

selected[0] = 1;

ALGORITHM Kruskal(G).

// Kruskal's algorithm for constructing a minimum
// cost spanning tree
// Input: A weighted connected graph $G = (V, E)$
// Output: E_T, the set of edge composing a minimum spanning tree of G sort E in nondecreasing order of the edges weights $w(e_{i_1}) \leq \cdots \leq w(e_{i_{|E|}})$.

$E_T \leftarrow \phi$; ecounte $\leftarrow 0$ // initialize the set of tree edges & its size.
$k \leftarrow 0$; // initialize no. of processed edges.

while ecounte $< |V| - 1$ do
  $k \leftarrow k + 1$
  if $E_T \cup \{e_{i_k}\}$ is acyclic
    $E_T \leftarrow E_T \cup \{e_{i_k}\}$; ecounte $\leftarrow$ ecounte
return E_T.

Kruskal:

$O(e \log e)$

where e → no. of edges.

Prim's

$O(e \log V)$

where e → no. of edges
V → no. of vertices.

26/10/18

Graph Traversal Methods :

Q. Construct a spanning Tree using DFS (Depth first search) and BFS (Breath first search) and also find. DFS traversing order, DFS order, BFS traversing Order, BFS order.

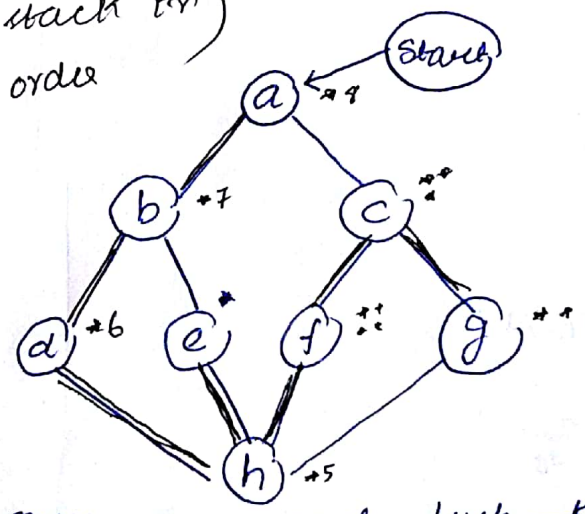
* BFS and DFS order can be used to construct spanning tree. For that we need the unweighted connected graph.
* If graph is disconnect we can't construct spanning tree using DFS or BFS technique but we can create no. of trees. i.e. Forest.

→ If nodes are accessed from bottom level to up level then (DFS order). (bottom up approch)

→ If nodes are accessed from up level to down level then (BFS order) (Top down approch).

- For DFS implementation → stack is efficent data structure
- For BFS implementation → Queue is efficent data structure

Ex. 1) DFS (insert into stack for descending order)



unweighted. connected cyclic undirected dense graph.

(i) Consider a stack of some max size ~~and push the starting node into the stack~~.

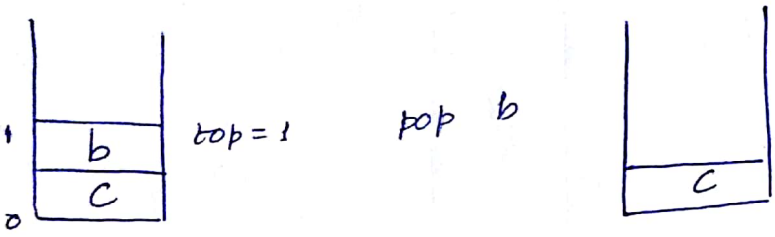
Initialize stack

Initialize the visited array with 0.

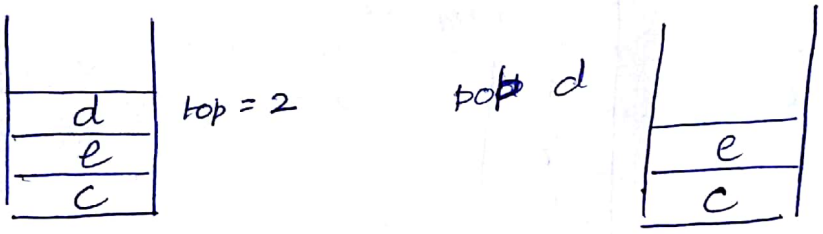
0	a	b	c	d	e	f	g	h
	∅	∅	∅	∅	∅	∅	∅	∅
	1	1						

a b d f

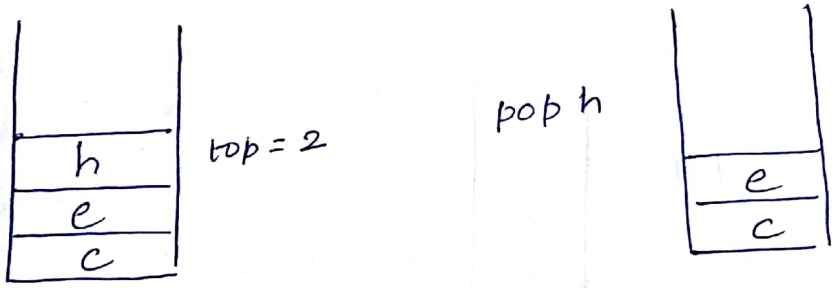
1) start, a is visited; now push all connected unvisited node into the stack.
push (b and c)



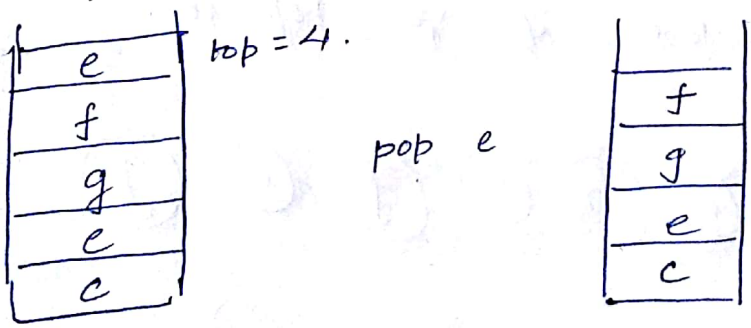
2) Find all the unvisited connected nodes to b and push it into the stack.



3) From d find all the unvisited connected nodes to d and push it into the stack.

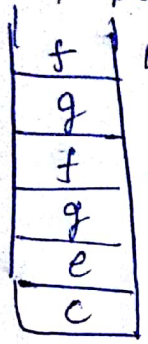


4) From h find all the unvisited connected nodes and push it into stack

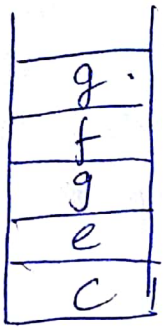


∴ there is no node that is directly connected to (e) and is unvisited node hence back track.
We are at (h).

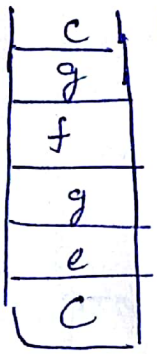
push all unvisited connected node from h into stack



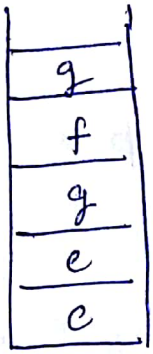
pop f



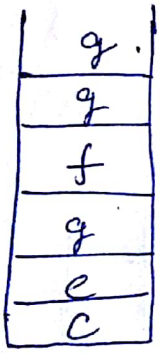
now, from (f) ~~deed~~ push all directly connected unvisited elements from (f). a



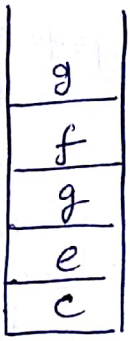
pop c



push all unvisited connected node from (c)



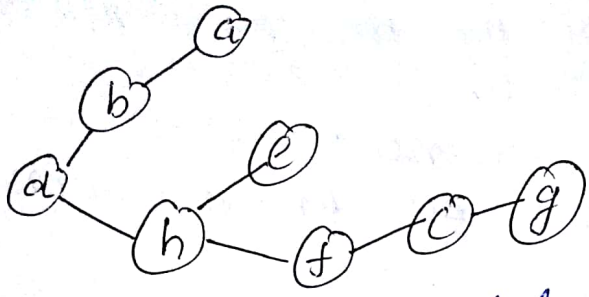
pop g



check if top is visited, if yes pop it.

V.O : (a) (b) (d) (h) (e) (f) (c) (g)

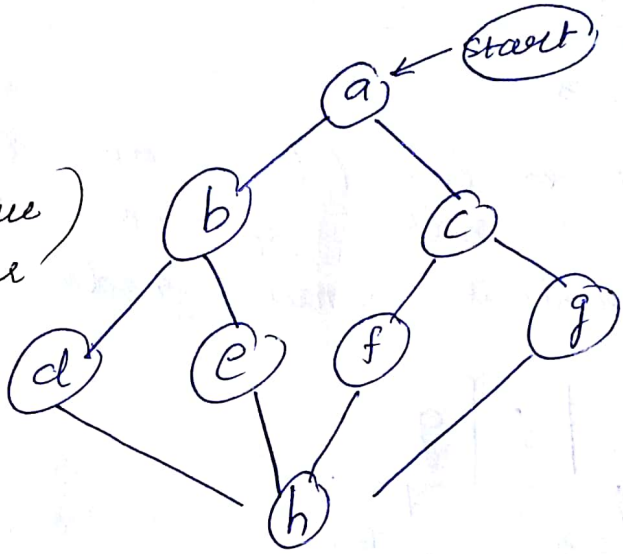
DFS : (e) (g) (c) (f) (h) (d) (b) (a)



Spanning tree is constructed when we only include the backtracked edges.

2) BFS

(Insert into the queue in ascending order)



Initialize Queue as empty.

i,e front = -0 ; rear = -1

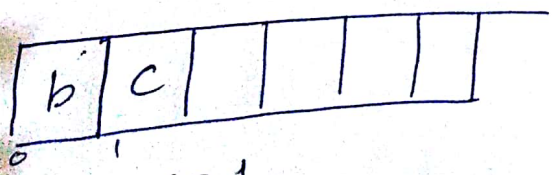
Initialize visited array to zero.

a	b	c	d	e	f	g	h	
ϕ_1	ϕ_1	ϕ_1	ϕ_1	ϕ_1	ϕ_1	ϕ_1	ϕ_1	

VO: a b c d e f g h

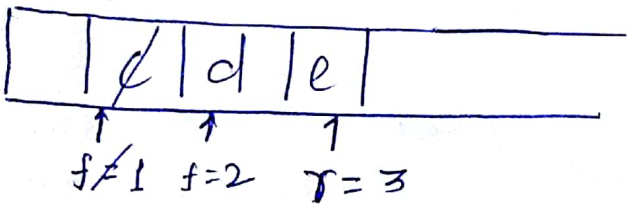
BFS: a b c d e f g h.

Start from @, find all the unvisited connected node to @ and insert it into the queue in alphabetical order

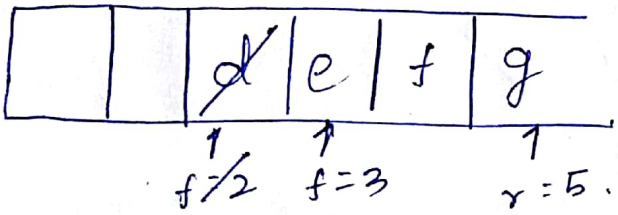


f=0 r=1

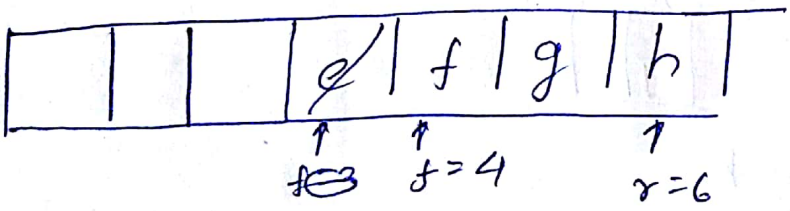
Now, delete the front end of the tree and update the front pointer. (f = 1, r = 1)
check wether (b) is visited or not.
if not visit (b) and find all unvisited connected node into the queue.



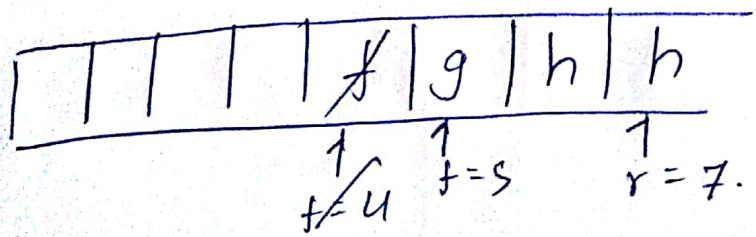
delete from front end (c) and check wether (c) is visited or not if not visit it & insert all unvisited connected node into queue



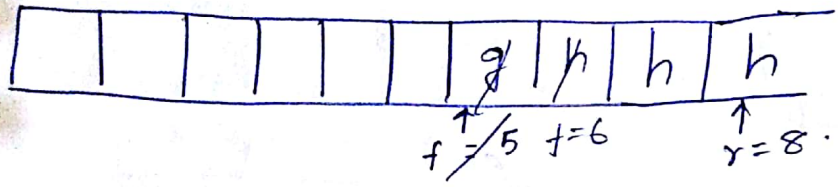
delete front end i,e (d) and check wether (d) is visited or not, if not visit it and insert all unvisited connected node from (d)



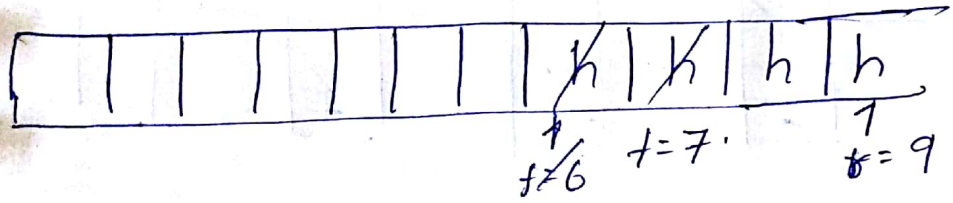
delet the front end i,e (e) and check wether (e) is visited or not, if not visit it and insert all unvisited connected node.
(h) is unvisited connected



delete front end of the queue i.e (f) check wether it is visited or not, if not insert all unvisited connected node.

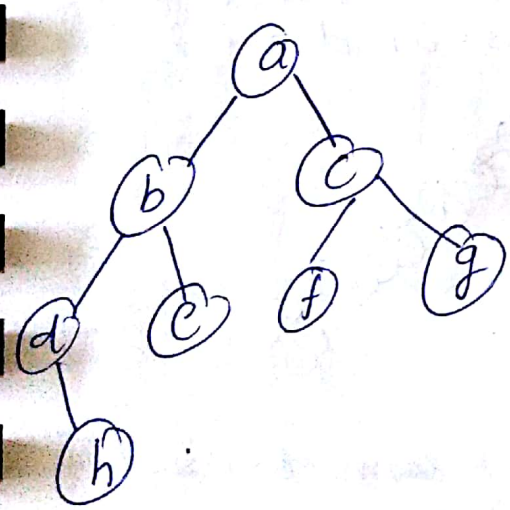


delete the front end of the queue i,e (g) check wether it is visited or not, if not insert all unvisited connected nodes

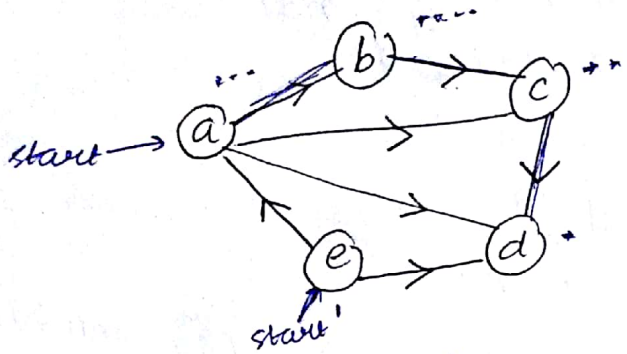


delet the front end of the queue i,e (h) check wether it is visited or not if not insert all univ connected node.

delete all the elements till queue is empty



Construct DFS and BFS tree for the following graph



Initialization

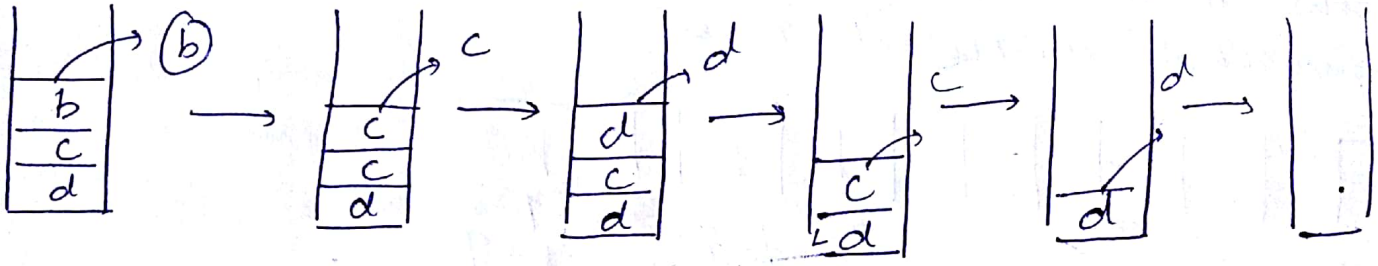
top = -1

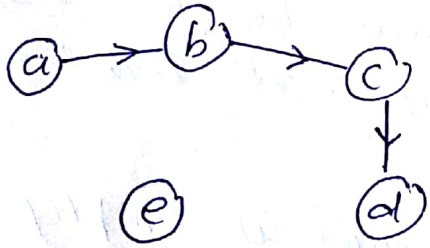
a	b	c	d	e
1	1	1	1	0

vo : a b c d e

DFS : d c b a

1) DFS.

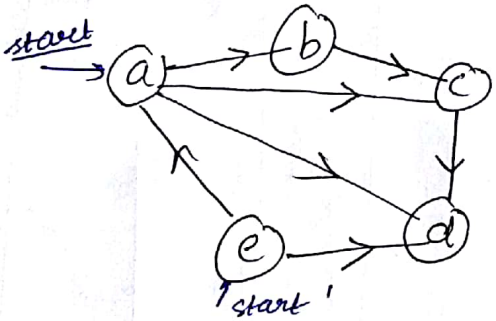




Initialization :

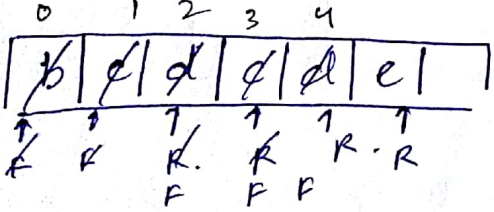
F = 0 ; R = -1.

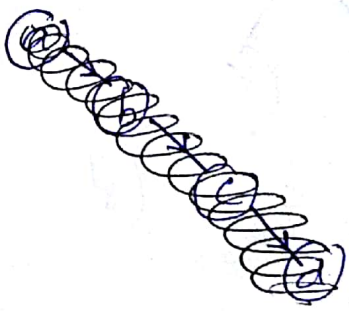
ii) BFS.



vo : a b c d, e

BFS : a b c d, e .

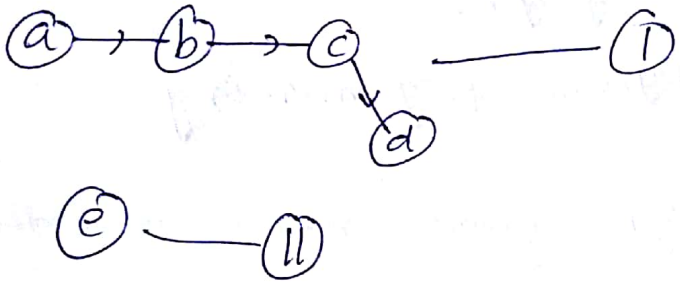




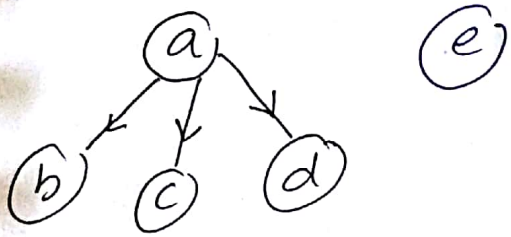
If the graph is disconnected then we can't ~~connect~~ have a spanning tree ; we have multiple tree (forest) as a solution.

* Here, (e) is a unvisited node, start from least unvisited node so, we will visit (e) and check wether there is any other unvisited connected nodes.

We will have multiple tree as a solution.



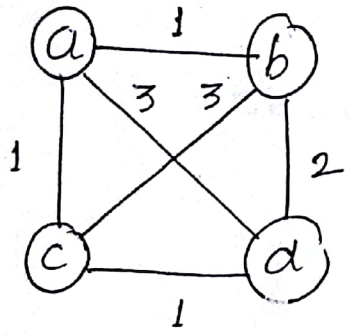
BFS tree :



∵ (b) was inserted when (a) was visited (a→b)
(c) " " " (a) was visited (a→c)
(d) " " " (a) was visited (a→d)

(e) is disconnected so we insert it into the queue in last.

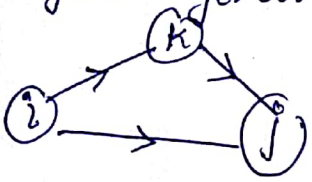
Q. Find shortest path for the following graph using floyd's algorithm. (All pair shortest path algo.)



- It is used to find the minimum cost b/w any pair of nodes from the graph.
- Dynamic programming

Step1 : create a matrix of $n \times n$ where $n \rightarrow$ no. of nodes.

Floyd's algorithm :



$$P(i,j) = \min \{ P(i,j) , P(i,k) + P(k,j) \}$$

where, $k \rightarrow$ various intermediate node.

step 2 : make cost adjency matrix.

$P(0) =$

	a	b	c	d
a	0	1	1	3
b	1	0	3	2
c	1	3	0	1
d	3	2	1	0

Path matrix without using any intermediated node.

i,e.

taking (a) as intermediate node

$P(1) =$

	a	b	c	d
a	0	1	1	3
b	1	0	2	2
c	1	2	0	1
d	3	2	1	0

$P(2) =$

	a	b	c	d
a	0	1	1	3
b	1	0	2	2
c	1	2	0	1
d	3	2	1	0

taking ⓐ & ⓑ as intermediate

$P(3) =$

	a	b	c	d
a	0	1	1	2
b	1	0	2	2
c	1	2	0	1
d	2	2	1	0

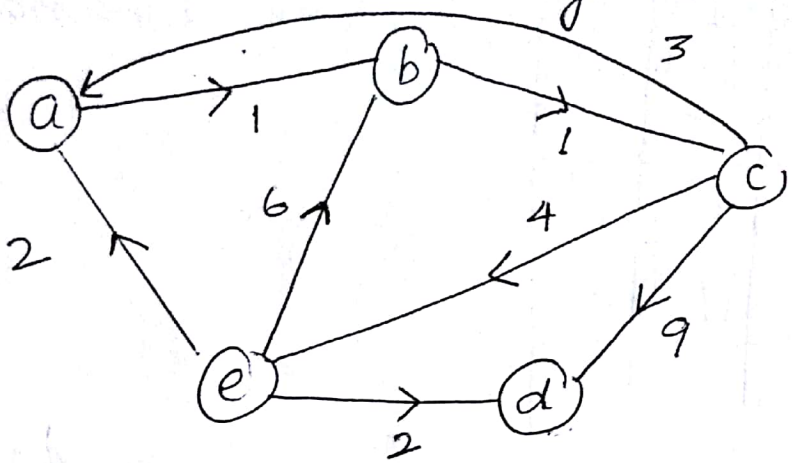
ⓐ, ⓑ & ⓒ taking as intermediate.

$P(4) =$

	a	b	c	d
a	0	1	1	2
b	1	0	2	2
c	1	2	0	1
d	2	2	1	0

ⓐ, ⓑ ⓒ and taking ⓓ as intermediate.

vertex. Shortest path using (b) and (c) intermediate



P(0) =

	a	b	c	d	e
a	0	1	∞	∞	∞
b	∞	0	1	∞	∞
c	3	∞	0	9	4
d	∞	∞	∞	0	∞
e	2	6	∞	2	0

P(1) =

	a	b	c	d	e
a	0	1	2	∞	∞
b	∞	0	1	∞	∞
c	3	∞	6	9	4
d	∞	∞	∞	0	∞
e	2	6	7	2	0

b as intermediate

P(2) =

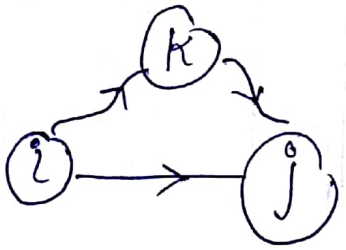
	a	b	c	d	e
a	0	1	2	11	6
b	4	0	1	10	5
c	3	∞	0	9	4
d	∞	∞	∞	0	∞
e	2	6	7	2	0

(b) and (c) as intermediates

WARSHALL'S ALGORITHM

Used to find wether ∃ a path b/w pair of vertices.

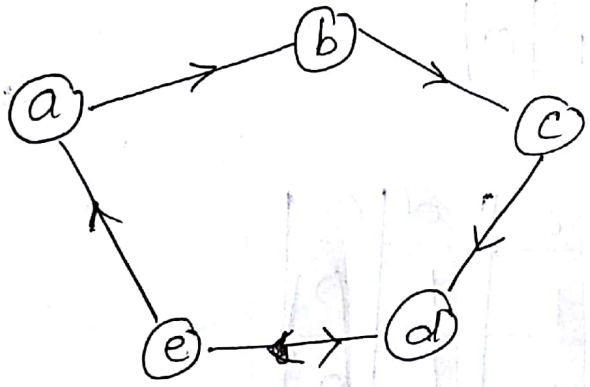
$$P(i,j) = \left(P(i,j) \;||\; P(i,k) \;\&\&\; P(k,j) \right)$$



where, k is various intermediate nodes.

- Here, we use logical expression to find a path.
- In warshall's algo we use unweighted graph.
- Time complexity $T(n) = O(n^3)$.
  (for both Warshall's and Floyd's algorithm)

Qs. Find a path matrix for the following graph using warshall's algorithm.



$P(0)$ =

	a	b	c	d	e
a	0	1	0	0	0
b	0	0	1	0	0
c	0	0	0	1	0
d	0	0	0	0	0
e	1	0	0	1	0

$P(1)$ =

	a	b	c	d	e
a	0	1	0	0	0
b	0	0	1	0	0
c	0	0	0	1	0
d	0	0	0	0	0
e	1	1	0	1	0

P(2) =

	a	b	c	d	e
a	0	1	0	0	0
b	0	0	1	0	0
c	0	0	0	1	0
d	0	0	0	0	0
e	1	1	1	1	0

P(3) =

	a	b	c	d	e
a	0	1	1	1	0
b	0	0	1	1	0
c	0	0	0	1	0
d	0	0	0	0	0
e	1	1	1	1	0

P(4) =

	a	b	c	d	e
a	0	1	1	1	0
b	0	0	1	1	0
c	0	0	0	1	0
d	0	0	0	0	0
e	1	1	1	1	0

$P(5) =$

	a	b	c	d	e
a	0	1	1	1	0
b	0	0	1	1	0
c	0	0	0	1	0
d	0	0	0	0	0
e	1	1	1	1	0

Bellman Ford Algorithm :

* single source shortest path algorithm (similar to Dijikstra)
* Whenever the weights are -ve. We can't find the shortest path using dijkstra's algorithm. So, we have Bellman ford algo.

Home work (Find the algorithms ?).

P(5) =

	a	b	c	d	e
a	0	1	1	1	0
b	0	0	1	1	0
c	0	0	0	1	0
d	0	0	0	0	0
e	1	1	1	1	0

Bellman Ford Algorithm :

- single source shortest path algorithm (similar to Dijkstra)
- whenever the weights are -ve. we can't find the shortest path using dijkstra's algorithm. so, we have Bellman ford algo.

Home work (Find the algorithms ?).

14/11/2018

Optimal Binary Search Tree : (Dynamic Programming)

Element	0	1	2	3
key data	10	12	16	21
Freq. of search	4	2	6	3

Ex: construct optimal binary search tree of the following data using Dynamic Programming designing tech.

Step 1 : create a matrix of $n \times n$ in which $n \rightarrow$ no. of element.

m

	0	1	2	3
0	4			
1		2		
2			6	
3				3

for $l = 1$ i.e assuming that their is single node / data. i.e 0, 1, 2, 3 individually.

$l_a = 2$ (m_{01}, m_{12}, m_{23})

$l = 1$ $(m_{00}, m_{11}, m_{22}, m_{33})$.

⇐ cost for the binary tree is

	0	1	2	3
0	4	$8^{(0)}$	$20^{(1)}$	$26^{(2)}$
1		2	$10^{(2)}$	$16^{(2)}$
2			6	$12^{(2)}$
3				3

l_3, l_2, l_1

0	1	2	3
10	12	16	21
4	2	6	3

$l = 1$

$l_2 = 2$ $\quad m_{01} = \{$ sum of cost of $m_{00} + m_{10} + \min\{ m_{00}^{(1)}, m_{11}^{(0)} \}$

$= 4+2 + \min(4, 2)$

$= 6+2 = 8.$

$m_{12} = m_{11} + m_{22} + \min\{ m_{11}^{(2)} = 6, m_{22} = 2 \}$ (1)

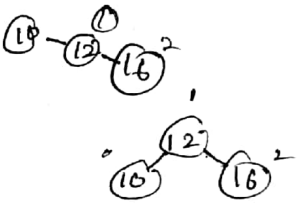
$= 2+6+2$

$= 10$

$m_{23} = m_2 + m_3 + \min\{ m_{33}^{(2)} = 3, m_{22} = 6 \}$ l_3

$= 6+3+3$

$= 12$

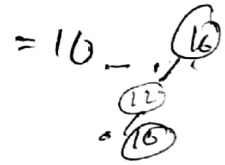


$l = 3$ $\quad m_{02} = m_0 + m_2 + m_6 + \min\{$

(0) is root
$m_{12} = 10$

(1) is root
cost $= m_0 + m_2 = 4+6 = 10$

(2) is root
cost $= m_{01} = 8$



$m_{02} = 4+2+6+8$

$= 12+8$

$= 20$

$m_{13} = m_1 + m_2 + m_3 + \min\{$

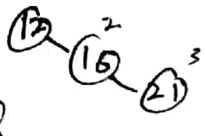
(1) is root
$m_{23} = 12$

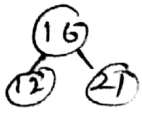
(2) is root
cost $= m_{11} + m_{33}$
$= 2+3$
$= 5$

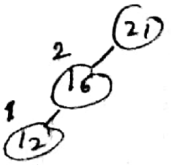
(3) is root
$m_{12} = 10$

$= 2+6+3+5$

$= 16.$







$l = 4$ $\quad m_{03} = m_0 + m_1 + m_2 + m_3 + \min\{$

(0) is root.
$m_{13} = 16$

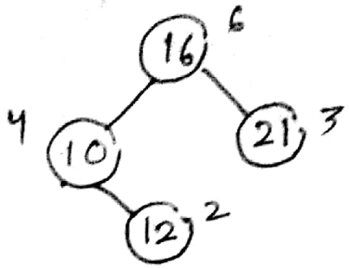
(1) is root.
cost $= m_{00} + m_{23} = 4 + 12 = 16$

(2) is root
cost $= m_{01} + m_{33} = 8 + 3 = 11$

(3) is root

$= 15+11$

$= 26.$



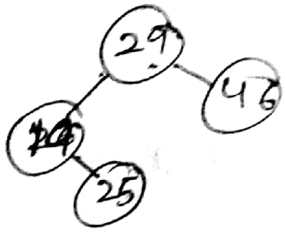
$6 = 1 + 4 + 2 + 3 = 2 + 12 = 2$

$= 6 + 8 + 6 + 12$

$= 26$

4. construct optimal binary tree of the following data

Element	0	1	2	3
key data	19	25	29	46
Freq.	16	5	9	12



	0	1	2	3
0	16	$26^{(1)}$	$49^{(1)}$	$80^{(2)}$
1		5	$19^{(2)}$	$43^{(2)}$
2			9	$30^{(3)}$
3				12

for $l = 1$ $m_{00} = 16$, $m_{11} = 5$, $m_{22} = 9$, $m_{33} = 12$

for $l = 2$ $m_{01} = m_0 + m_1 + \min \begin{cases} m_{00}^{(1)} = 16 \\ m_{11}^{(0)} = 5 \checkmark \end{cases}$

$= 16 + 5 + 5$

$= 26.$

$m_{12} = m_1 + m_2 + \min \begin{cases} m_{11}^{(2)} = 5 \checkmark \\ m_{22}^{(1)} = 9 \end{cases}$

$= 5 + 9 + 5$

$= 19.$

$m_{23} = m_2 + m_3 + \min \begin{cases} m_{22}^{(3)} = 9 \checkmark \\ m_{33}^{(2)} = 12 \end{cases}$

$= 9 + 12 + 9$

$= 30.$

for $l = 3$ $m_{02} = m_0 + m_1 + m_2 + \min \begin{cases} (0)\ 19 \checkmark \\ (1)\ 16 + 9 = 25 \\ (2)\ 26 \end{cases}$

$= 16 + 5 + 9 + 19$

$= 49.$

$m_{13} = m_1 + m_2 + m_3 + \min \begin{cases} (1)\ (30) \\ (2)\ 5 + 12 = 17 \checkmark \\ 3\ 19 \end{cases}$

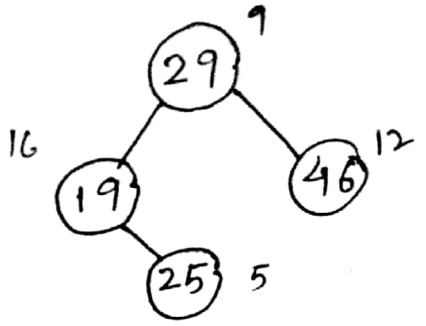
$= 5 + 9 + 12 + 17.$

$= 43.$

for $l = 4$ $\quad m_{03} = m_0 + m_1 + m_2 + m_3 + \min$ (0) 43, (1) 16+30 = 46, (2) 26+12 = 38 ✓, (3) (49)

$= 16 + 5 + 9 + 12 + 38$

$= 80.$



$9*1 + 16*2 + 12*2 + 5*3$

$= 9 + 32 + 24 + 15$

$= 24 + 32 + 24$

$= 80.$

15/011/2018.

Chained Matrix Multiplication :-

- (comes under dynamic programing)
- optimization technique.
- To get minimum no. of scalar multiplication operation.

$A = 13 \times 5$

$B = 5 \times 89$

$C = 89 \times 3$

$D = 3 \times 34$

① using Brute force Techniques :-

i) $((A.B)C)D$

$= (T_1.C)D$

$= T_2.D$

$= Ans$

$T_1 = (A.B)_{13\times89} = 13 \times 5 \times 89$

$= 5785$ scalar operation

$T_2 = (T_1.C)_{13\times3} = 3471$

$T_3 = (T_2.D)_{13\times34} = 1326.$

Total no. of scalar operation

$= 3471 + 5785 + 1326$

$= 10582$

② $(AB)(CD)$

$= T_1(C.D)$

$= T_1 T_2$

$= T_3$

$T_1 = (A.B)_{13\times89} = 13 \times 5 \times 89$

$= 5785$

$T_2 = (C.D)_{89\times34} = 89 \times 3 \times 34.$

$= 9078.$

$T_3 = T_1.T_2 = 13 \times 89 \times 34$

$= 39,338.$

$\Big\}$ 54201

(3) $(A(B.C))D$ =

$= (AT_1)D$

$= T_2 . D$

$= T_3$

Optimal

$T_1 = (B.C)_{5\times3} = 5\times89\times3$

$= 1335$

$T_2 = (A.T_1)_{13\times3} = 13\times5\times3$

$= 195$

$T_3 = (T_2.D)_{13\times34} = 13\times3\times34$

$= 1,326.$

$= 2856$

(4) $A((BC)D)$

$= A(T_2 . D)$

$= A . T_2$

$T_1 = (B.C)_{5\times3} = 5\times89\times3$

$= 1335$

$T_2 = (T_1 \times D)_{5\times34} = 5\times3\times34$

$= 510$

$T_3 = (A \times T_2)_{13\times34} = 5\times34\times13$

$= 2210.$

$15130.$

(5) $A(B(CD))$

$= A(B.T_1)$

$= A . T_2$

$= T_2$

$T_1 = (C.D)_{89\times34} = 89\times3\times34$

$= 9,078.$

$T_2 = (B.T_1)_{5\times34} = 5\times89\times34$

$= 15,130.$

$T_3 = (A.T_2)_{13\times34} = 13\times5\times34$

$= 2210.$

$26,418.$

When we use brute force technique we have to perform a large no. of operations (simple operation) many time.

n	2	3	4	5
T(n)	1	2	5	

$$T(n): \sum_{i=1}^{n} T(i) \cdot T(n-i)$$

i → is position of parantheses in the expression.

Dynamic Programming Method:

step 1: create a matrix of n×n when n → no. of matrixes given.

M	(1) A	(2) B	(3) C	(4) D
A (1)	0	5785	1530	2856.
B (2)		0	1335	1845
C (3)			0	9078.
D (4)				0

S=1

S=0

$$M_{ij} = \begin{cases} 0 & \text{if } i=j \mid s=0. \\ \text{how } s=0, \text{means no. of parentheses is zero or we only have one matrix.} & \\ d_{i-1} * d_i * d_{i+1} & \text{if } j=i+1 \mid s=1 \\ s=1; \text{ one parenthesis, two matrix} & \\ \min_{i \le k < i+s} \{ M_{ik} + M_{k+1,i+s} + d_{i-1} * d_k * d_{i+s} \} & \text{if } 1<s<n \mid i-1=j \end{cases}$$

$d_0\ d_1\ d_2\ d_3\ d_4$

$d = \{13, 5, 89, 3, 34\}$

if $j = i+1 \mid s = 1$

$m_{12} = d_0 d_1 d_2 = 13 \times 5 \times 89 = 5785$

$m_{23} = d_1 d_2 d_3 = 5 \times 84 \times 3 = 1335$

$m_{34} = d_2 . d_3 . d_4 = 89 \times 3 \times 34 = 9078.$

if $1 < s < n \mid i! = j$

$$\min \left\{ m_{ik} + m_{i+1,j+s} + d_{i-1} \cdot d_k \cdot d_{i+s} \right\}.$$

$i \le k \le i+s$

$m_{11} + m_{23} + d_0 d_1 d_3 \mid m_{11} + m_{23} + d_0 d_1 d_3$

$m_{12} + m_{33} + d_0 d_2 d_3. \mid m_{12} + m_{33} + d_0 d_2 d_3.$

for $s = 2$

$$m_{13} = \min \begin{cases} m_{11} + m_{23} + d_0 \cdot d_1 \cdot d_3 \\ m_{12} + m_{33} + d_0 d_2 d_3 \end{cases}$$

$i = 1 \quad s = 2$

$j = 3 \quad k = 12$

$$= \min \begin{cases} 0 + 1335 + 195 = 1530 \\ 5785 + 0 + 3471 = 9256 \end{cases}$$

$\boxed{m_{13} = 1530.}$

$$m_{24} = \min \begin{cases} m_{22} + m_{34} + d_1 d_2 d_4 \\ m_{23} + m_{44} + d_1 d_3 d_4 \end{cases}$$

$i = 2 \quad s = 2$

$j = 4 \quad k = 23$

$$\min \begin{cases} 0 + 9078 + 15130. = 24208. \\ 1335 + 0 + 510 = 1845 \end{cases}$$

$\boxed{m_{24} = 1845}$

$m_{ik} + m_{k+1,j+s} + d_{i-1} \cdot d_k \cdot d_{j+s}.$

$s = 3$

$$m_{14} = \min \begin{cases} m_{11} + m_{24} + d_0 * d_1 * d_4 = 4055 \\ m_{12} + m_{34} + d_0 d_2 d_4 = 54201 \\ m_{13} + m_{44} + d_0 d_3 d_4 = 2856. \end{cases}$$

$i = 1 \quad s = 3$

$j = 4 \quad k = 1 \text{ to } 3$

$$\boxed{m_{14} = 2856}$$

$1 \le k < 4$

$1 \le k < 4$

for $s = 1$ check the min in digonal i.e 1335.

$(B.C)$.

for $s = 2$ check the min digonal i.e

$A(B.C)$

for $s = 3$

$(A(B.C))\ D.$

1. Backtracking:
   - N-Queen's Problem
   - Subset sum problem

2. Branch and ~~bound~~ bound:
   - Travelling salesperson Problem { To reach to every node from a starting node }
   - Knapsack Problem.

Backtracking: [If we solve N-queens problem using brute force then we have to check for N^N solution's validation]

(A) 4- Queens Problem

$Q_1 \rightarrow$	11	12	13	14
$Q_2 \rightarrow$	21	22	23	24
$Q_3 \rightarrow$	31	32	33	34
$Q_4 \rightarrow$	41	42	43	44

* To solve problem under backtracking it follows DFS order.
  (construct solution space tree)
  - ↳ We create a tree (called solution state space tree or solution space tree)
  - ↳ To construct solution state tree following rules/constraints are used & / are followed:

① Implicit constraints ② Explicit constraints

we define these construct for ~~different~~ problems.

For N-Queens Problem: (NXN chess board we have to place N-queens in such a way:

No two queens on same row
No two queens on same column
No two queens on same diagonal. } Implicit constraint

No. of queens should not exceed the size } explicit constraints.

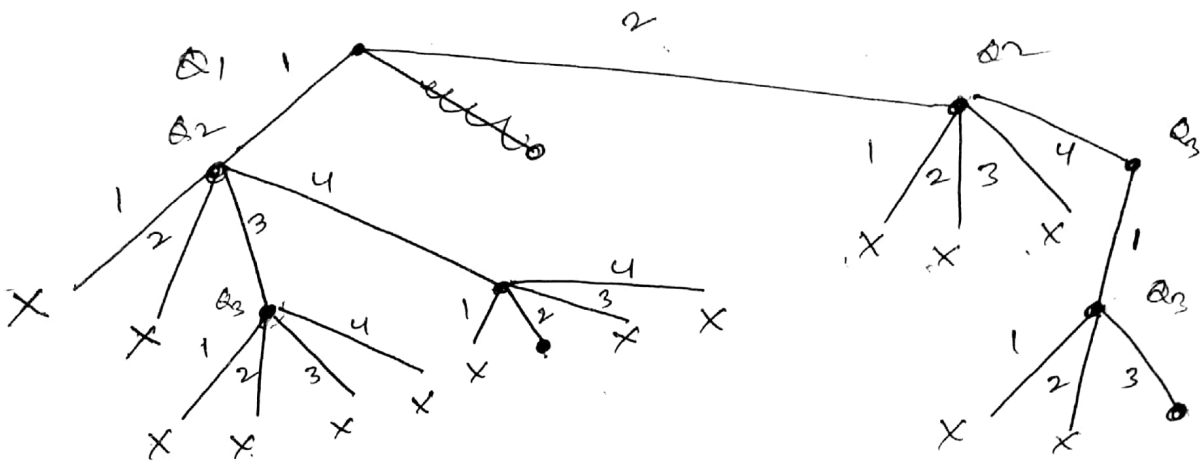
Row represent → Queen number
column represent → position of Queen.

Initially considering chess board is empty.

To check for Row attack.

To check column attack.

To check digonal attack.



$$X = \{2, 4, 1, 3\}.$$

$$X = \{3, 1, 4, 2\}.$$

N-Queen's Problem :

ALGORITHM :

```
NQueen(K,n)
// K → Queen no. and n → total no. of queens.
for i ← 1 to n do
    if ( place(K,i)) then        // whether it is possible to place Kth queen at ith column.
        x[K] = i
        if (K == n) then         // explicit constraint
            write (x[1:n])
        else
            NQueens(K+1,n)
        end if
    end if
end for.
```

```
ALGORITHM Place(k, i)
for (j ← 1 to k-1) do
    if (x[j] == i) OR ABS[x[j] - i) == ABS(j - k)) then
        return false
    return true
    end if
end for
```

Absolute function |x-y| → implicit constraint

row attack / column attack. col diff. row diff.

diagonal attack.

x[] is a unidirectional array which stores the column number (already placed queen column number).

SUBSET SUM PROBLEM:

$S = \{5, 6, 7, 8, 9\}$ sum = 20

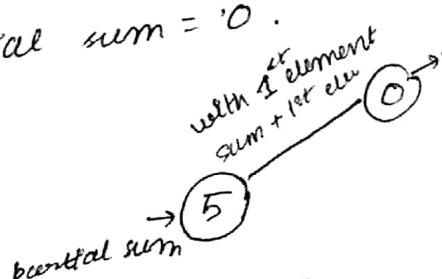
We have to find the subset whose sum is equal to "sum".

Using Brute force approach,
→ We will check for 2^n subset & check for every subset
→ High complexity.

Implicit constraint: The sum of the elements of the subset should be equal to the 'sum'.

Explicit constraint: No. of elements in the set should be less than or equal to N.

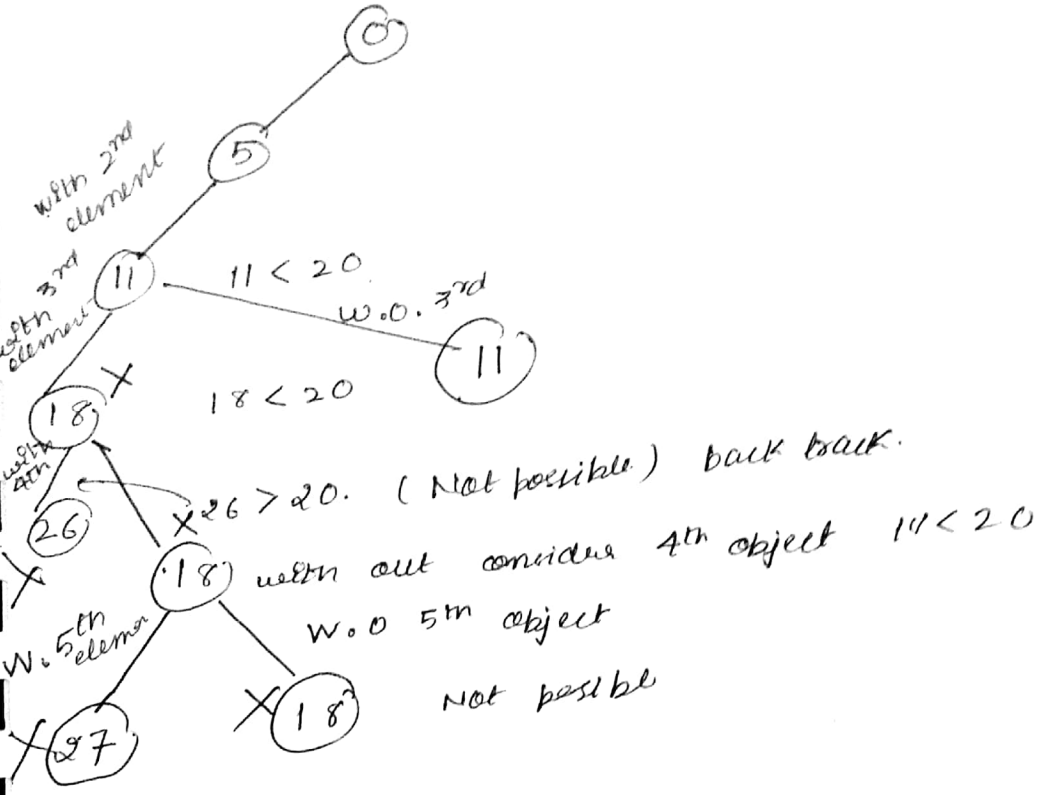
Initial sum = 0.

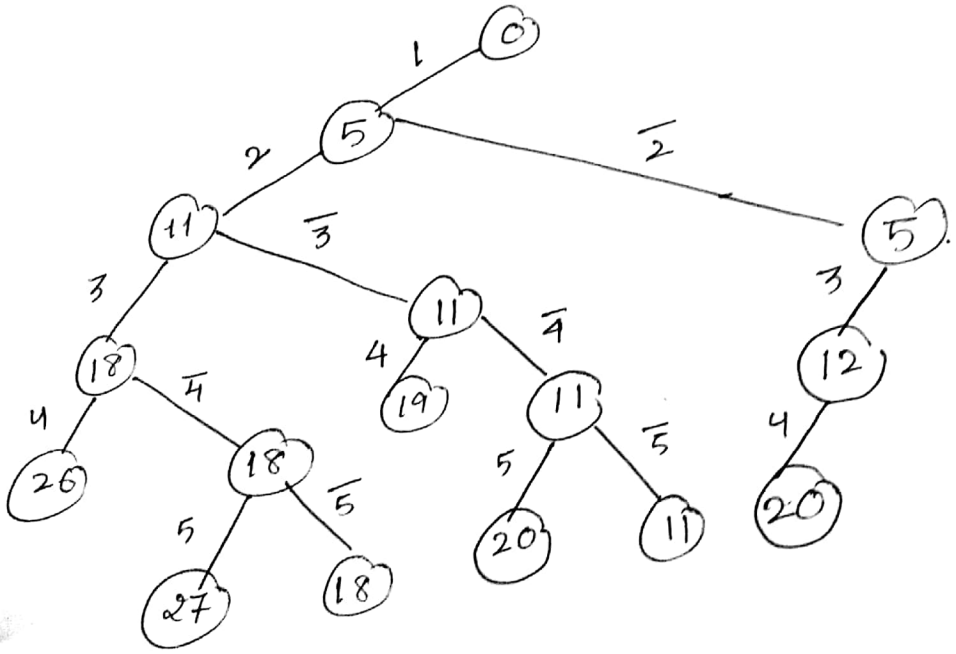


if partial sum < sum then we may get solution in further.

if total sum of S > sum solution may exist

if TS of S < sum
No solution.





$X_1 = \{1\ 1\ 0\ 0\ 1\} = S = \{5, 6, 9\}$

$X_2 = \{1\ 0\ 1\ 1\ 0\}$ $S = \{5, 7, 8\}$.

```
ALGORITHM Sumofsub ( s, K, r )
    if ( sum(w[i]) ≥ m) then   // sum till i ≥ m. we may get the solution
    { x[K] = 1
      if ( s+w[K] == m ) then
        write ( x[1:K] )
      else if ( s+r-w[K]+w[K+1] ≤ m ) then
        sumofsub ( s+w[K], K+1, r-w[K] )
      if ( s+r-w[K] ≥ m ) and ( s+w[K+1] ≤ m ) then
        { x[K] = 0.
          sumofsub (s, K+1, r-w[K] )
        }
      end if
    }
    end if
```

w[] ⇒ holds the list of set element in ascending order.

K ⇒ object number.

r ⇒ sum of all the elements

m ⇒ given sum.

Branch and Bound:

1. Knapsack Problems : (optimization problem).

M = 16.

	n	w	v	v:w
		4	40	10
①	1	7	42	6.~~8~~
②	2	5	25	5
③	3	3	12	4
④	4			

Not consider ② object or w > m →

$UB = V + (M-w)\frac{V_{i+1}}{w_{i+1}}$

To construct a solution space tree we follow BFS order.

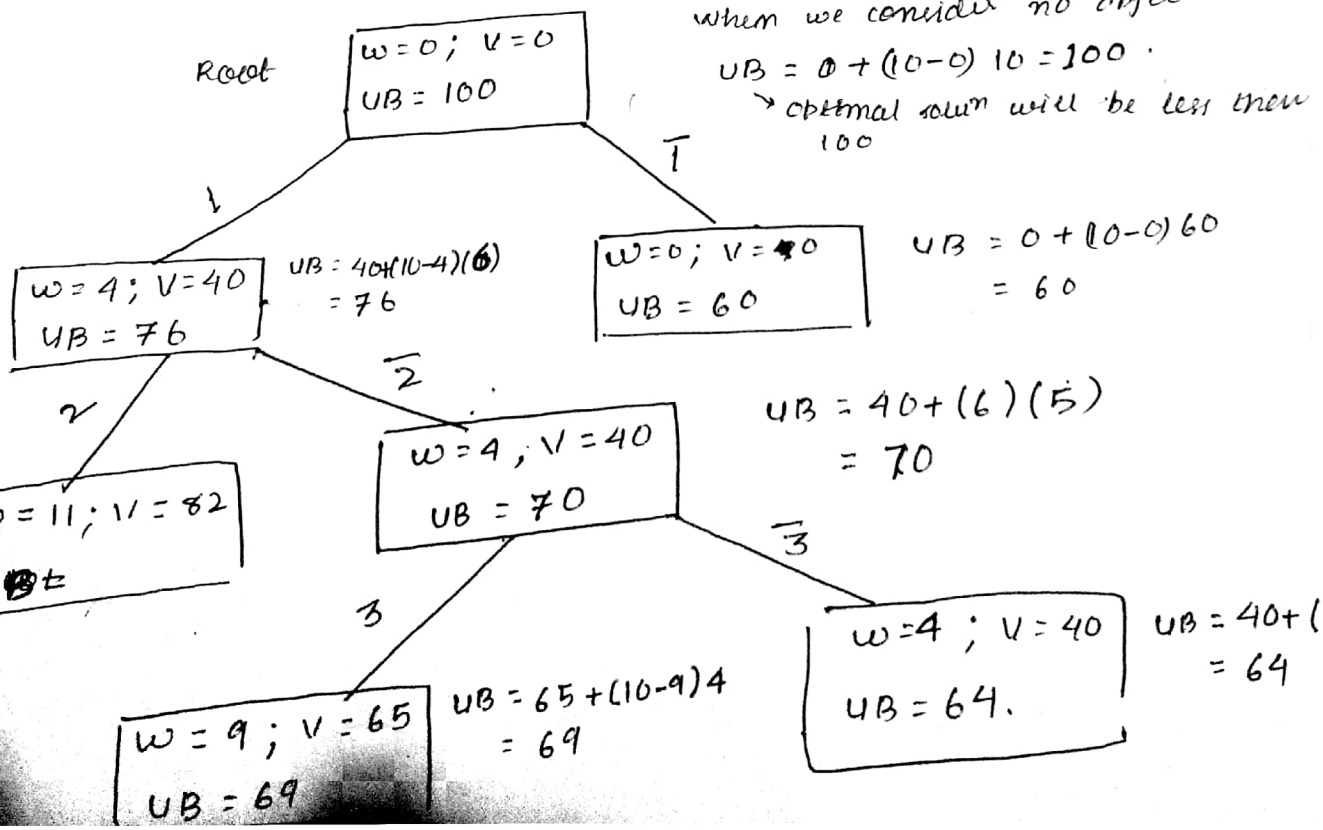
Backtracking.	Branch and Bound.
→ For some set of problem we get one or multiple solution or no solution.	→ For every problem we will get a solution and we get only one solution
→ Use DFS order for construction of solution space tree.	→ Use BFS order for construction of solution tree.
→ To expand tree we check for internal or external constraints.	→ To expand the tree in BnB we use a bound function. Used for optimization, maximization or minimization.

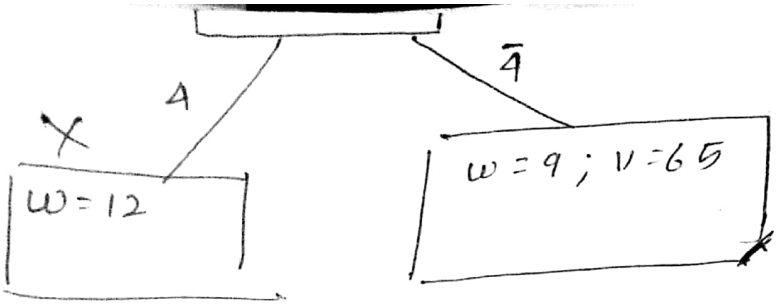
step 1 : Find the V : w ratio.

step 2 : Arrange the objects in descending order of V : w ratio.

* To find the objects that should be enclluded in solution, we will construct state space tree. (Rooted tree).

We will · w → Initial weight of the object
V → Initial profit for the object.
UB → Upper bound (∵ we are optimizing).





can't calculate UB as all the objects can considered.

So, find a path from root to the final leaf node.

Right sub tree → object is considered

left sub tree → objet is not consider

$X = \{1\ 0\ 1\ 0\}$

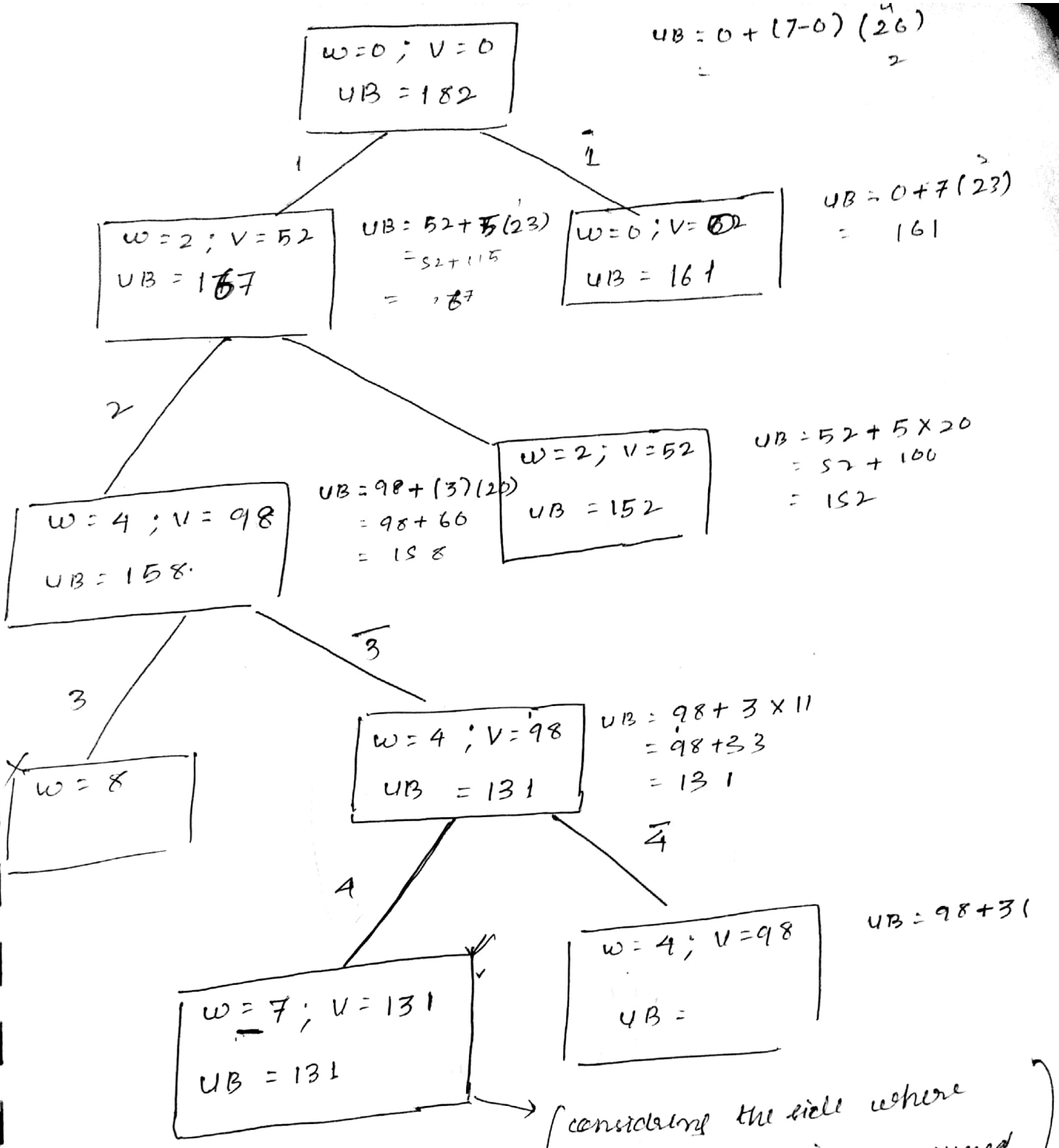
4) Find an optimal solution for the following instance of knapsack problem using branch & bound designing technique.

$m = 7$.

n	w	v	v : w
1	3	33	11
2	2	46	23
3	2	52	26
4	4	80	20

	n	w	v	v : w
①	3	2	52	26
②	2	2	46	23
③	4	4	80	20
④	1	3	33	11

$UB = v + (m - w)\left(\frac{v_{i+1}}{w_{i+1}}\right)$



$X = \{1\ 1\ 0\ 1\} \rightarrow V = \{52 + 46 + 33\} = 131$.

$X = \{1\ 1\ 1\ 0\}$.

$V = 33 + 46 + 52$

$= 131$.

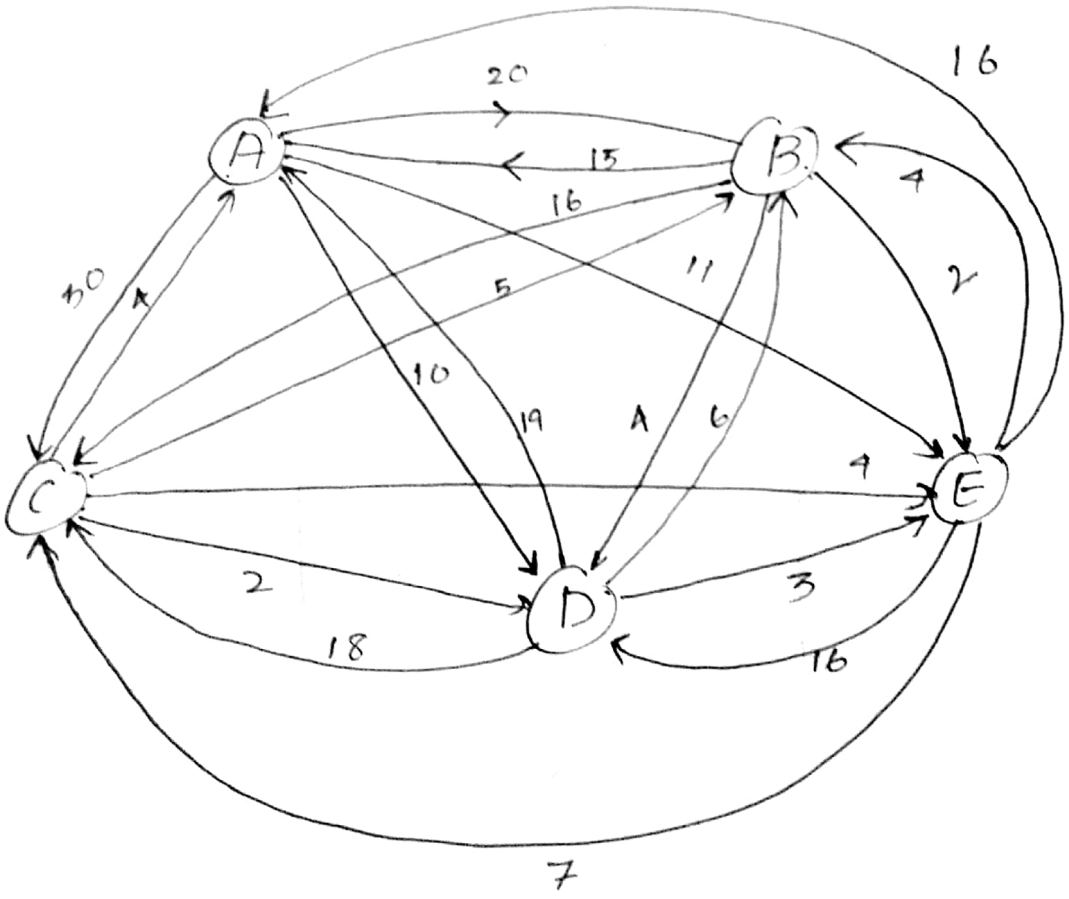
Q. Find an optimal tour for the following instance of travelling salesperson problem using branch and bound designing Technique.

	A	B	C	D	E
A	∞	20	30	10	11
B	15	∞	16	4	2
C	3	5	∞	2	4
D	19	6	18	∞	3
E	16	4	7	16	∞

Travelling Salesperson Problem

↳ minimization problem

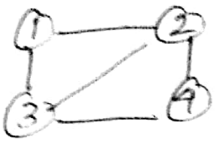
↳ Hamiltonian Path i.e. shortest path from start node to every other node by visiting every node once & come back to starting node.

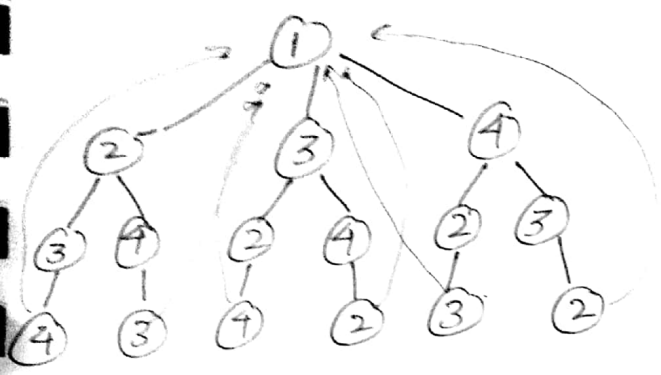


No. of tours : $(n-1)!$ only one is valid.

$n \rightarrow$ no. of nodes.

key for ① node





≡ 6 tour only one 1 is optimal.

- Matrix Reduction Technique :

Step 1 : Identify the minimum value from each row.

Step 2 : Subtract the minimum value corresponding to each row from every element.

$$\begin{bmatrix} - & 20 & 30 & 10 & 11 \\ 15 & - & 16 & 4 & 2 \\ 3 & 5 & - & 2 & 4 \\ 19 & 6 & 18 & - & 3 \\ 16 & 4 & 7 & 16 & - \end{bmatrix} \begin{matrix} 10 \\ 2 \\ 2 \\ 3 \\ 4 \end{matrix} = \begin{bmatrix} - & 10 & 20 & 0 & 1 \\ 13 & - & 14 & 2 & 0 \\ 1 & 3 & - & 0 & 2 \\ 16 & 3 & 15 & - & 0 \\ 12 & 0 & 3 & 12 & - \end{bmatrix}$$

$$\begin{matrix} & 1 & 0 & 3 & 0 & 0 \end{matrix}$$

Step 3 : Identify the minimum value from each column. and subtract the min value correspond to each col. from every element.

$$\begin{bmatrix} - & 10 & 20 & 0 & 1 \\ 13 & - & 14 & 2 & 0 \\ 1 & 3 & - & 0 & 2 \\ 16 & 3 & 15 & - & 0 \\ 12 & 0 & 3 & 12 & - \end{bmatrix} = \begin{bmatrix} - & 10 & 17 & 0 & 1 \\ 12 & - & 11 & 2 & 0 \\ 0 & 3 & - & 0 & 2 \\ 15 & 3 & 12 & - & 0 \\ 11 & 0 & 0 & 12 & - \end{bmatrix}$$

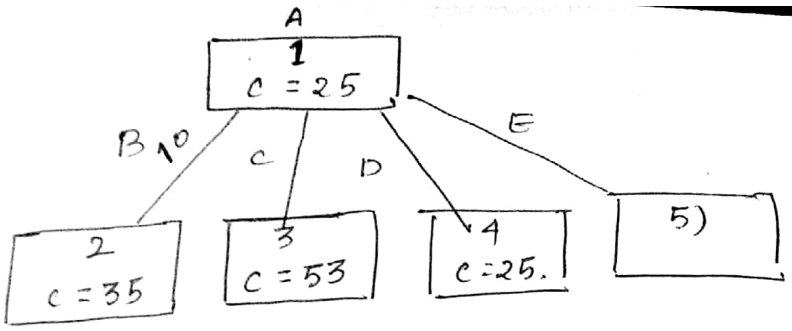
$$\begin{matrix} & 1 & 0 & 3 & 0 & 0 \end{matrix}$$

Lower bound = sum of min of each row + sum of all min of each column

$= (10+2+2+3+4) + (1+0+3+0+0)$.

$= 25$.

Step 5: Root.



5)

2)

i) $C(A,B) = 10$.

ii) $\mathcal{U}_A, C_B = \infty$.

iii) set $(B,A) = \infty$.

$$\begin{bmatrix} - & - & - & - & - \\ - & - & 11 & 2 & 0 \\ 0 & - & - & 0 & 2 \\ 15 & - & 12 & - & 0 \\ 11 & - & 0 & 12 & - \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \quad RCS = 0$$

$0 \quad 0 \quad 0 \quad 0 \quad 0$

$cost(B) = cost(A,B) + cost(1) + RCS$

$= 10 + 25 + 6$

$= 35.$

3) $C(A,C) = 17$

ii) $\mathcal{U}_A, C_C = \infty$

iii) set $(C,A) = \infty$

	A	B	C	D	E	
A	-	-	-	-	-	0
B	12	-	-	2	0	0
C	-	3	-	0	2	0
D	15	3	-	-	0	0
E	11	0	-	12	-	0
	11	0	0	0	0	

$cost(C) = cost(A,C) + cost(1) + RCS$

$= 17 + 25 + 11$

$= 53.$

4) $C(A,D) = 0$

$\mathcal{U}_A, \mathcal{U}_B = 0$

set $(D,A) = 0$

	A	B	C	D	E	
A	-	-	-	-	-	0
B	12	-	11	-	0	0
C	0	3	-	-	2	0
D	-	3	12	-	0	0
E	11	0	0	-	-	0
	0	0	0		0	

$cost(D) = cost(A,D) + cost(1) + RCS$

$= 0 + 25 + 0$

$= 25$

5) cost (A, E) = 1

$r_A, c_E = \infty$

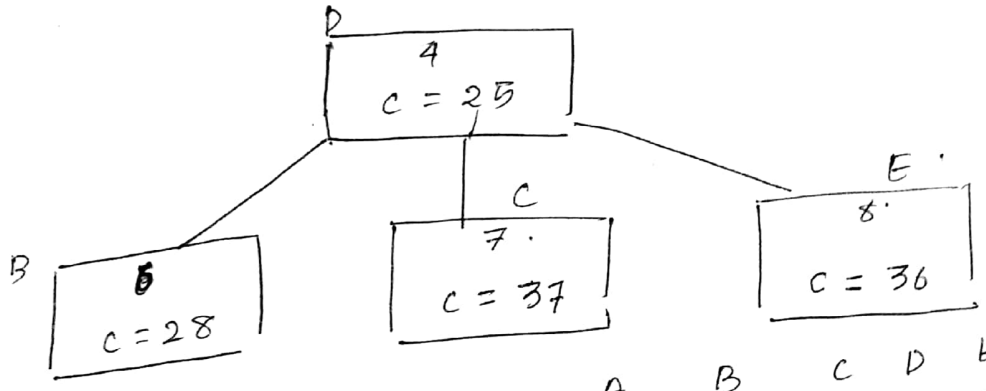
set (E, A) = ∞

	A	B	C	D	E	
A	–	–	–	–	–	0
B	12	–	11	2	–	2
C	0	3	–	0	–	0
D	15	3	12	–	–	3
E	11	0	0	12	–	0
	0	0	0	0	0	

cost (E) = cost (A, E) + RCS + cost (1) RCS = 5.

= 5 + 1 + 25

= 31.



6) cost (D, B) = 3.

$r_D, c_B = \infty$

set (B, D) = ∞

	A	B	C	D	E	
A	∞	∞	∞	∞	∞	0
B	12	∞	11	∞	0	0
C	0	∞	∞	∞	2	0
D	∞	∞	∞	∞	∞	0
E	11	∞	0	∞	0	0
	0	0	0	0	0	

cost (B) = cost (D, B) + cost (D) + RCS

= 3 + 25

= 28.

7) cost (D, C) = 12.

$r_D, c_C = \infty$

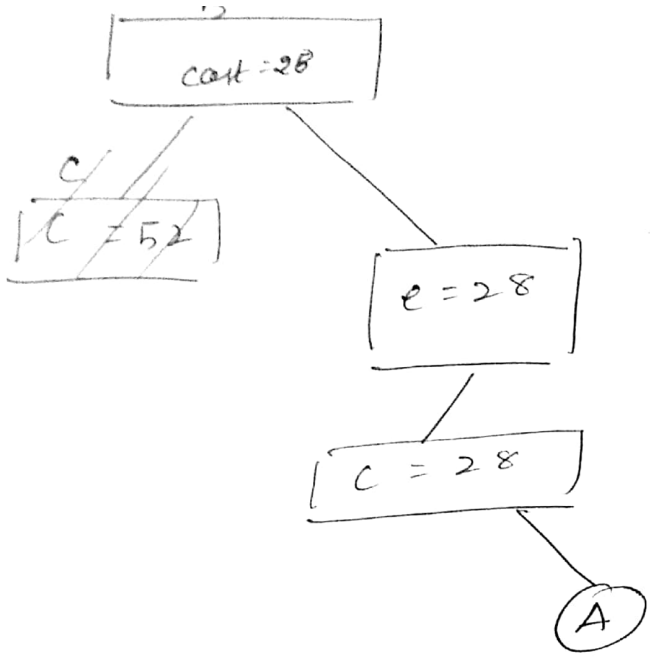
set (C, D) = ∞.

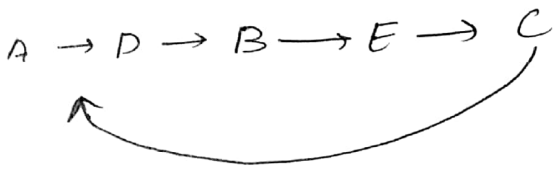
	A	B	C	D	E	
A	–	–	–	–	–	0
B	12	–	–	–	0	0
C	0	3	–	–	2	0
D	–	–	–	–	–	0
E	11	0	–	–	–	0
	0	0	0	0	0	

cost (C) = cost (D, C) + cost (D) + RCS

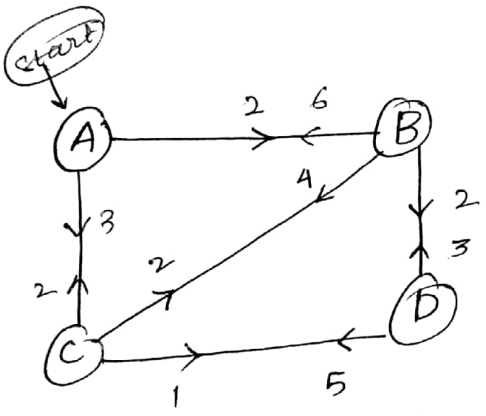
= 12 + 25

= 37





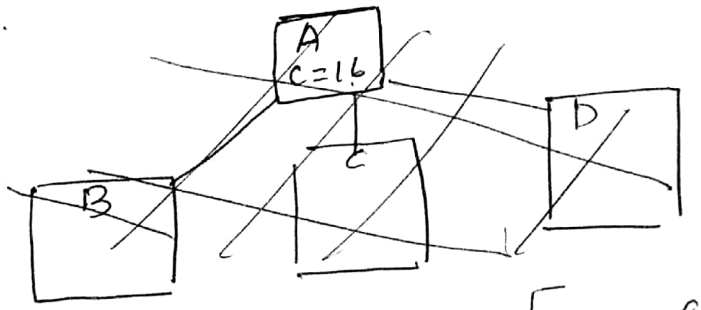
4. o



	A	B	C	D	E
A	-	2	3	∞	2
B	6	-	4	2	2
C	2	2	∞	1	1
D	∞	3	5	∞	3
E	2	2	3	8	8

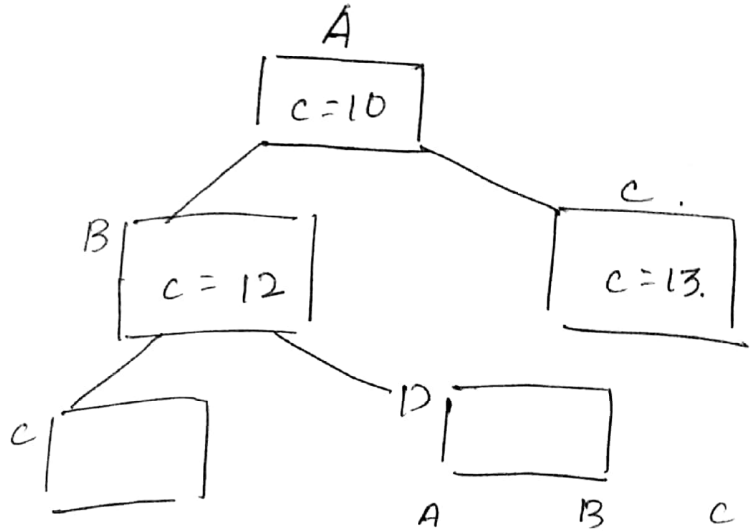
8 + 8 = 16.

LS = 16.



$$\begin{bmatrix} - & 0 & 1 & \infty \\ 4 & - & 2 & 0 \\ 1 & 1 & \infty & 0 \\ \infty & 0 & 2 & \infty \\ \infty & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} - & 0 & 0 & \infty \\ 3 & - & 1 & 0 \\ 0 & 1 & \infty & 0 \\ \infty & 0 & 1 & \infty \end{bmatrix} = 2$$

LB = 8 + 2 = 10



cost (A, B) = 0
α_A, α_B = ∞
cost (B, A) = ∞

	A	B	C	D	
A	∞	∞	∞	∞	0
B	3	∞	1	0	0
C	0	∞	∞	0	0
D	∞	∞	1	∞	1
	0	0	1	0	

cost (B) = cost (A, B) + cost (A) + RCS
= 0 + 10 + 2
= 12.

cost (A, C) = 0
α_A, α_C = ∞
cost (C, A) = ∞.

∞	∞	∞	∞	0
3	∞	∞	0	0
∞	1	∞	0	0
∞	0	∞	∞	0
3	0	0	0	

cost C = cost (A, C) + cost (A) + RCS
= 0 + 10 + 3
= 13.