

⇒ Amortized Cost Analysis:-

- It is also known as average cost analysis.
It is the method of analysing the cost associated with the data structure that averages the worst operations out over time. This is required because one worst case cannot signify the overall performance of an algorithm. So, amortized analysis is used to average out the costly operations in the worst case.

- There are three main types of amortized analysis:

- ① Aggregate Analysis.
- ② Accounting Analysis.
- ③ Potential Analysis.

1) Aggregate Analysis:-

In aggregate analysis, there are two steps, first, we must show that the sequence of n operations takes $T(n)$ time in the worst case. Then, we show that each operation takes $\frac{T(n)}{n}$ time on average. Therefore, in aggregate analysis, each operation has some cost.

Consider an example of Hash Table / Insert.

1	<u>Insert</u>	1	copy → 1	copy → 1	copy → 1	copy → 1
2	<u>Insert</u>	2	copy → 2	copy → 2	copy → 2	copy → 2
3	<u>Insert</u>	3	copy → 3	copy → 3	copy → 3	copy → 3
4	<u>Insert</u>	4	copy → 4	copy → 4	copy → 4	copy → 4
5	<u>Insert</u>				copy → 5	copy → 5
6	<u>Insert</u>				copy → 6	copy → 6
7	<u>Insert</u>				copy → 7	copy → 7
8	<u>Insert</u>				copy → 8	copy → 8

Insert	0	1	2	3	4	5	6	7	8	9	10
size	0	1	2	4	4	8	8	8	8	16	16
cost	C _i	1	2	3	15	1	1	1	1	9	1

Here, C_i is the cost involved, which is the sum of copy + insert.

Now,

$$C_i = \begin{cases} 9 & \text{if } (i-1) \text{ is the exact power of 2} \\ 1 & \text{otherwise.} \end{cases}$$

Again,

$$C_i = n + \sum_{j=0}^{\lfloor \log_2 i \rfloor} 2^j \quad \text{or} \quad C_i = [(1+1+1+1+\dots) + (1+2+4+8+\dots)]$$

As this is the geometric series, it can be generalized as,

$$C_i = n + 2n + 3$$

$$= 3n - 3$$

$$O(n) = n$$

$$\text{Average cost } O(n) = \frac{O(n)}{n} = 1$$

Amortized cost analysis of hash table, if the table doubling is also included.

→ While using dynamic table as array, following are the steps needed to be followed, when table become full.

- i) Allocate memory for twice the old table size.
- ii) Copy the contents of old table to new table.
- iii) free the old table.

If the table doubling cost is also included then,

Item	size	operation cost	copying cost	Doubling cost	Total
1	1	1	0	0	1
2	2	1	1	1	3
3	4	1	2	1	4
4	8	1	0	0	1
5	8	1	4	1	6
6	8	1	0	0	1
7	8	1	0	0	1
8	8	1	0	0	1
9	16	1	8	1	10
10	16	1	0	0	1

Here, $C_i = \text{operation cost} + \text{copying cost} + \text{doubling cost}$

$$\text{i.e. } C_i = n + \sum_{i=0}^{\log_2 n} 2^i + \sum_{j=0}^{\log_2 n - 1} 1^j$$

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

$$= 2n + 2n - 3$$

$$= 4n - 3$$

$$= O(n)$$

$$\therefore C_i = O(n) = O(n)/n = 1 \#$$

2) Accounting Analysis:-

The accounting method is apply named because it borrows ideas and terms from accounting. This method is borrowed from finance where there is the concept of ~~storing~~ storing excess amount in the bank for future use. Here, each operation is assigned a

charge called the amortised cost. Some operation can be charged more or less than they actually cost, we assign the difference called credit. credit can be never be negative in any sequence of operations. This method requires setting the cost of C_p (Amortized Cost), which should be equal for all inserts such that $\text{Bank} \geq 0$ (cost) which is turn will satisfy the inserts of all the input $\hat{C}_p = 3$.

<u>Qn-1</u>		$\leftarrow \text{Bank}_i = \cancel{\text{Bank}_i} + (\hat{C}_p - C_i)$
C_p	3 3 3 3 3 3 3 3 3 3	
Bank _i	2 3 3 5 3 5 7 9 3 1	
C_i	1 2 3 1 5 1 1 1 9 1	
$\hat{C}_p = 3n$		

Q n	1 2 3 4 5 6 7 8 9 10
size _i	1 2 4 4 8 8 8 8 16 16
C_i	1 2 3 1 5 1 1 1 9 1

\hat{C}_p	3 3 3 3 3 3 3 3 3 3
Bank	2 2 3 8 3 8 7 9 13 8

if guess = 2

\hat{C}_p	2 2 2 2 2 2 2 2 2 2
Bank	1 1 0 5 -1

which is invalid.

so, $\hat{C}_p = 3 \leftarrow$ if it is correct guess.

If every 1 operation cost \hat{C}_p 3 then,

n operation cost \hat{C}_p $3n$.

$$\therefore \Theta(n) = \frac{3n}{n} = 3 = 1 \text{ (constant)}.$$

3) Potential ~~Net~~ Analysis - :

Here, we have to define potential function
~~(ϕ)~~ (ϕ).

$$\phi(D_0)$$

$$\phi(D_i)$$

$$\phi(D_{i-1})$$

$$\text{potential difference} = \phi(D_i) - \phi(D_{i-1})$$

$$\begin{aligned} & \text{lets assume } \phi(D_i) = 2^i - 2^{\lceil \log i \rceil} \\ & \cdot \phi(D_{i-1}) = 2^{i-1} - 2^{\lceil \log(i-1) \rceil} \end{aligned}$$

$$\hat{C}_i = C_p + \phi(D_i) - \phi(D_{i-1})$$

Here,

$$C_p = \begin{cases} 9 & \text{if } p-1 \text{ is an exact power of 2} \\ 1 & \text{otherwise.} \end{cases}$$

$$\therefore \hat{C}_i = \begin{cases} p, & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{otherwise} \end{cases} - [2^i - 2^{\lceil \log i \rceil}] - [2^{i-1} - 2^{\lceil \log(i-1) \rceil}]$$

Case I :-

$$\hat{C}_4 = 9 - 2^4 - 2^{\lceil \log 4 \rceil} - 2^3 - 2 - 2^{\lceil \log(3-1) \rceil}$$

$$= 9 - 2(9-1) + 2 + (3-1) \text{ generalisation.}$$

$$= 9 - 2^p + 2 + 2 + 9 - 1$$

$$= 4 - 1$$

$$= 3$$

Case II :-

$$\hat{C}_1 = 1 - 2^{\lceil \log 1 \rceil} + 2 + 2^{\lceil \log(1-1) \rceil}$$

$$= 1 - (1-1) + 2 + (1-1) \text{ (generalization)}$$

$$= 1 - 1 + 1 + 2 + 1 - 1$$

$$= 3$$

⇒ Randomized Algorithm :-

- A probabilistic algorithm is an algorithm where the result and/or the way the result is obtained depend on chance. These algorithms are also sometimes called randomized algorithms.
- It makes use of randomizer (random number generator) to make some decision in a algorithm.
- The output and/or execution time of a randomized algorithm could also differ from run to run for the same input.
- It can be categorized into :
 - 1) Las Vegas Algorithm
 - 2) Monte Carlo algorithm

⇒ Las Vegas Algorithm :-

- The algorithm always same (correct) output for the same input.
- Execution time of this algorithm is characterized as a random variable.
- The execution time depends on the output of ~~random~~ randomizer.
- Type of Randomized quick sort.
- It guarantees the correctness of the solution.
- Upper bound cannot be fixed.

⇒ Monte Carlo Algorithm :-

- It is a algorithm whose output might differ from run to run for the same input.
- It does not guarantees the correctness of the solution.
- Type of primility testing.
- Upper bound can be fixed.
- The execution time is fixed.

~~Las Vegas Search~~ \Rightarrow Las Vegas Algorithm

while (true) {

void LasVegas () {

{

while (1) {

int i = random () % 2;

if (i) return ;

}

}

LasVegasSearch (A, n) {

while (true) {

{

randomly choose an element out of n.

if (a found)

return true ;

}

}

\Rightarrow monte Carlo Algorithm - :

MonteCarloSearch (A, n, x) {

P=0

while ($P \leq x$) {

randomly choose an element ~~out~~ out of n.

if (a found) {

flag = true ;

}

return flag ;

}

\Rightarrow primality Testing :-

- to check whether the given number is prime or not.

\Rightarrow Bruth force Algorithm

if n is the no to be checked

i = 2

for P = 2 to n-1

$n \% i = 0$

$n == 0$

prime

not prime .

$\rightarrow \text{if } a^{n-1} \% n == 1$
 prime

else
 not prime.

n is the number to be checked.

$a < n$ (the numbers)

let $n = 2$

$a = 1$

$$\Rightarrow 1^{2-1} \% 2 \Rightarrow 1 \% 2 \neq 1$$

$n = 3$

$a = 1, 2$

$$\text{and } 2^{3-1} \% 3 = 4 \% 3$$

$$\Rightarrow 1^{3-1} \% 3 = 9 \% 3 = 1 \neq 1$$

$n = 4$

$a = 1, 2, 3$

$$\text{if } a=1 \Rightarrow 1^{4-1} \% 4 = 1$$

$$\text{if } a=2 \Rightarrow 2^{4-1} \% 4 = 0 \Rightarrow \text{complexity } O(1)$$

$$\text{if } a=3 \Rightarrow 3^{4-1} \% 4 = 3 \neq 0$$

monte carlo

\Rightarrow Primality Testing :-

- Any integer greater than one is said to be a prime if its only divisors are 1 and the integer itself. By convention, we take 1 to be a non-prime, then 2, 3, 5, 7, 11, and 13 are the first six primes. Given an integer n , the problem of deciding whether n is a prime is known as primality testing.

Las Vegas () {

 while (true) do {

r = Random () mod 2;

 if (*r* >= 1) then return;

}

Unit - 1.2 : Advance Algorithm Design Techniques:-

→ Greedy Algorithms :-

- It is a simplest and straight forward method (approach).
- It takes decision on the basis of current available information without worrying about the effect of the current decision in future.
- It works in phases. At each phase :
 - You take the best you can get right now, without regard from future consequences.
 - You hope that by choosing a local optimum at each step.
 - You will end up at a global optimum.
- At each stages decision is made regarding whether particular input is an optimal solution.

Greedy (A[], n) {

 for (i=1; i < n) {

 x = select (A)

 if (Feasible x) {

 solution = solution + x;

}

 return solution;

}

e.g. Huffman coding, Shannon Fano Algorithm
Job scheduling etc.

- lossless data encoding schemes is disadvantageous.
- Relatively slower process.

- tags

⇒ Huffman Coding :-

- By David A. Huffman (1949)
- Huffman coding is a very popular algorithm for encoding data.
- It is greedy (best for greedy approach) algorithm.
- It reduces the average access time of codes as much as possible.
- It is a unique code generator algorithm.
- It generates always prefix code (optimal prefix).
- It is a lossless data compression algorithm.
- It uses variable length code words to represent characters in a message.
- Shorter code words are assigned to more frequent characters, and longer code words to less frequent characters.
- This reduces the total no of bits required to represent the message.
- Algorithm constructs the binary tree of nodes where each leaf node represents a character in the message.
- The frequency of each character is used to determine the position of nodes in the tree.
- Characters with higher frequency are closer to the root, while those with lower frequency are farther from the root.
- Root to leaf node to generate binary code.
- The resulting code words are prefix-free, ensuring unique decoding.
- Huffman coding are used in data compression, image and video compression, and file archiving.
- e.g. best is Huffman coding for greedy.

Example - 1

String = 'aaaaa bbb e f ggggppp'

frequency = {a=4, b=3, e=1, f=1, g=6}

Sol:

A character is 1 byte long = 8-bit

$\therefore 8 \times 19 = 152$ bit ($19 \rightarrow$ total string)

code = a = 01 = 2

b = 110 = 3

Total bits = (frequency of 'i') * length of code for 'i'
 + (frequency of 'a') * length of code for 'a' + ...
 $= 6 \times 2 + 4 \times 2 + 1 \times 2 + 3 \times 3 + 1 \times 1 + 4 \times 1 = 45$ bits.

Average code length = Total no of bits / total no of symbols
 $= 45 / 19 = 2.37$

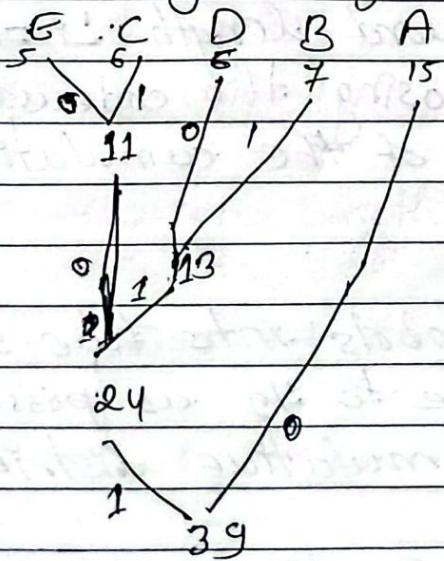
\therefore Average code length is 2.37 bits per symbol
 as compared to 8-bit per symbol before encoding.

Example - 2

symbol	A	B	C	D	E
count	15	7	6	6	5

Sol:

Sorting all symbols first and use Huffman coding.



$$\Rightarrow A = 0 = 1$$

$$B = 111 = 3$$

$$C = 101 = 3$$

$$D = 110 = 3$$

$$E = 100 = 3$$

$$\therefore \text{Average length} = 1 \times 15 + 3 \times 7 + 3 \times 6 + 3 \times 6 \\ = 87$$

$$\text{Average bit length} = 87 / 39 = 2.23 \text{ bits}$$

$$\text{Entropy } (H) = \sum_{i=1}^n p_i \log (1/n).$$

$$\text{or} - \sum_{i=1}^n p_i \log (p_i)$$

$$\text{eg Hello} \Rightarrow H = 1/5, e = 1/5, l = 2/5, o = 1/5$$

$$\therefore H = -(1/5) \log_2(1/5) - (1/5) \log_2(1/5) - (2/5) \log_2(2/5) \\ - (1/5) \log_2(1/5)$$

$$\Rightarrow H = \dots$$

→ Shannon-Fano coding:-

- Related to field of data compression.
- credit shannon and Robert Fano - scientist
- Related techniques for constructing a prefix code based on a set of symbols and their ~~probabilities~~ probabilities. (estimated or measured).

→ Shannon's method:-

- chooses a prefix code where a source symbol 'r' is given the codeword length $L_r = [-\log_2 p_i]$
- one common way of choosing the codewords uses the binary expansion of the cumulative probabilities.

→ Fano's method:-

- divides the source symbols into two sets (odd/even) with probabilities as close to 1/2 as possible.
- It is based on the cumulative distribution function.
- It provides slow optimization.

Example :-

m.	a	b	c	d	e
P	0.3	0.15	0.15	0.2	0.2

SFT¹⁰

	0.3	0.15	0.15	0.2	0.2
0.3 (a)	0	0			
0.2 (d)	0	1			
0.2 (e)	1	0			
0.15 (b)	1	1	0		
0.15 (c)	1	1	1		

Example :- Applying shannon fano coding for following message.

$$[M] = [m_1 \ m_2 \ m_3 \ m_4 \ m_5 \ m_6]$$

$$[P] = [0.30 \ 0.25 \ 0.15 \ 0.12 \ 0.08 \ 0.10]$$

SFT¹⁰

M	P		Stage 1	Stage 2	Stage 3	Code word	Avg length
m_1	0.30	0.55	0	0		00	2
m_2	0.25		0	1		01	2
m_3	0.15		1	0	0	100	3
m_4	0.12	0.45	1	0	1	101	3
m_5	0.10		1	1	0	110	3
m_6	0.08		1	1	1	111	3

$$\text{Average length } L = \sum_{k=1}^n p_k h_k = \sum_{k=1}^6 p_k h_k$$

$$= (0.30 \times 2) + (0.25 \times 2) + (0.15 \times 3) + (0.12 \times 3) + (0.10 \times 3) + (0.08 \times 3) = 2.48 \text{ bits}$$

$$\therefore L = 2.48 \text{ bits}$$

\Rightarrow Job scheduling with deadlines:-

- It has unique processor machine.
- Each job ' J_i ' requires one unit time to complete.
- Each job ' J_i ' has d_i deadlines and P_i - profits.
- J_i is associated with P_i iff it meets its deadline.
- Find the subset of jobs and its sequence, so that its profit is maximum.
- A machine can provide service to any one job at a time.
- The greedy approach to this problem involves sorting the job by their profits. in descending order and then scheduling them in the order in which they appear in the sorted list, as long as their deadlines have not been exceeded.

Algorithm:-

- Arrange all job J_p in descending order with respect to P_i .
- Allocate timeslot with respect to maximum deadlines.
- $k = \min(\max(\text{deadline}), d_i)$
- Allocate the time slot for J_i with respect to value of k .
- If the time slot is already occupied then search for the positions before it.
- If all the time slot is occupied publish the set and sequence and max profit.
- It has uniprocessor, no-preemption, and every job takes one unit of time.

n = maximum deadline & m = total job.
 \therefore total time = $n \times m$.

Example - 1

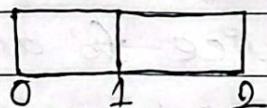
	Job (J_i)	J_1	J_2	J_3	J_4
Profit (P_i)	100	10	15	27	
deadline (d_i)	2	1	2	1	

sof

Time slot = max deadline \rightarrow max deadline = 2

$S_1 \quad S_2$

Since time slot = 2.

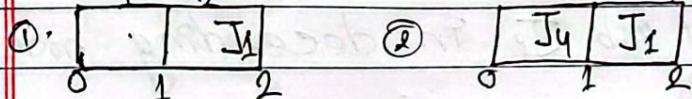


descending order sorting:

J_i	J_1	J_4	J_3	J_2
P_i	100	27	15	10
d_i	2	1	2	1

Iteration - 1 $i=1$

$$K = \min(\max(\text{deadline}), d_i) = \min(2, 2) = 2.$$



\rightarrow set = { J_1, J_4 }

Sequen = { J_4, J_1 } \Rightarrow profit max = 27 + 100
 $= 127$ #

Iteration - 2: $i=2$

$$k = \min(\max(\text{deadline}), d_i) = \min(2, 1) = 1 \neq$$

Example :

J_i	J_1	J_2	J_3	J_4
P_i	30	15	75	80
d_i	1	2	3	2

sof descending order - sorting:

J_i	J_4	J_3	J_1	J_2
P_i	80	75	30	15
d_i	2	3	1	2

Total time slot = 3

for $i=1$

$$k = \min(\max(\text{deadline}), d_i) = \max(3, 2) = 2$$

$\Rightarrow J_4$ in slot 2

J_1	J_4	J_3
0	1	2

$$\Rightarrow 30 + 80 + 75 = 185 \quad \# \text{ profit max.}$$

for $i=2$

$$k = \min(\max(\text{deadline}), d_i) = \min(3, 3) = 3 \quad \#$$

for $i=3$

$$k = \min(\max(\text{deadline}), d_i) = \min(3, 1) = 1 \quad \#$$

$$\Rightarrow \text{Set} = \{J_1, J_3, J_4\}$$

$$\text{Sequence} = \{J_1, J_4, J_3\}$$

$$\text{profit max} = 30 + 80 + 75 = 185 \quad \#$$

\Rightarrow Analysis :-

Case-I :- Each time slot calculated is vacant, so just insert it required which is equal to the no of insertion. $\therefore O(n) = n \quad \#$

Case-II :- Time slot calculated is not vacant, so required another iteration to find the vacant slot. $\therefore O(n) = n(n-1) \Rightarrow n^2 \quad \#$

Greedy Job (d, J, n) {

$J = \{J\}$

for $i=2$ to n do {

• If all deadlines in $J \cup \{J_i\}$ are feasible then

$J = J \cup \{J_i\}$

} return J

Algorithm

\Rightarrow Tree Vertex splitting - (TVS) -

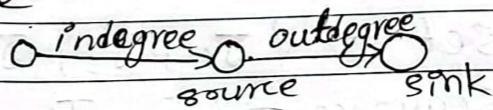
- Consider directed binary tree each edge is leveled with weights.
- TVS can be defined as:
Let $T = (V, E, W)$ be a weighted directed tree.
where,

$V \rightarrow$ set of vertices

$E \rightarrow$ set of edges

$W_{(i,j)} \rightarrow$ weight of edge $\langle i, j \rangle \in E$. and

- Source vertex has in-degree '0' and sink vertex has out degree .



- we define delay of path 'p' denoted as
 $d(p) = \text{sum of weight of path}$
 \Rightarrow Delay of Prece., $d(T) = \max(d(p))$

- Let $T = (V, E, W)$ be a directed tree. A weight tree can be used to model a distribution network in which electrical signal are transmitted.
- The transmission of power from node to another may result in some loss.
- Each edge in the tree is labeled with the loss that occurs in traversing that edge.
- The network may not be able to tolerate losses beyond a critical limit.
- In places where the loss exceeds the tolerance level, boosters have to be placed.
- Given a network and a loss tolerance level, the tree vertex splitting problem is to

determine an optimal placement of boosters.

- A greedy approach to solve this problem is to compute for each node $u \in V$, the maximum delay $d(u)$ from u to any other node in its subtree.
- If u has a parent v such that $d(u) = \max\{d(v) + w(\text{parent}(u), v)\}$
 $d(u) \rightarrow$ delay of node.
 $\forall v \in$ set of all edges & v belongs to $\text{child}(u)$
 δ tolerance value.

- Algorithm -

Algorithm TVS(T, δ)

if ($T \neq \emptyset$) then

{

$d[T] = 0$;

for each child v to T do

{

 Tvs(v, δ);

$d[T] = \max\{d[T], d[v] + w(T, v)\}$;

}

if ((T is not the root) and ($d[T] + w(\text{parent}(T), T) \geq \delta$)) then

{

 write $[T]$;

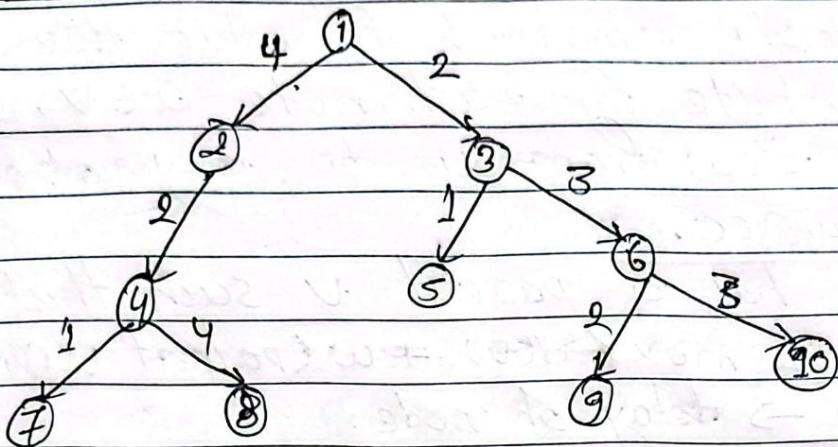
$d[T] = 0$;

}

Analyze -

- Visit each vertex n ;
- In each vertex there is constant time required for calculation and computing its depth δ . $\therefore O(n) = n$

Example -



If $d(u) \geq f$, then place the booster.

Sol

Here all leaf nodes delay = 0 and leaf nodes are 7, 8, 9, 10, 5.

$$f = 5$$

$$d[7] = \max \{ 0 + w(4,7) \} = 1$$

$$d[8] = \max \{ 0 + w(4,8) \} = 4$$

$$d[9] = \max \{ 0 + w(6,9) \} = 2$$

$$d[10] = \max \{ 0 + w(6,10) \} = 3$$

$$d[5] = \max \{ 0 + w(3,5) \} = 1$$

$$\Rightarrow d[7], d[8], d[9], d[10], d[5] \leq f = 5$$

$$d[4] = \max \{ 1 + w(2,4) + 4 + w(2,4) \} \\ = \max \{ 1 + 2, 4 + 2 \} = 6 > f \text{ booster}$$

$$d[6] = \max \{ 2 + w(3,6) + 3 + w(3,6) \} \\ = \max \{ 2 + 3, 3 + 3 \} = 6 > f \text{ booster}$$

$$d[2] = \max \{ 6 + w(1,2) \} \\ = \max \{ 6 + 4 \} = 10 > f \text{ booster}$$

$$d[3] = \max \{ 1 + w(1,3), 6 + w(1,3) \} \\ = \max \{ 3, 8 \} = 8 > f \text{ booster}$$

\therefore No need to find tolerance value for node 1 because from source only power is transmitting.

⇒ Dynamic Programming :-

- It is a technique in computer programming used to solve optimized problems by breaking down a large problem into smaller problems and solving each sub-problem only once.
- principle of optimal substructure which means that the optimal solution to a large problem can be found by combining the optimal solutions to smaller sub-problems.
- Dynamic programming is efficient in finding optimal solutions for cases with lots of overlapping sub-problems.
- It solves problems by recombining solutions to sub-problems, when the sub-problems themselves may share sub-sub-problems.

~~e.g.~~ Fibonacci number -

1, 1, 2, 3, 5, 8, 13, 21, 34, - - -

~~solⁿ~~ def fib(n):

 if n == 0 or n == 1:

 return n

 else:

 return fib(n-1) + fib(n-2)

- Dynamic programming is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decision.

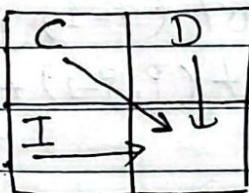
⇒ Difference between Greedy vs Dynamic Algorithm:-

Greedy Algorithm	Dynamic Algorithm
→ makes locally optimal choices at each step	→ Breaks down a problem in smaller subproblems.

→ Does not guarantee an optimal solution.	→ Guarantee an optimal solution.
→ Faster than dynamic programming.	→ slower than greedy programming.
→ may not consider all possible choices	→ consider all possible choices.
→ Works well when the problem has the greedy choice property.	→ can solve a wide range of problems.
→ memory efficient as it only needs to store the current solution.	→ memory intensive, as it needs to store solutions to sub-problems.
• Example: Huffman coding, Kruskal's algorithm etc.	Example -: knapsack problem, Fibonacci sequence. etc.

⇒ String Editing :-

- There are two string source and destination. The goal of the problem is to convert source to destination by applying minimum edit operations on string 'source'.
- we have given two string $x = x_1, x_2, x_3, \dots, x_n$ and $y = y_1, y_2, y_3, \dots, y_m$; where $x_i = 1 \leq i \leq n$ and $y_j = 1 \leq j \leq m$ are number of finite set of symbols known as the alphabets, we want's to transform x to y using a sequence of edit operation on x .
- The permissible edit operations are!
 - Delete — 1 cost → denote $D(x_i)$
 - change / copy — 2 cost → denote $c(x_i, y_j)$
 - Insert — 1 cost → denote $I(y_j)$
- The problem of string editing is to identify a minimum cost sequence of edit operations that will transform x into y .



Example :-

String-1 $\rightarrow a a b a b$ (Source)

String-2 $\rightarrow b a b b$ (destination)

Total cost = 5 (source) + 4 (destination) = 9 #

Technique-1:- Remove all characters within string-1,

Insert the required string:

$\begin{matrix} A & A & B & A & B \\ D & D & D & D & D \end{matrix} \rightarrow b \rightarrow a \rightarrow b \rightarrow b$
 $I \quad I \quad I \quad I \quad I \Rightarrow \text{cost} = 9 *$

Technique - 2 :-

~~a d b a b~~ → first Insert
 D D I $\Rightarrow \text{cost} = 3$

Technique - 3 :-

~~a b a b~~ Replace with b
 R D $\Rightarrow \text{cost} = 2 + 1 = 3 \#$

Technique - 4 :-

~~a b a b a b~~
 R R D $\Rightarrow \text{cost} = 5 \#$

Technique - 5 :-

~~b a a b a b~~
 I D D $\Rightarrow \text{cost} = 3$

$$\text{cost}(P, J) = \begin{cases} 0 & P = J = 0 \\ \text{cost}(P-1, 0) + D(x_i) \cdot i > 0; J = 0 \\ \text{cost}(0, J-1) + I(y_j) \cdot J > 0; P = 0 \end{cases}$$

or

$$\text{cost}(P, J) = \min \begin{cases} \text{cost}(P-1, J) + D(x_i) & P > 0 \\ \text{cost}(P-1, J-1) + C(x_i, y_j) \cdot J & P > 0 \\ \text{cost}(P, J-1) + I(y_j) \cdot J & J > 0 \end{cases}$$

Example :-

Source = a a b a b

destination = b a b b

Step	q	Null	a ₁	a ₂	b ₃	b ₄
	0	0 → 1	1	2	3	4
-	b ₁	1 → 2	1	2	3	
-	b ₂	2 → 3	2	3	4	
-	b ₃	3 → 2	3	2	3	
-	b ₄	4 → 3	2	3	3	4
-	b ₅	5 → 4	4	3	2 → 3	

Step 1: Case $i=0, j=0$, $\text{cost}(0,0) = 0$

Case $i > 1, j=0$

$$\text{cost}(0,1) = \text{cost}(0,1-1) + I(y_1) = 0+1 = 1$$

$$\text{cost}(0,2) = \text{cost}(0,2-1) + I(y_2) = 1+1 = 2$$

$$\text{cost}(0,3) = \text{cost}(0,3-1) + I(y_3) = 2+1 = 3$$

$$\text{cost}(0,4) = \text{cost}(0,4-1) + I(y_4) = 3+1 = 4$$

Step 2

$$\text{cost}(1,0) = \text{cost}(1-1,0) + D(x_1) = 0+1 = 1$$

$$\text{cost}(2,0) = \text{cost}(2-1,0) + D(x_2) = 1+1 = 2$$

$$\text{cost}(3,0) = \text{cost}(3-1,0) + D(x_3) = 2+1 = 3$$

$$\text{cost}(4,0) = \text{cost}(4-1,0) + D(x_4) = 3+1 = 4$$

$$\text{cost}(5,0) = \text{cost}(5-1,0) + D(x_5) = 4+1 = 5$$

Row 1

$$\begin{aligned} \text{cost}(1,1) &= \min \left\{ \begin{array}{l} \text{cost}(1-1,1) + D(x_1) \\ \text{cost}(1-1,1-1) + c(x_1, y_1) \rightarrow \text{change cost} \\ \text{cost}(1,1-1) + I(y_1) \end{array} \right. \\ &= \min \{ 1+1, 0+2, 1+1 \} = 2 \# \end{aligned}$$

$$\begin{aligned} \text{cost}(1,2) &= \min \left\{ \begin{array}{l} \text{cost}(1-1,2) + D(x_1) \\ \text{cost}(1-1,2-1) + c(x_1, y_2) \\ \text{cost}(1,2-1) + I(y_2) \end{array} \right. \\ &= \min \{ 2+1, 1+0, 2+1 \} = 1 \# \end{aligned}$$

$$\begin{aligned} \text{cost}(1,3) &= \min \left\{ \begin{array}{l} \text{cost}(1-1,3) + D(x_1) \\ \text{cost}(1-1,3-1) + c(x_1, y_3) \\ \text{cost}(1,3-1) + I(y_3) \end{array} \right. \\ &= \min \{ 3+1, 2+2, 1+1 \} = 2 \# \end{aligned}$$

$$\begin{aligned} \text{cost}(1,4) &= \min \left\{ \begin{array}{l} \text{cost}(1-1,4) + D(x_1) \\ \text{cost}(1-1,4-1) + c(x_1, y_4) \\ \text{cost}(1,4-1) + I(y_4) \end{array} \right. \\ &= \min \{ 4+1, 3+2, 2+1 \} = 3 \# \end{aligned}$$

Row-2 $\text{cost}(2,1) = \min \left\{ \begin{array}{l} \text{cost}(2-1,1) + D(x_2) \\ \text{cost}(2-1,1-1) + C(x_2, y_1) \\ \text{cost}(2,1-1) + I(y_1) \end{array} \right\}$

$$\min \{ 2+1, 1+2, 2+1 \} = 3 \#$$

$\text{cost}(2,2) = \min \left\{ \begin{array}{l} \text{cost}(2-1,2) + D(x_2) \\ \text{cost}(2-1,2-1) + C(x_2, y_2) \\ \text{cost}(2,2-1) + I(y_2) \end{array} \right\}$

$$= \min \{ 1+1, 2+0, 3+1 \} = 2 \#$$

$\text{cost}(2,3) = \min \left\{ \begin{array}{l} \text{cost}(2-1,3) + D(x_2) \\ \text{cost}(2-1,3-1) + C(x_2, y_3) \\ \text{cost}(2,3-1) + I(y_3) \end{array} \right\}$

$$= \min \{ 2+1, 1+2, 2+1 \} = 3 \#$$

$\text{cost}(2,4) = \min \left\{ \begin{array}{l} \text{cost}(2-1,4) + D(x_2) \\ \text{cost}(2-1,4-1) + C(x_2, y_4) \\ \text{cost}(2,4-1) + I(y_4) \end{array} \right\}$

$$= \min \{ 4+1, 2+2, 3+1 \} = 4 \#$$

Row-3 $\text{cost}(3,1) = \min \left\{ \begin{array}{l} \text{cost}(3-1,1) + D(x_3) \\ \text{cost}(3-1,1-1) + C(x_3, y_1) \\ \text{cost}(3,1-1) + I(y_1) \end{array} \right\}$

$$= \min \{ 3+1, 2+0, 3+1 \} = 2 \#$$

$\text{cost}(3,2) = \min \left\{ \begin{array}{l} \text{cost}(3-1,2) + D(x_3) \\ \text{cost}(3-1,2-1) + C(x_3, y_2) \\ \text{cost}(3,2-1) + I(y_2) \end{array} \right\}$

$$= \min \{ 2+1, 3+2, 2+1 \} = 3 \#$$

$$\begin{aligned} \text{cost}(3,3) &= \min \left\{ \begin{array}{l} \text{cost}(3-1, 3) + D(x_3) \\ \text{cost}(3-1, 3-1) + C(x_3, y_3) \\ \text{cost}(3, 3-1) + I(y_3) \end{array} \right\} \\ &= \min \{ 3+1, 2+0, 3+1 \} = 2 \# \end{aligned}$$

$$\begin{aligned} \text{cost}(3,4) &= \min \left\{ \begin{array}{l} \text{cost}(3-1, 4) + D(x_3) \\ \text{cost}(3-1, 4-1) + C(x_3, y_4) \\ \text{cost}(3, 4-1) + I(y_4) \end{array} \right\} \\ &= \min \{ 2+1, 3+0, 2+1 \} = 3 \# \end{aligned}$$

Row-4

$$\begin{aligned} \text{cost}(4,1) &= \min \left\{ \begin{array}{l} \text{cost}(4-1, 1) + D(x_4, y_1) \\ \text{cost}(4-1, 1-1) + C(x_4, y_1) \\ \text{cost}(4, 1-1) + I(y_1) \end{array} \right\} \\ &= \min \{ 2+1, 3+2, 4+1 \} = 3 \# \end{aligned}$$

$$\begin{aligned} \text{cost}(4,2) &= \min \left\{ \begin{array}{l} \text{cost}(4-1, 2) + D(x_4) \\ \text{cost}(4-1, 2-1) + C(x_4, y_2) \\ \text{cost}(4, 2-1) + I(y_2) \end{array} \right\} \\ &= \min \{ 3+1, 2+0, 3+1 \} = 2 \# \end{aligned}$$

$$\begin{aligned} \text{cost}(4,3) &= \min \left\{ \begin{array}{l} \text{cost}(4-1, 3) + D(x_4) \\ \text{cost}(4-1, 3-1) + C(x_4, y_3) \\ \text{cost}(4, 3-1) + I(y_3) \end{array} \right\} \\ &= \min \{ 2+1, 3+2, 2+1 \} = 3 \# \end{aligned}$$

$$\begin{aligned} \text{cost}(4,4) &= \min \left\{ \begin{array}{l} \text{cost}(4-1, 4) + D(x_4) \\ \text{cost}(4-1, 4-1) + C(x_4, y_4) \\ \text{cost}(4, 4-1) + I(y_4) \end{array} \right\} \\ &= \min \{ 3+1, 2+2, 3+1 \} = 4 \# \end{aligned}$$

Row 5 $\text{cost}(5,1) = \min \begin{cases} \text{cost}(5-1,1) + D(x_5) \\ \text{cost}(5-1,1-1) + C(x_5, y_1) \\ \text{cost}(5,1-1) + I(y_1) \end{cases}$

 $= \min \{ 3+1, 4+0, 5+1 \} = 4 \#$

$\text{cost}(5,2) = \min \begin{cases} \text{cost}(5-1,2) + D(x_5) \\ \text{cost}(5-1,2-1) + C(x_5, y_2) \\ \text{cost}(5,2-1) + I(y_2) \end{cases}$

 $= \min \{ 2+1, 3+0, 4+1 \} = 3 \#$

$\text{cost}(5,3) = \min \begin{cases} \text{cost}(5-1,3) + D(x_5) \\ \text{cost}(5-1,3-1) + C(x_5, y_3) \\ \text{cost}(5,3-1) + I(y_3) \end{cases}$

 $= \min \{ 3+1, 2+0, 3+1 \} = 2 \#$

$\text{cost}(5,4) = \min \begin{cases} \text{cost}(5-1,4) + D(x_5) \\ \text{cost}(5-1,4-1) + C(x_5, y_4) \\ \text{cost}(5,4-1) + I(y_4) \end{cases}$

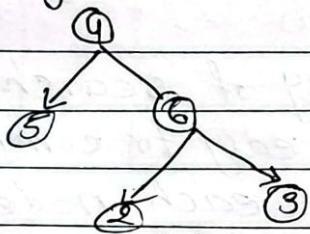
 $= \min \{ 4+1, 3+0, 2+1 \} = 3 \#$

$\text{minimum cost} = 1+1+0+0+0 \cancel{+1} = 3$
 optional cost(3) \neq

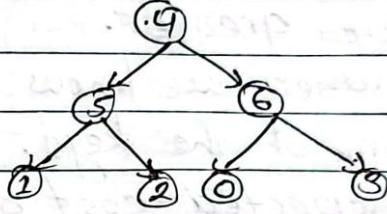
⇒ Optimal Binary Search Tree - (OBST) :-

- 1) Incomplete Binary Tree
- 2) Complete Binary Tree
- 3) Left skewed
- 4) Right skewed.

⇒ Binary Tree (BT) :-

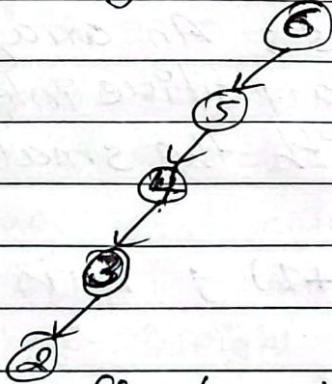


Eg: Incomplete Binary Tree

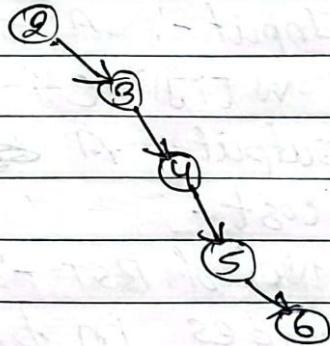


Eg: Complete Binary Tree.

⇒ Binary Search Tree :-



Eg: Left skewed BST



Eg: Right skewed BST.

Binary Tree

- A tree data structure in which each node has at most two children without any ordering restrictions.

- Ordering of node not required .

- Efficiency : $O(n^2)$

Binary Search Tree

- A tree data structure in which each node follows a specific ordering rule, left subtree contains only nodes with keys less than parent nodes & right subtree contains only nodes with keys greater than the parent node.

- Ordering of nodes are sorted.

- Efficiency : $O(n \log n)$.

Definition:-

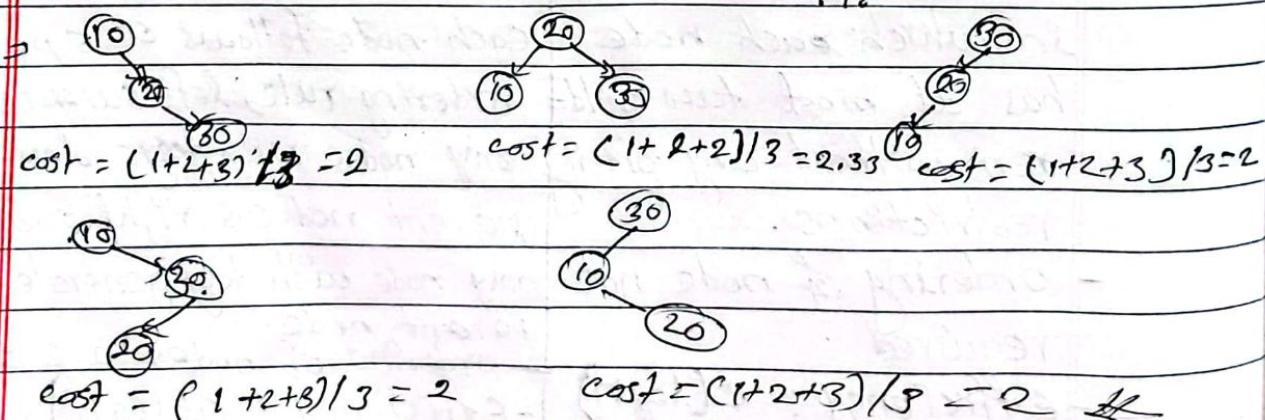
- Binary search Tree (BST) is a tree where the key values are stored in the internal nodes. The external nodes are null node. The keys are stored lexicographically, i.e. for each internal node all the keys in the left sub-tree are less than the keys in the node, and all the keys in the right sub-tree are greater.
- When we know the frequency of searching each one of the keys, it is quite easy to compute the expected cost of accessing each node in the tree. An optimal binary search tree is a BST, which has minimal expected cost of locating each node.
- Input :- A set of n integers; An array W where $W[i] \cdot (1 \leq i \leq n)$ stores a positive integer weight
- Output :- A ~~BST~~ BST on S with the smallest average cost.
- No. of BST :- $= \frac{2^n C_n}{(n+1)}$; n is no of nodes in tree.

eg $n=4 \Rightarrow \frac{8!}{4!5!} = 14 \#$

Example :- Given node: 10, 20, 30

sof The possible BST are:

for $n=3 \Rightarrow$ no. of BST = $\frac{6!}{3!3!} = 5 \#$



Example :-	x :	10	20	30	40
	freq ⁿ :	4	2	6	3

Sof^{to} solving with dynamic programming :

i ↴	j →	0	1	2	3	4
0	0	0	4 ⁽⁰⁾	8 ⁽¹⁾	20 ⁽²⁾	26 ⁽³⁾
1		0	2 ⁽¹⁾	10 ⁽²⁾	16 ⁽³⁾	
2			0	6 ⁽⁰⁾	12 ⁽¹⁾	
3				0	3 ⁽⁰⁾	
4					0	

case I :- $j-i=0$

$$0-0=0 \Rightarrow c[0,0]=0$$

$$1-1=0 \Rightarrow c[1,1]=0$$

$$2-2=0 \Rightarrow c[2,2]=0$$

$$3-3=0 \Rightarrow c[3,3]=0$$

$$4-4=0 \Rightarrow c[4,4]=0$$

now, cost calculate ;

$$c[i,j] = \min \{ c[i,k-1] + c[k,j] \} + w[i,j]$$

where weight $w[i,j] = \sum_{i=1}^n f(i)$.

then,

case III $j-i=2$

$$2-0=2 \Rightarrow c[0,2] = \min_{k=1,2} \left\{ c[0,0] + c[1,k] \right\} + w[0,2]$$

$$= \min \left\{ 0+2, 4+2 \right\} + 2 = 2+4+2 = 8^{(0)}$$

$$3-1=2 \Rightarrow c[1,3] = \min_{k=2,3} \left\{ c[1,k] + c[2,3] \right\} + w[1,3]$$

$$= \min \left\{ 2+6, 2+6 \right\} + 6 = 2+2+6 = 10^{(3)}$$

$$= \min \left\{ 2+6, 2+6 \right\} + 6 = 2+2+6 = 10^{(3)}$$

$$4-2=2 \Rightarrow c[2,4] = \min_{k=3,4} \left\{ \begin{array}{l} c[2,2] + c[3,4], \\ c[2,3] + c[4,4] \end{array} \right\} + w[2,4]$$

$$= \min_{k=0,1} \left\{ \begin{array}{l} 0+3, \\ 6+0 \end{array} \right\} + 6+3 = 3+6+3 = 12^{(3)} \#$$

Case III: $j-i = 3$

$$8-0=3 \Rightarrow c[0,3] = \min_{k=1,2,3} \left\{ \begin{array}{l} c[0,1] + c[2,3], \\ c[0,2] + c[3,3] \end{array} \right\} + w[0,3]$$

$$= \min \left\{ \begin{array}{l} 0+10, \\ 4+6, \\ 8+0 \end{array} \right\} + 4+2+6 = 8+4+2+6 = 20^{(3)} \#$$

$$4-1=3 \Rightarrow c[1,4] = \min_{k=2,3,4} \left\{ \begin{array}{l} c[1,1] + c[2,4], \\ c[1,2] + c[3,4], \\ c[1,3] + c[4,4] \end{array} \right\} + w[1,4]$$

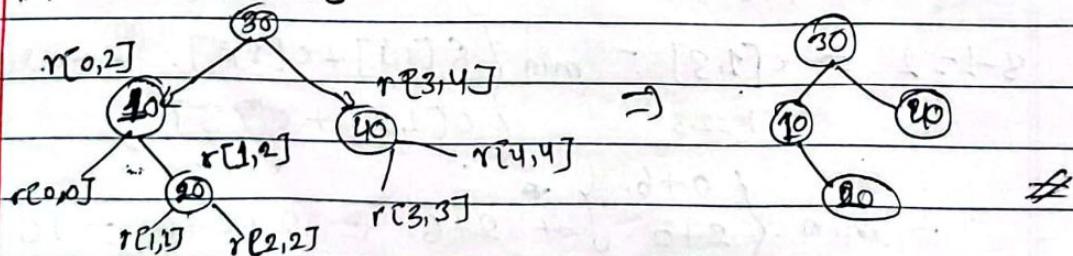
$$= \min \left\{ \begin{array}{l} 0+12, \\ 2+3, \\ 10+0 \end{array} \right\} + 2+6+3 = 5+2+6+3 = 16^{(3)} \#$$

Case-IV: $j-i = 4$

$$4-0=4 \Rightarrow c[0,4] = \min_{k=1,2,3,4} \left\{ \begin{array}{l} c[0,0] + c[3,4], \\ c[0,1] + c[2,4], \\ c[0,2] + c[3,4], \\ c[0,3] + c[4,4] \end{array} \right\} + w[0,4]$$

$$= \min \left\{ \begin{array}{l} 0+16, \\ 4+12, \\ 8+3, \\ 20+0 \end{array} \right\} + 4+2+6+3 = 11+4+2+6+3 = 26^{(3)} \#$$

$$\therefore r[0,4] = 3$$



Analysis :- Algorithm requires $O(n^3)$ time, since there are three (3) level of nested for loop • each of these loops take an at most n value #

Example :- Optimal Binary Search Tree (OBST) :-

x_i	-	10	20	30	40
f_i	-	4	2	6	3
p_j	-	3	3	1	1
q_j	2	3	1	1	1

Soln Now we construct value of cost, weight and root and fill the following table:

$w_{0,0} = 2$	$w_{1,1} = 3$	$w_{2,2} = 1$	$w_{3,3} = 1$	$w_{4,4} = 1$
$c_{0,0} = 0$	$c_{1,1} = 0$	$c_{2,2} = 0$	$c_{3,3} = 0$	$c_{4,4} = 0$
$r_{0,0} = 0$	$r_{1,1} = 0$	$r_{2,2} = 0$	$r_{3,3} = 0$	$r_{4,4} = 0$
$w_{0,1} = 8$	$w_{1,2} = 7$	$w_{2,3} = 3$	$w_{3,4} = 3$	
$c_{0,1} = 8$	$c_{1,2} = 7$	$c_{2,3} = 3$	$c_{3,4} = 3$	
$r_{0,1} = 1$	$r_{1,2} = 2$	$r_{2,3} = 3$	$r_{3,4} = 4$	
$w_{0,2} = 12$	$w_{1,3} = 9$	$w_{2,4} = 5$		
$c_{0,2} = 19$	$c_{1,3} = 12$	$c_{2,4} = 8$		
$r_{0,2} = 1$	$r_{1,3} = 2$	$r_{2,4} = 3$		
$w_{0,3} = 14$	$w_{1,4} = 11$			
$c_{0,3} = 25$	$c_{1,4} = 19$			
$r_{0,3} = 2$	$r_{1,4} = 2$			
$w_{0,4} = 16$				
$c_{0,4} = 32$				
$r_{0,4} = 2$				

$$c[i,j] = \min_{1 \leq k \leq j} \{ c[i,k-1] + c[k,j] \} + w[i,j]$$

and weight $w[i,j] = w[i,j-1] + p_j + q_j$

~~$w[0,0], w[1,1], w[2,2], w[3,3], w[4,4]$ are initial values of $q_j = 2, 3, 1, 1, 1$, then root all zero and initial cost is also all zero.~~

$j-i=0 \Rightarrow c[0,0], c[1,1], c[2,2], c[3,3], c[4,4] = 0$

$j-i=1 \Rightarrow c[0,1], c[1,2], c[2,3], c[3,4]$

$j-i=2 \Rightarrow c[0,2], c[1,3], c[2,4]$

$j-i=3 \Rightarrow c[0,3], c[1,4]$

$j-i=4 \Rightarrow c[0,4]$.

Case-I $j-i=0$

$w[0,0] = q_0 + \dots = 2 \quad c[0,0] = 0 \quad r[0,0] = 0$

$w[1,1] = q_1 + \dots = 3 \quad c[1,1] = 0 \quad r[1,1] = 0$

$w[2,2] = q_2 + \dots = 4 \quad c[2,2] = 0 \quad r[2,2] = 0$

$w[3,3] = q_3 + \dots = 1 \quad c[3,3] = 0 \quad r[3,3] = 0$

$w[4,4] = q_4 + \dots = 1 \quad c[4,4] = 0 \quad r[4,4] = 0$

Case-II $j-i=1$

$$w[0,1] = w[0,0] + p_1 + q_1 \Rightarrow 2 + 3 + 3 = 8$$

$$w[1,2] = w[1,1] + p_2 + q_2 \Rightarrow 3 + 3 + 1 = 7$$

$$w[2,3] = w[2,2] + p_3 + q_3 \Rightarrow 1 + 1 + 1 = 3$$

$$w[3,4] = w[3,3] + p_4 + q_4 \Rightarrow 1 + 1 + 1 = 3 \text{ and}$$

$$c[0,1] = \min_{k=1}^4 \{ c[0,0] + c[1,k] \} + w[0,1] = \min\{0+0\} + 8 = 8$$

$$c[1,2] = \min_{k=2}^3 \{ c[1,1] + c[2,k] \} + w[1,2] \Rightarrow \min\{0+0\} + 7 = 7 \quad (1)$$

$$c[2,3] = \min_{k=3}^3 \{ c[2,2] + c[3,k] \} + w[2,3] \Rightarrow \min\{0+0\} + 3 = 3 \quad (2)$$

$$c[3,4] = \min_{k=4}^4 \{ c[3,3] + c[4,k] \} + w[3,4] \Rightarrow \min\{0+0\} + 3 = 3 \quad (3)$$

Case-II $j-i=2$

$$w[0,2] = w[0,1] + p_2 + q_2 \Rightarrow 8 + 3 + 1 = 12$$

$$w[1,3] = w[1,2] + p_3 + q_3 \Rightarrow 7 + 1 + 1 = 9$$

$$w[2,4] = w[2,3] + p_4 + q_4 \Rightarrow 3 + 1 + 1 = 5$$

$$c[0,2] = \min_{k=1,2} \left\{ \begin{array}{l} c[0,0] + c[1,k], \\ c[0,1] + c[2,k] \end{array} \right\} + w[0,2]$$

$$= \min \left\{ \begin{array}{l} 0+7 \\ 8+0 \end{array} \right\} + 12 = 7+12 = 19 \text{ and}$$

$$\cdot r_{0,2} = 1$$

$$c[1,3] = \min_{k=2,3} \left\{ c[1,1] + c[2,3], c[1,2] + c[3,3] \right\} + w[1,3]$$

$$= \min \left\{ \begin{array}{l} 0+3 \\ 7+0 \end{array} \right\} + 9 \Rightarrow 3+9 = 12 \quad \text{and } r_{1,3} = 2$$

$$c[2,4] = \min_{k=3,4} \left\{ c[2,2] + c[3,4], c[2,3] + c[4,4] \right\} + w[2,4]$$

$$= \min \left\{ \begin{array}{l} 0+3 \\ 3+0 \end{array} \right\} + 5 \Rightarrow 3+5 = 8 \quad \text{and } r_{2,4} = 3 \approx 4$$

Case IV $j-i=3$

$$w[0,3] = w[0,2] + p_3 + q_3 = 12 + 1 + 1 = 14$$

$$w[1,4] = w[1,3] + p_4 + q_4 = 9 + 1 + 1 = 11$$

$$c[0,3] = \min_{k=1,2,3} \left\{ c[0,0] + c[1,3], c[0,1] + c[2,3], c[0,2] + c[3,3] \right\} + w[0,3]$$

$$= \min \left\{ \begin{array}{l} 0+12 \\ 8+3, \\ 12+0 \end{array} \right\} + 14 = 11 + 14 = 25 \quad \text{and } r_{0,3} = 2,$$

$$c[1,4] = \min_{k=2,3,4} \left\{ c[1,1] + c[2,4], c[1,2] + c[3,4], c[1,3] + c[4,4] \right\} + w[1,4]$$

$$= \min \left\{ \begin{array}{l} 0+8, \\ 7+3, \\ 12+0 \end{array} \right\} + 11 = 8+11 = 19 \quad \text{and } r_{1,4} = 2$$

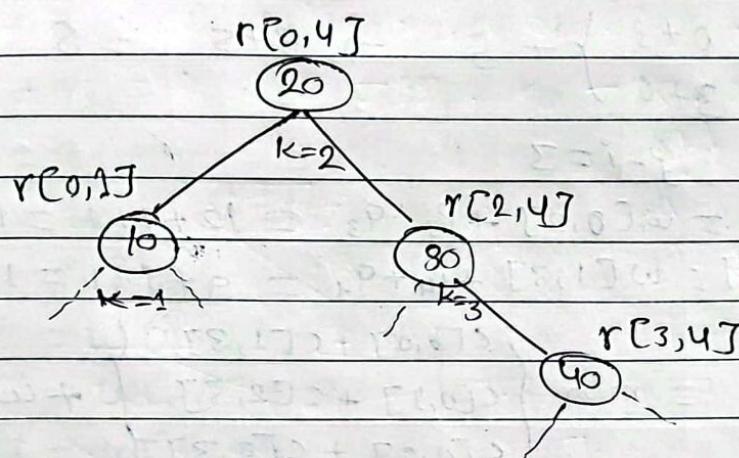
Case V $j-i=4$

$$w[0,4] = w[0,3] + p_4 + q_4 \Rightarrow 14 + 1 + 1 = 16$$

$$c[0,4] = \min \left\{ c[0,0] + c[1,4], c[0,1] + c[2,4], c[0,2] + c[3,4], c[0,3] + c[4,4] \right\} + w[0,4]$$

$$\min \left\{ \begin{array}{l} 0+19, \\ 8+8, \\ 19+3, \\ 25+0 \end{array} \right\} + 16 = 16 + 16 = 32 \text{ & } r_{0,4} = 2$$

\Rightarrow minimum cost = $32/16 = 2$ &
and OBST root is 2 or $r_{0,4} = 2$ then,



⇒ Backtracking :-

- Backtracking is a general algorithmic technique that involves exploring all possible solutions to a problem by incrementally building solution and backtracking when we determine that it cannot be complete further.
- It is used for situations, where we need to explore a large, but finite, search space to find a solution.
- It is particularly useful when the search space is too large to explore ~~each~~ exhaustively by using heuristics. ~~too large to explore~~ to eliminate paths that can't lead to valid solution.

Forward

- General solution vector one at a time.

- Add other feasible alternatives to the vector until the criteria is met.

Backward

- The criteria is not met & traverse backward to immediate node & change the direction.

- The concept of coming backward is also known as pruning.

Algorithm :-

- 1) sum of subsets
- 2) knapsack problem.

⇒ Sum of Subsets :-

- The sum of subset problem is a combinatorial optimization problem that involves finding all possible subset of a given set of elements whose

sum equal a target value. This problem uses solved by using backtracking method.

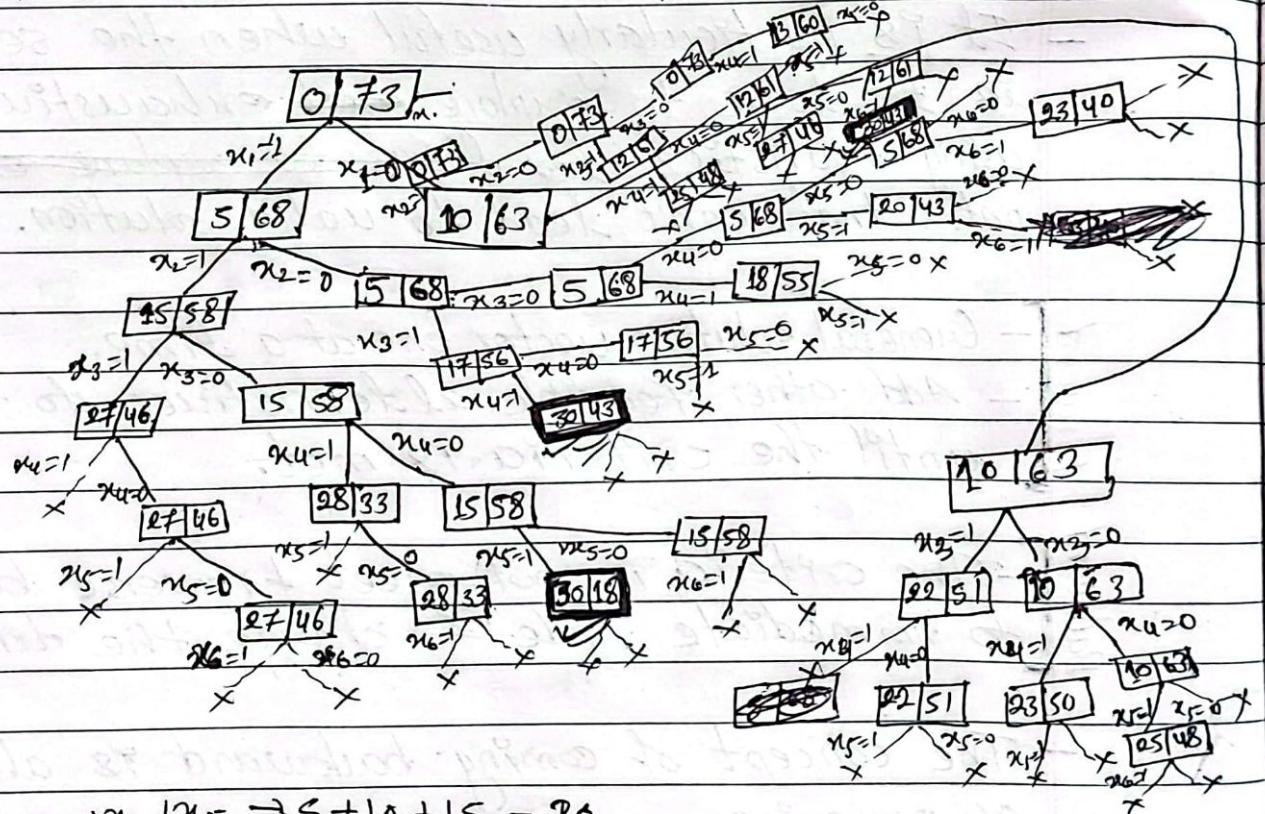
Example :- $W[1:6] = \{5, 10, 12, 13, 15, 18\}$

$n=6$ and sum of subset $m=30$.

8/14

we make total path $2^n = 2^6 = 64$ possible construct tree using backtracking method.

$x_1=5, x_2=10, x_3=12, x_4=13, x_5=15$ & $x_6=18$



$$\Rightarrow x_1 + x_2 + x_5 \rightarrow 5 + 10 + 15 = 30$$

$$x_1 + x_3 + x_4 \rightarrow 5 + 12 + 13 = 30$$

$$x_3 + x_6 \rightarrow 12 + 18 = 30 // \#$$

$$\sum_{i=1}^K w_i x_i + w_{K+1} \leq m$$

$$= \sum_{i=1}^K w_i x_i + \sum_{i=K+1}^n w_i > m$$

→ knapsack Algorithm -1

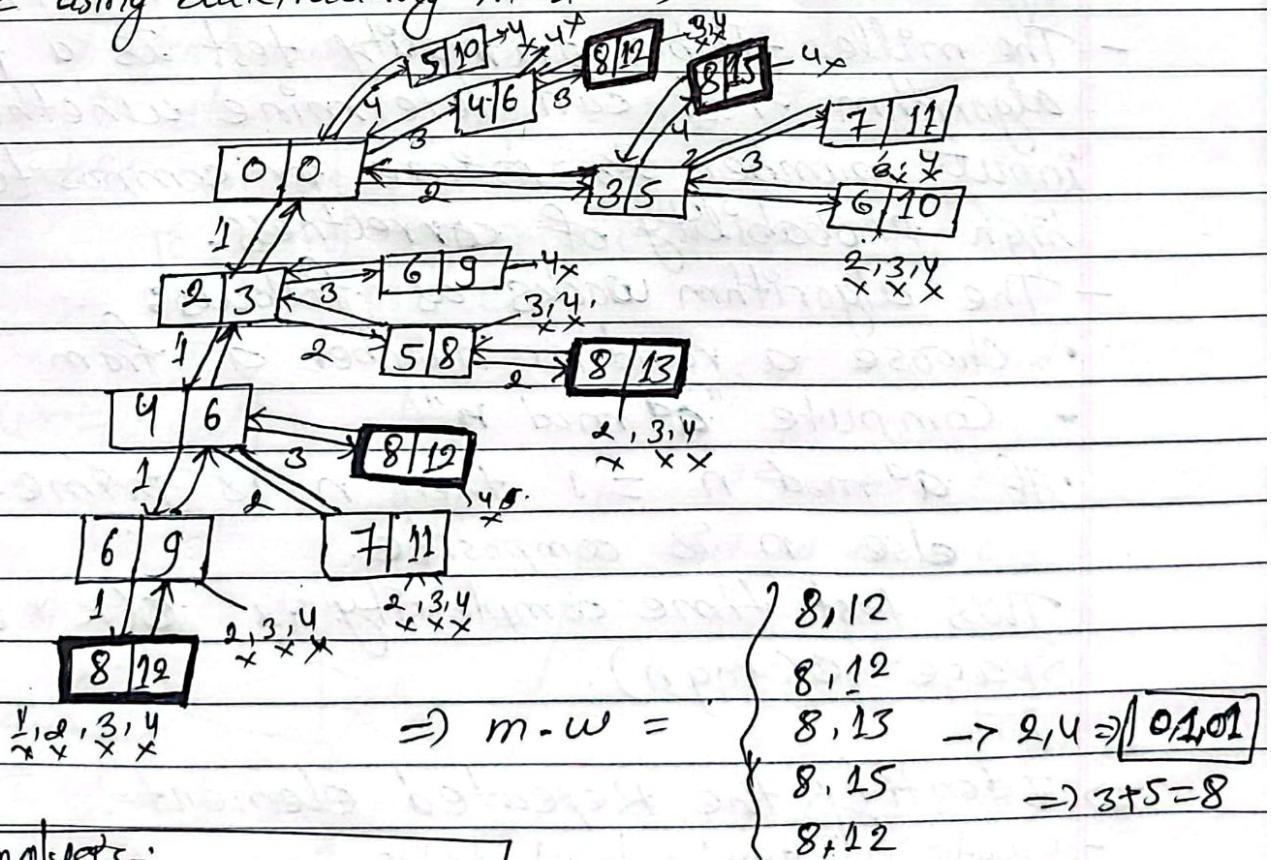
- The knapsack problem is a type of optimization problem where a set of items with different weights and values need to be selected to fill a knapsack of limited capacity.
- The objective is to maximize the total value of the selected items while not exceeding the capacity of the knapsack.
- This knapsack problem using technique is backtracking.

Example -1:

P	1	2	3	4	
w	2	3	4	5	kg
P	3	5	6	10	

max weight $m = 8$ kg.

~~soln~~ using backtracking in tree;



Analysis:-

$O(2^n)$ worst case

$\therefore P = 5+10 = 15 \text{ max}$

\Rightarrow Randomized Algorithm -

- 1. Average cost is minimum than other algorithm
 - 2. It might not provide the correct output.
 - To tackle point 2, we should embed some deterministic approach within randomization.
- eg. monte Carlo.
- Las Vegas
 - primality Test.
- ↳ miller-Robin Algorithm.

\Rightarrow Primality Testing -

- To perform primality testing using randomized algorithm is called the miller-Robin primality.
- The miller-Robin primality test is a probabilistic algorithm that can determine whether an input number is prime or composite with a high probability of correctness.
- The algorithm works as follows:
 - choose a random number a from n .
 - Compute " $a^d \bmod n$ "
 - If $a^d \bmod n = 1$ then n is prime.
else n is composite.

This test time complexity is $\Theta(K * \log^3 n)$ & space $\Theta(\log n)$.

\Rightarrow Identify the Repeated element -

- create an empty hash table.
- for each element in the empty array,

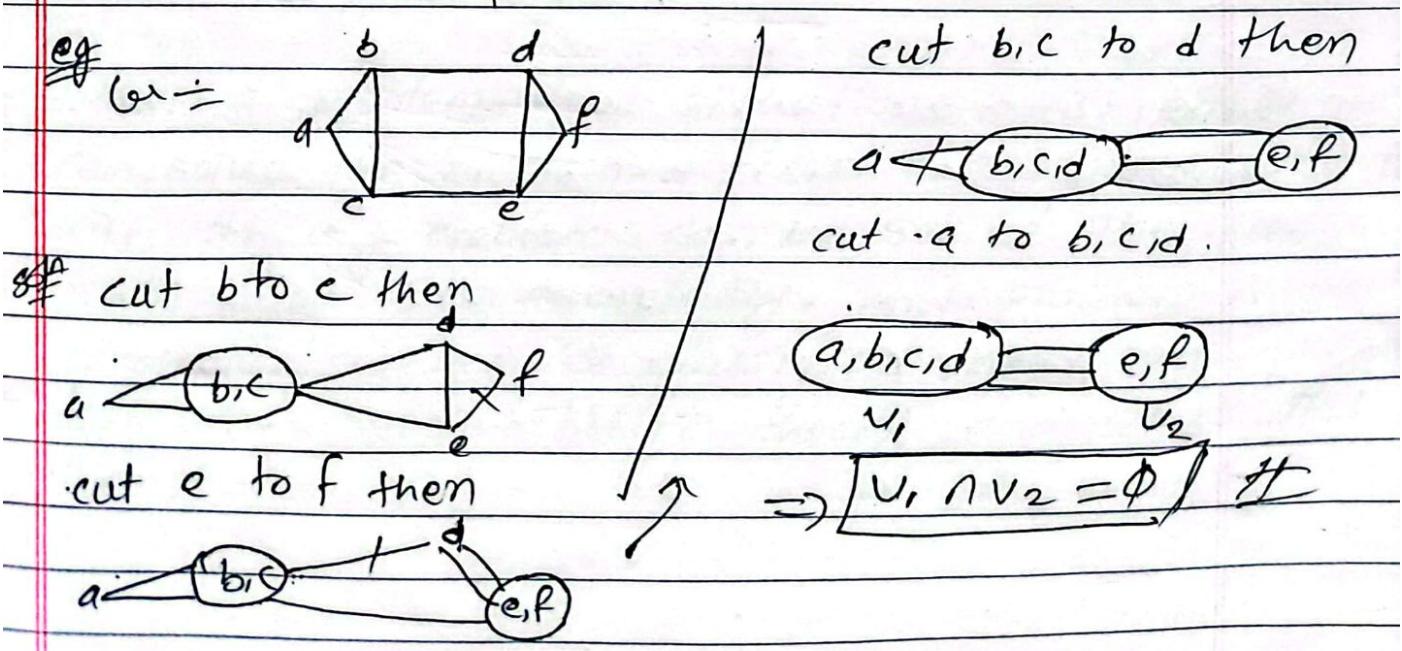
- check element in hash table return element if repeated else element inserted in hash table and not repeated.
- If no repeated element found after processing all element in the array, return null.

Example:

$n = \{1, 2, 3, 5, 1, 3, 7\}$	1	---
1 2 3 5 1 3 7	2	---
k ← key	3	---
	4	.. -
	5	.. -
	6	.. -
	7	.. -

Hash Table.

- Karger's Algorithm:-
- Randomized Algorithm
- provide graph $G_1 = (V, E)$
- Create sub-graph with minimum cut of edge.
- It is also known as min-cut problem.
- It is also called monte carlo algorithm.
- used in computer network.
 - To control congestion control
 - To set protocol.



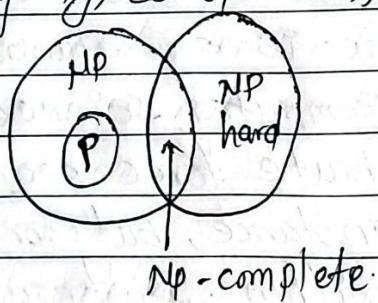
Unit-2 :- Computational Complexity Theory :

⇒ Complexity Theory :-

- In a computer science or mathematical problem is computable. If it can be solved in principle by a computing device. It is also known as solvable, decidable.
- There is an extensive classification of computable problems into computational complexity classes, according to how much computational - as a function of the size of problem instance is needed to answer that instance.
- It concerned with the resources, such as time and space, needed to solve computational problem.
- There is an abundance of problems from mathematics & computer science that are trivially solvable by brute force search of an exponential number of instances, but for which currently no efficient algorithm is known.
- Complexity theory is the appropriate setting for the study of such problems.
- Given a problem, two situations exist, either we can solve the problem or there exists no solution at all. If any problem can be solved the question rises above its complexity. Any problem in computer science is classified ~~and~~ mainly in terms of computability theory and complexity theory.

⇒ Complexity classes -:

- In a computer science there exist some problems, whose solutions are not yet found, the problems are divided into classes known as complexity classes.
- In a complexity theory, a complexity class is a set of problems with related complexity. These classes help scientists to group problems based on how much time and space they require to solve problems and verify the solution.
- Time complexity requires how much number of steps to ~~take~~ completion a problem. and space complexity requires how much memory are useful in organizing similar type of problem.
- There are following types of complexity classes:
 - P class
 - NP class
 - NP-hard
 - NP-complete.



⇒ P-class -:

- The P contains decision problems (problem with Yes or No answer) that are solved by deterministic Turing machine in polynomial time.
- It is a type of problem an algorithm ~~can~~ would take a polynomial amount of time to solve the problem (easy).
- The Big-O notation of these problems is always a polynomial (i.e. $O(1)$, $O(n)$, $O(n^2)$) in general. Then $T(n) = O(c * n^k)$ for all P problem, where $c > 0$ & $k > 0$ and both c & k are constant on n input size.
- ∴ $O(n^k) \neq$

- This class contains many natural problems:
- calculating the greatest common divisor.
- finding a maximum matching.
- Decision versions of linear programming.
- problems are easy & quick to solve in polynomial time.

\Rightarrow NP-class :-

- NP-class consists of those problems that are verifiable in polynomial time.
 - NP is the class of decision problems for which it is easy to check the correctness of a claimed answer, with the aid of a little extra information.
 - Hence, we aren't asking for a way to find a solution, but only to verify that an alleged solution is really correct.
 - In this class every problem can be solved in exponential time using exhaustive search.
 - The NP in NP class stands for non-deterministic polynomial time. It is the collection of decision problems that can be solved by a non-deterministic machine in polynomial time.
 - The solutions of the NP class are hard to find since they are being solved by a non-deterministic machine but the solutions are easy to verify.
 - problems of NP can be verified by a Turing machine in polynomial time.
- eg.
- Boolean satisfiability problem (SAT)
 - Hamiltonian path problem
 - Graph coloring.
 - problems are quick to verify but slow to solve.

⇒ NP-hard :-

- An NP-hard problem is at least as hard as the hardest problem in NP and it is a class of problems such that every problem in NP reduces to NP-hard.
- All NP-hard problems are not in NP.
- It takes a long time to check them. This means if a solution for an NP-hard problem is given then it takes a long time to check whether it is right or not.
- A problem A is in NP-hard if, for every problem L in NP, there exists a polynomial time reduction from L to A.
- NP-hard problems are slow to verify, slow to solve and can be reduced to any other NP problem.
- All NP-hard problems are not in NP & it takes a long time to check them.

- Halting problem
- Qualified Boolean formulas
- No Hamiltonian cycle.

⇒ NP-complete :-

- A problem is NP-complete, if it is both NP and NP-hard. NP-complete problems are the hardest problems in NP.
- NP-complete problems are special as any problem in NP-~~class~~ class can be transformed or reduced into NP-complete problems in polynomial time.
- If one could solve an NP-complete problem in polynomial time, then one could also solve any NP-problem in polynomial time.

- Decision version of 1/0 knapsack
- Hamiltonian cycle
- Satisfiability
- Vertex cover.
- NP-complete problems are also quick to verify slow to solve and can be reduced to any other NP-complete problem.
- A problem that is NP and NP-hard is NP-complete.

⇒ Decision Problems:-

- A decision problem is the problem of determining an answer to a class of yes/no questions about some objects of interest (graph, DFA's, CFG's, TMs; Integers, Boolean formulas, etc.) (0/1, T/F or accept & reject).

⇒ Language Recognition Problems:-

- Observe that a decision problem can also be thought of as a language recognition problem. we could define a language L .

$L = \{(G_i, k) : G_i \text{ has a MST of weight at most } k\}$

- This set consists of pairs, the elements is graph (e.g. the adjacency matrix encoded as a string) followed by an integer k encoded as a binary number.
- any thing broken into somehow string of bits.
- when presented with an input string (G_i, k) , the algorithm would answer "Yes" if $(G_i, k) \in L$. implying that G_i has a spanning tree of weight at most k , and "No" otherwise.

2.2. Problem Reduction :-

Reduction :-

- A reduction is a transformation of one problem into another.
- A reduction of decision problem \mathcal{Q}_1 to decision problem \mathcal{Q}_2 is a mapping of every instance q_1 of problem \mathcal{Q}_1 to an instance q_2 of problem \mathcal{Q}_2 such that q_1 is 'yes' if and only if q_2 is 'yes'.
- If \mathcal{Q}_1 reduces to \mathcal{Q}_2 in polynomial time and \mathcal{Q}_2 reduces to \mathcal{Q}_3 in polynomial time, then \mathcal{Q}_1 reduces to \mathcal{Q}_3 in polynomial time.

e.g. An instance of the Hamiltonian problem is a graph G_i with vertices v_1 and v_n .

- construct a weighted graph G'_i by adding weight 1 to all edges of G_i .
- Travelling Salesman problem.

Polynomial Time Reduction :-

- A polynomial time reduction is a mapping reducing that can be computed in polynomial time.
- Suppose that A and B are two language.
- function f is a polynomial time reduction from A to B if both of the following are true:

 - a) f is computable in polynomial time.
 - b) for every x , $x \in A \Leftrightarrow f(x) \in B$.

- say that $A \leq_p B$ if there exists a polynomial time reduction from A to B .

e.g. we saw the trivial reduction $f(n) = n+1$ from the set of even integers to the set of odd integers. that is a polynomial time reduction because.

- There is an algorithm that adds 1 to n digit number in time $O(n)$.
- For every integer x , x is even $\Leftrightarrow x+1$ is odd.

\Rightarrow Cook's Theorem -

- Cook's Theorem states that the Boolean satisfiability problem is NP-complete. That is, any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of determining whether a Boolean formula is satisfiable.

Corollary: SAT $\in P$ if and only if $P = NP$.

Proof: There are two parts to proving that the Boolean satisfiability problem (SAT) is NP-complete.

- One is to show that SAT is an NP problem.
- SAT is NP because any assignment of Boolean values to Boolean variables that is claimed to satisfy the given expression can be verified in polynomial time by a deterministic Turing machine.
- The other is to show that every NP problem can be reduced to an instance of a SAT problem by a polynomial-time many-one reduction.
- Suppose that a given problem in NP can be solved by the nondeterministic Turing machine $M = (Q, \Sigma, S, F, \delta)$ where.

$Q \rightarrow$ set of states

$\Sigma \rightarrow$ ~~set~~ Alphabet

$S \subseteq Q \rightarrow$ set of ~~initial~~ accepting states.

$F \subseteq Q \rightarrow$ set of accepting states.

and $\delta \subseteq ((Q \setminus F) \times \Sigma)^* \times (Q \times \Sigma^* \times \{-1, +1\})$ is transition relation. and m accepts or rejects an instance of the problem in time $p(n)$ p-polynomial function with size of instance n .

For each input, I , we specify a Boolean expression expression which is satisfiable if and only if the machine m accept I .

- Boolean expression uses the variables set such that, $q \in Q$, $-p(n) \leq i \leq p(n)$, $\sigma \in \Sigma$, and $0 \leq k \leq p(n)$
- If there is an accepting computation for m on input I , then B is satisfiable by assigning their intended interpretations. On the other hand, if B is satisfiable, then there is an accepting computation for m on input I that follows the steps indicated by the assignments to the variables.
- There are $O(p(n)^2)$ Boolean variables; each encodable in space $O(\log p(n))$. The number of clauses is $O(p(n)^3)$ so the size of B is $O((\log(p(n)))p(n)^3)$. Thus the transformation is certainly a polynomial-time many-one reduction, as required.

$\Rightarrow SAT$:

- SAT is also called Boolean satisfiability problem.
- A propositional logic formula φ is called satisfiable if there is some assignment to its variable that makes it evaluate to true.
- $p \wedge q$ is satisfiable i.e. for $p \Rightarrow 1 \wedge q \Rightarrow 1$ $p \wedge q$ is true.

- $P \neq NP$ is not satisfiable, but no value of p could make boolean function ($P \neq NP$) true, so not satisfiable.
- SAT is NP complete.

\Rightarrow Prove that formula satisfiable is a NP class.

Proof SAT \in NP

- Given an assignment for x_1, x_2, \dots, x_n . You can check it: $\phi(x_1, x_2, \dots, x_n) = 1$ in polynomial-time by evaluating a formula on a given assignment.
Hence SAT is in NP.

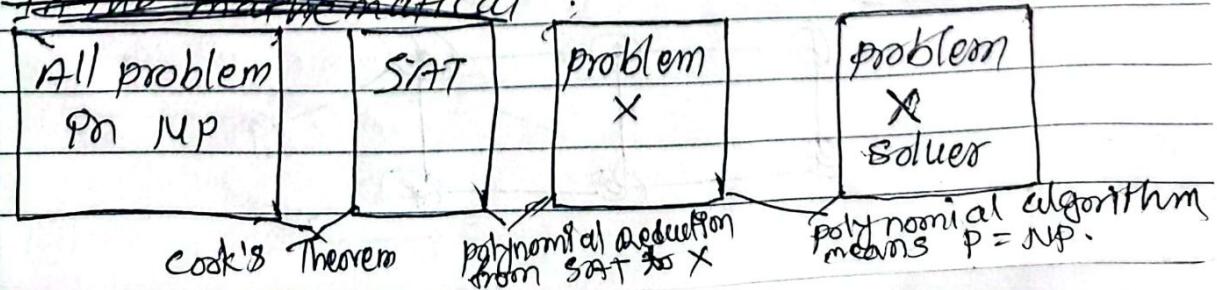
Proof SAT \in NP-hard -

\Rightarrow 3SAT -

- problem: Given a CNF where each clause has 3 variables; decide whether it is satisfiable or not.
- $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$
- A language 3SAT is $\{\psi | \psi$ is satisfiable $\}$ CNF formula ψ .
- 3SAT is NP complete.

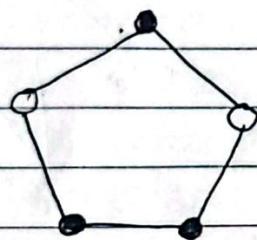
\Rightarrow ~~3SAT~~ Vertex Cover -

- ~~For the mathematical~~:



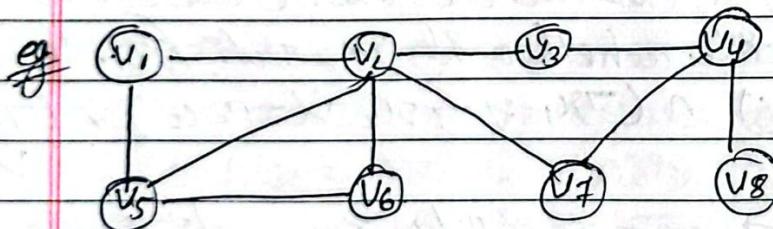
⇒ vertex cover:

- In the mathematical discipline of graph theory, "A vertex cover (sometimes node cover) of a graph is a subset of vertices which "covers" every edge."
- An edge is covered if only if its endpoint is chosen.
- In other words 'a vertex cover for a graph G_1 is a set of vertices incident to every edge in G_1 '
- The vertex cover problem: what is the maximum size vertex cover in G_1 ?
- problem: Given graph $G_1 = (V, E)$, find smallest $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ or both.



This problem can be solved by

- ① Greedy Approach.
- ② Approx optimal solution.

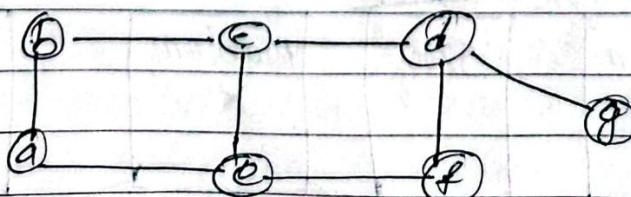


vertex set $\{v_1, v_3, v_5, v_6, v_7, v_8\}$ is a vertex cover.

vertex set $\{v_2, v_4, v_5\}$ is vertex cover.

vertex set $\{v_2, v_3, v_8\}$ is not a vertex cover since edge $\{v_4, v_7\}$ is not covered.

Example:



$$\text{Step } 1 \quad C = \emptyset$$

$$E' = \{(a,b), (b,c), (c,d), (c,e), (e,f), (d,f), (d,g), (d,h)\}$$

Initially let's select (b,c) edge, where,

$$u = b$$

$$v = c$$

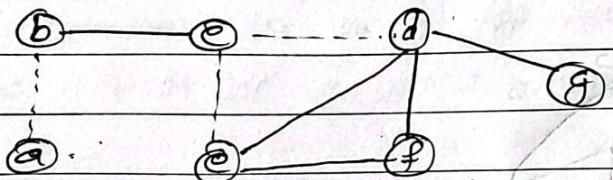
$$C = \emptyset \cup \{b, c\}$$

$$= \{b, c\}$$

Now, remove from E' every edge incident

$$E' = \{(d,e), (e,f), (d,f), (d,g)\}$$

Again, let's select (e,f) vertex.



Again, let's select (e,f) vertex where,

$$u = e$$

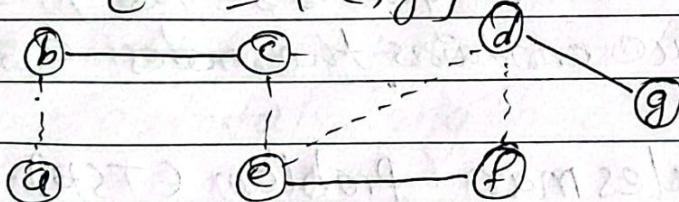
$$v = f$$

$$C = \{b, c\} \cup \{e, f\}$$

$$= \{b, c, e, f\}$$

Now, remove E' every edge incident.

$$E' = \{d, g\}$$



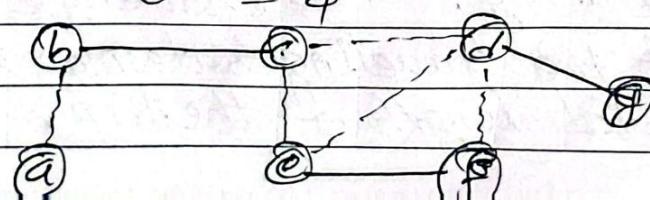
Again let's select (d,g) vertex where,

$$u = d$$

$$v = g$$

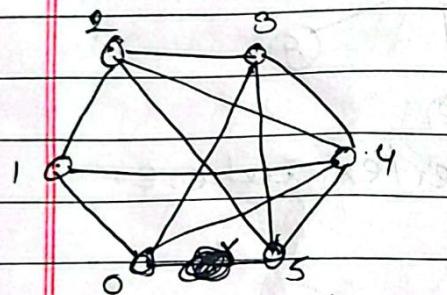
$$C = \{b, c, e, f\} \cup \{d, g\} = \{b, c, d, e, f, g\}$$

$$E' = \emptyset$$



⇒ Maximum clique problem:-

- In a graph G , a subset of vertices fully connected to each other, i.e. a complete subgraph of G is called clique.
- The maximum clique problem: how large is the maximum-size clique in a graph?
- In another words, given a graph of vertices some of which have edges in between them, the maximal clique is the largest subset of vertices in which each point is directly connected to every other vertex in the subset.

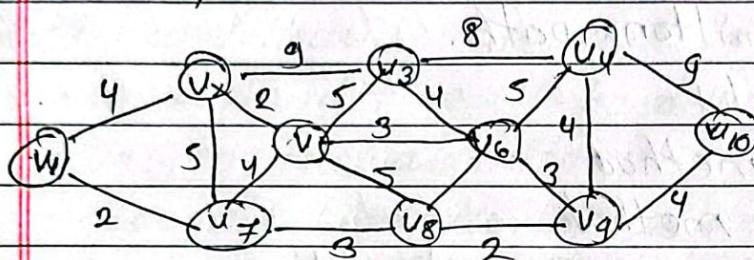


- Given a graph with vertices 0, 1, 2, 3, 4, 5. Here, the maximum clique problem is the largest subset of graph which is complete.
- And the maximum clique size is 4, and the maximum clique contains the nodes 2, 3, 4, 5.

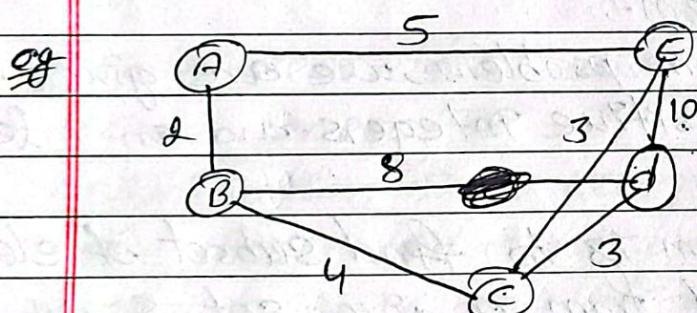
⇒ Travelling Salesman problem (TSP) :-

- The travelling salesman problem consists of a salesman and a set of cities. The salesman has to visit each one of the cities starting from a certain one (e.g. the hometown) and return to the same city. The challenge of the problem is that the travelling salesman wants to minimize the total length of the trip.

- If find shortest possible route.
- Given a weighted graph G_1 , find the minimum cost path from v_1 to v_n which visits each vertex exactly once.
- No known polynomial time algorithm for solving this problem.



Find the minimum cost path from v_1 to v_{10} which visits each vertex exactly once.

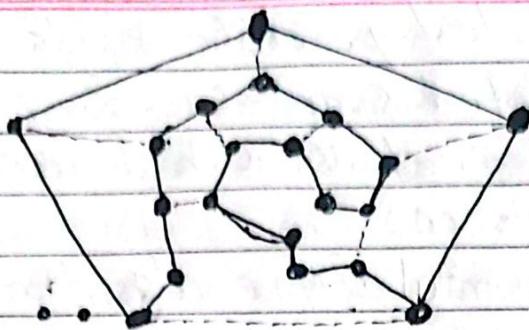


- from path-1 : $\langle A, B, C, D, E, A \rangle$ and path-2 - $\langle A, B, C, E, D, A \rangle$ pass all the vertices but path-1 has total length of 24 and path-2 has a total length of 31

- If solve brute-force approach & Dynamic approach.

\Rightarrow Hamiltonian cycle -

- Hamiltonian cycle or circuit is a graph cycle (i.e. closed loop) through a graph that visits each node exactly once (except for the vertex that is both the start & end, which is visited twice).



Ex: Hamilton path.

- It can be solved by
 - Brute force method
 - Frank Rubin method
 - Dynamic programming algorithm
 - Monte Carlo Algorithm

⇒ Subset Sum problem:-

- In the subset-sum problem, we are given a finite set S of positive integers and an integer target $t \geq 0$.
- Subset sum problem is to find subset of elements that are selected from a given set S whose sum adds up to a given number k .
- Mathematically, subset sum: $f(S, t)$: there exists a subset S' , $S' \subseteq S$ such that $\sum_{s \in S'} s = t$.

Ex $a = [12, 1, 3, 8, 20, 50] \Rightarrow t=44 \rightarrow \text{true}, t=12 \rightarrow \text{false}$

2.3:- NP-hard Code Generation problems:-

- The function of a compiler is to translate programs written in some source language into an equivalent assembly language or machine language program.
- A translation of an expression ~~into~~ into the machine or assembly language of a given machine is opti-

mal if and only if it has a minimum number of instructions.

- A binary operator \odot is commutative in the domain D iff $a \odot b = b \odot a$ for all $a, b \in D$.
- It refers to the task of generating source code that solves an NP-hard problem.
- It has following steps:
 - understand the NP-hard problem.
 - Design the algorithm.
 - choose a programming language.
 - Implement the algorithm.
 - Test and validate.
 - optimize if needed and refine and iterate.

\Rightarrow Code Generation with common subexpression:-

- Code generation with common subexpression elimination is a technique used to optimize the generated code by identifying and eliminating redundant computations.
- The goal is to identify expressions that are computed multiple times and replace them with a single computation, storing the result in a temporary variable.

\Rightarrow Parallel Implementing Parallel Assignment Instructions:-

- It execution of multiple assignment statements simultaneously, taking advantage of parallelism to improve performance.
- the format $(v_1, v_2, \dots, v_n) = (e_1, e_2, \dots, e_n)$ where, v_i 's are distinct variable name & the e_i 's are expressions.
- multiple independent assignments can be execute concurrently.

2.4: Coping with NP-completeness -

⇒ performance ratios for approximation algorithm -

- The performance ratios for approximation algorithm are used to measure the quality of approximation algorithms. An approximation algorithm is designed to solve optimization problems with the goal of finding a near-optimal solution that is close to optimal solution but can be computed more ~~efficiently~~ efficiently.

- It is defined as worst-case ratio between the solution produced by the algorithm and the optimal solution. It quantifies how close the approximation algorithm's solution is to the optimal solution. The smaller the performance ratio, the better the approximation algorithm.

⇒ Approximate Algorithms -

- It is also known as approximation algorithms, are algorithms designed to find near-optimal solutions for optimization problems in a computationally efficient manner.
- It strike the balance between solution quality and computational complexity.
 - performance guarantee
 - Greedy algorithms
 - Heuristic algorithms
 - Randomized algorithm
 - Polynomial time complexity

Unit-3: Online and PRAM Algorithms - I

⇒ Online Algorithms - I

- An online algorithm is one that can process its input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the start.
- An offline algorithm is given the whole problem data from the beginning and is required to output an answer which solves the problem at hand.
~~e.g. selection sort requires that the entire list be given before it can sort it, while insertion sort doesn't.~~
- An online algorithm is one that can process its input piece by piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the start.
- An offline algorithm is given the whole problem data from the beginning and is required to output an answer which solves the problem at hand.
~~e.g. selection sort eg is this one.~~
- Because it doesn't know the whole input, an online algorithm is forced to make decisions that may later turn out not to be optimal, and the study of online algorithm has focused on the quality of decision making that is possible in this setting.
- Competitive analysis formalizes this idea by comparing the relative performance of an online and offline algorithm for the same problem instance.

- An on-line algorithm A is c -competitive if there is a constant b for all sequence s of operations..

$$A(s) \leq c \text{OPT}(s) + b.$$

- where $A(s)$ is the cost of A on the sequence s and $\text{OPT}(s)$ is the optimal off-line cost for the same sequence.
- Competitive ratio is a worst case bound.

\Rightarrow Ski Rental Problem -:

- Assume that you are taking ski lessons.
- It is also known as Buy-Rent problem.
- The training period is unknown.
- Decision has to be made whether to continue rent or buy at a particular instance.
- After each lesson you decide (depending on how much you enjoy it, and what is your bones status) whether to continue to ski or to stop totally.
- You have the choice of either renting skis for 1\$ a time or buying skis for y\$.
- Will you buy or rent?
- If you knew in advance how many times you would ski in your life then the choice of whether to rent or buy is simple. If you will ski more than y times. then buy before you start, otherwise always rent.
- The cost of this algorithm is $\min(f, y)$.
- This type of strategy, with perfect knowledge of the future, is known as an offline strategy.

- In practice, you don't know how many times you will ski. what should you do?
- An online strategy will be a number k such that after renting $k-1$ times you will buy skis (just before your k^{th} visit).
- claim : setting $k = \gamma$ guarantees that you never pay more than twice the cost of the offline strategy.
- Example: Assume $\gamma = 7\$$. Thus, after 6 rents, you buy. Your total payment: $6 + 7 = 13\$$.

\Rightarrow Theorem:

- setting $k = \gamma$ guarantees that you never pay more than twice the cost of the offline strategy.
- proof: When you buy skis in your k^{th} visit, even if you quit right after this time, $t >= \gamma$.
 - your total payment is $k - 1 + \gamma = 2\gamma - 1$.
 - The offline cost is $\min(t, \gamma) = \gamma$.
 - The ratio is $(2\gamma - 1)/\gamma = 2 - (1/\gamma)$.
- we say that this strategy is $[2 - (1/\gamma)]$ - competitive.

\Rightarrow conclusion:

- when balancing small incremental costs against a big one-time cost, you want to delay spending the big cost until you have accumulated roughly the same amount in small costs.

\Rightarrow Disk scheduling Algorithm -

- Access time = Seek Time + latency time.
- Seek Time \rightarrow Head positioning
- Latency Time \rightarrow Disk position.

1) FCFS (First come First Serve)

2) SSTF (Shortest Seek Time First)

3) SCAN (ELEVATOR)

4) C-SCAN

5) LOOK

6) C-LOOK.

← Left Inward

→ Right Outward

Example :- suppose the provided sequence of requests are:

sector	S ₁	S ₁	S ₂	S ₃	S ₁	S ₄
track	15	20	10	70	120	35

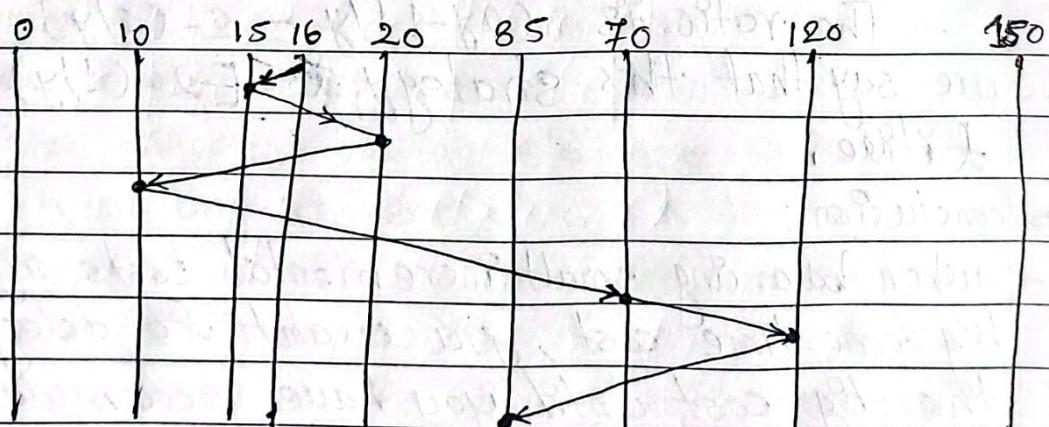
$$\text{latency Time} = 0.2 \text{ ms}$$

$$\text{seek Time} = 0.1 \text{ ms}$$

$$\text{max Track} = 150$$

$$\text{Head position} = 16$$

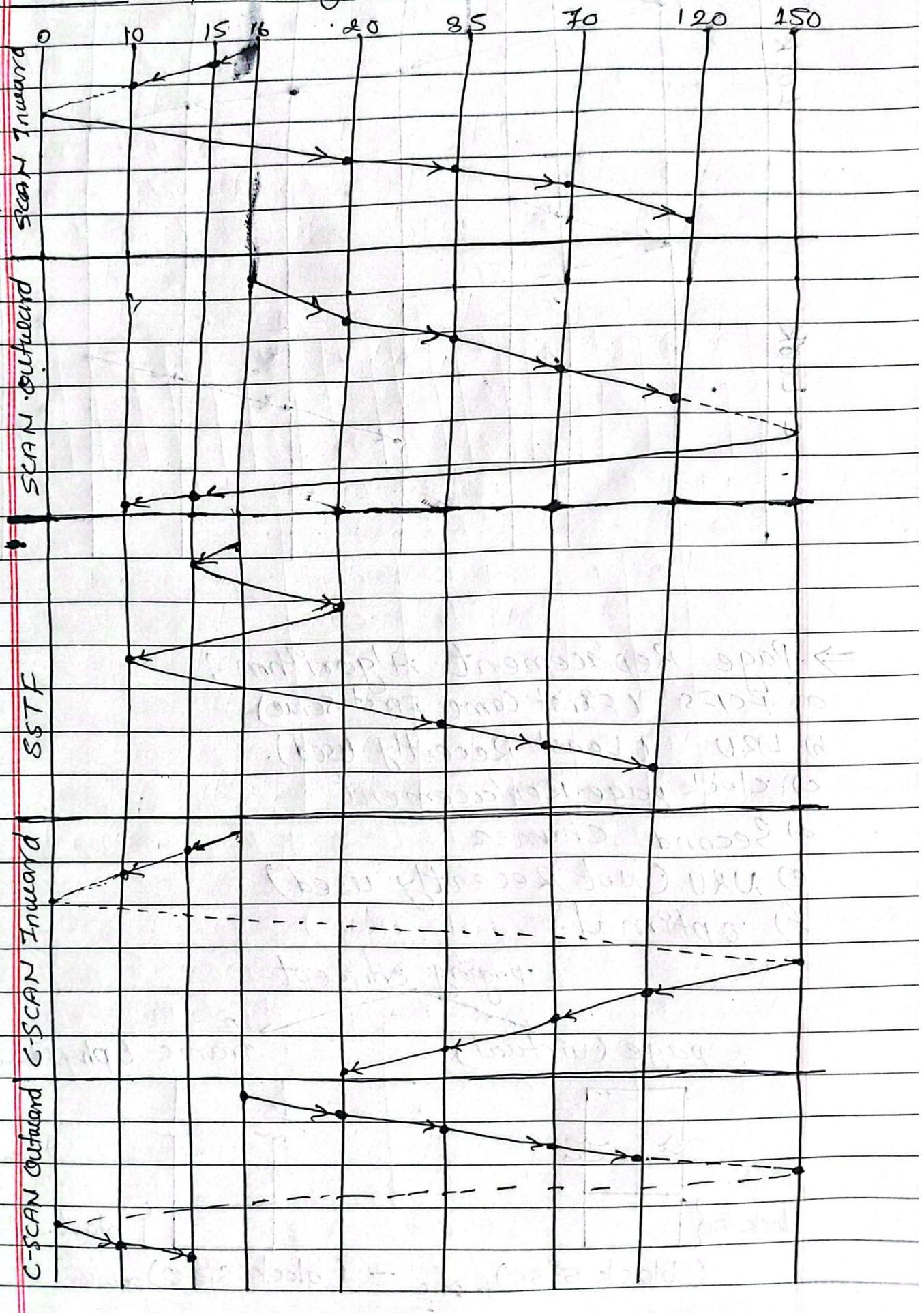
FCFS :-

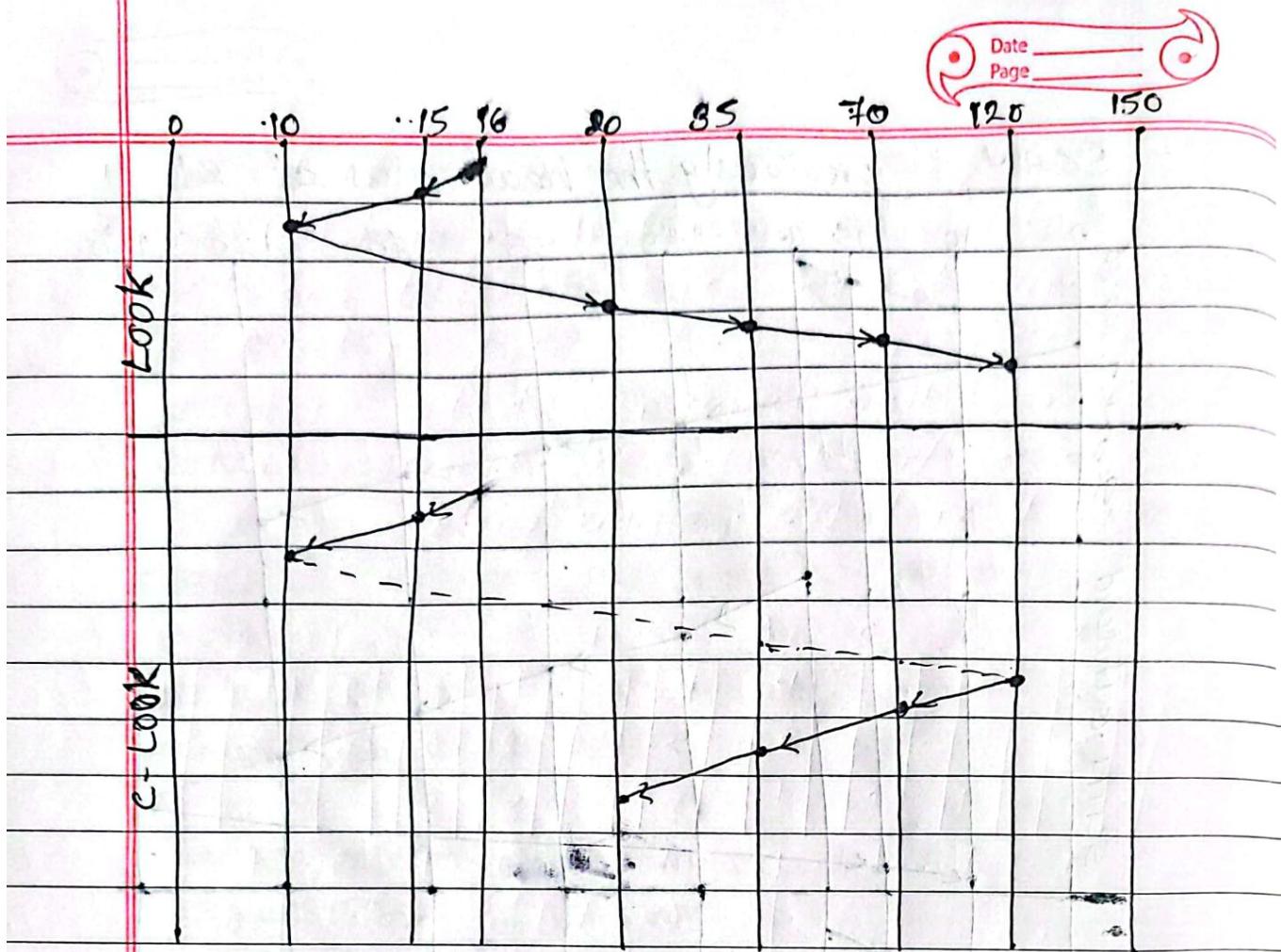


$$\begin{aligned} \text{Access Time} &= ((16-15) \times 0.1) + ((20-15) \times 0.1) + ((20-10) \times 0.1 \\ &+ 0.2) + ((70-10) \times 0.1) + 0.2) + (((120-70) \times 0.1) + 0.4) + \\ &((120-35) \times 0.1) + 0.6 \\ &= 22.5 \end{aligned}$$

~~SCAN~~ → previously the head was at 21.

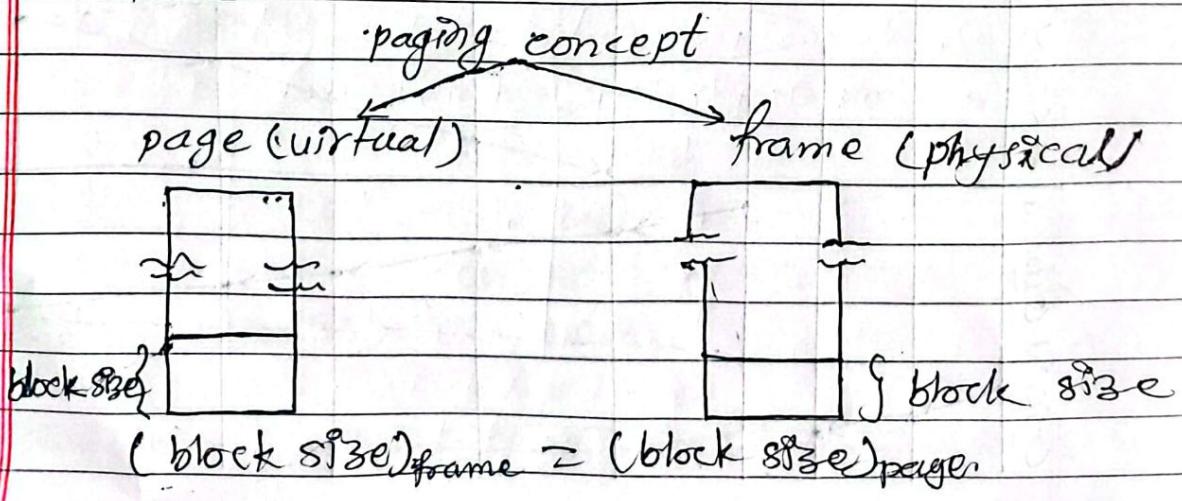
SCAN -: previously the head was at 21.





⇒ Page Replacement Algorithm -:

- a) FCFS (First Come First Serve)
- b) LRU (Least Recently Used).
- c) clock page Replacement
- d) Second chance
- e) NRU (Not Recently used)
- f) optimal.



Example :- provided sequence.

Request — 1, 2, 1, 3, 2, 4, 5, 1, 2, 6, 3, 2

size of page 3

a) FCFS - (First Come First Serve)

1	2	1	3	2	4	5	1	2	6	3	2
1	1	1	1	1	4	4	4	2	2	2	2
2	2	2	2	2	5	5	5	6	6	6	6
3	3	3	3	3	1	1	1	8	3		

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

Total page fault = 9 #

b) LRU (Least Recently Used) :-

1	2	1	3	2	4	5	1	2	6	3	2
1	1	1	1	1	4	4	4	2	2	2	2
2	2	2	2	2	2	2	1	1	1	3	3
3	3	3	3	3	5	5	5	6	6	6	6

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

Total page fault = 9

- Implemented in Linux family because it is free from Balady's Anomaly provided stack data structure is used. (page size increased condition \rightarrow Balady's Anomaly (LFD) \rightarrow Longest Forward Distance).

c) Optimal :- (Replace the page not in demand in the future assumption based on nearest request.)

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

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

Total page fault = 7 //

d) Second chance :-

- If R-bit is 0, that page is candidate for replacement which is carried out w.r.t. FCFS.

If R-bit is 1, second chance is provided & R-bit is set to 0, which will be replaced.

based on competition. If R-bit is already 1 and is referred again, R-bit is 1 again.

1	2	1	3	2	4	5	1	2	6	3	2
1	1	1	1	1	4	4	4	2	2	2	2
2	2	2	2	2	5	5	5	6	6	6	6
3	3	3	3	3	3	1	1	1	3	3	3
1	1	0	1	0	1	1	1	1	1	1	0
page	Reference bit (R)										
1	0	1	0	1	0	0	1	1	1	1	1
2	1	1	0	1	0	0	1	1	1	1	1
3	0	1	0	1	0	0	0	1	1	1	1
4	1	1	0	0	0	0	0	0	1	1	1
5	1	1	0	0	0	0	0	0	0	1	1
6	0	1	0	0	0	0	0	0	0	0	1

$$\therefore \text{Total page fault} = 9 \#$$

e) NRU (Not Recently used) :-

It uses two bit status framework of R & m.

Read (R)	modify (m)	Priority of page replacement
0	0	1
0	1	2
1	0	3
1	1	4

Assumption : All R-bit is 0

m bit of 3, 5, 6 is 1.

Arrival time is after every 2 sec. starting from 0. sec. the R-bit of page will reset to 0 after every 3 sec.

1	2	1	3	2	4	5	1	2	6	3	2
1	1	1	1	1	4	4	4	4	5	5	5
	2	2	2	2	2	2	5	5	5	5	5
			3	3	3	3	3	3	3	3	3
0	0	0	0	0	1	1					

Page	R	M	Timings(s)
1	0	0	0
2	0	0	2
1	0	0	4
3	1	1	6
2	0	0	8
4	0	0	10
5	1	1	12
1	0	0	14
2	0	0	16
6	1	1	18
3	1	1	20
2	0	0	22

→ NRU algorithm removes a page at random from the lowest numbered nonempty class. Implicit in this algorithm is that it is better to remove a modified page that has not been referenced at least one clock tick than a clean page that is in heavy use.

- It is easy to understand, moderately efficient to implement, and gives a performance that is not opti-

⇒ Load Balancing Algorithms:-

- Distribution of processes within servers, so that the stress is controlled. It is also known as stress testing.

Type -:
a) Round Robin

b) weighted Load Balancing.

- A set of m identical machines.

- A sequence of jobs with processing times p_i .

- Each job must be assigned to one of the machines.

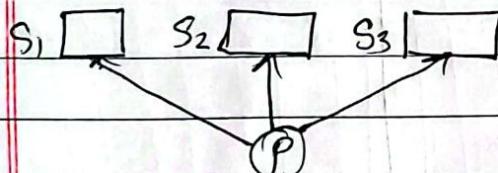
Goal:- schedule the jobs on machines in a way that

- minimize the makespan.

a) Round Robin

- Timing factor is used

- Selection of servers is carried out based on timing.



b) weighted Load Balancing:-

- Every servers is provided with some weights.

- The server with highest weight is chosen first.

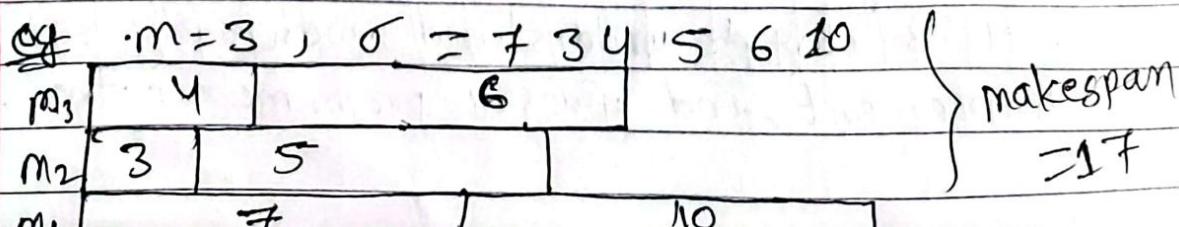
$w=10$ [S1] $w=4$ [S2] $w=5$ [S3]

first OK

- How to minimize the delay of context switch between servers in load balancing algorithm?

⇒ List Ranking:-

A greedy algorithm : always schedule a job on the least loaded machine.



⇒ 3.2 PRAM Algorithms :-

- Let there be 100 numbers to be added.
- o How much time will a person take to add?

$$\begin{array}{ccccccc} & & 1 & & 100 \\ & & \hline & & 99 \end{array}$$

- o How much time will two person take to add?

$$\begin{array}{ccccccc} & 1 & & 50 & . & 51 & 100 \\ & \hline & 49 & \underbrace{& & 1 & } & 49 \end{array}$$

i.e. 50 (provide 1 sec is required to add two no's)

- ~~Ans~~: The addition of processors does not always reduce the run time. therefore for a system to be work optimal the number of processors have to carefully chosen.

- For this eg -

$$\text{Asymptotic speed up} = 99/50 \approx 1.98 \approx 2$$

$$\text{Asymptotic speed up} = \frac{s(n)}{T(n,p)}$$

where,

$s(n)$ = sequential run time

$T(n,p)$ = PRAM run time

• n is number of inputs

• p is number of processors

$$\text{Total work done} : \Theta(s(n)) = p * T(n,p)$$

⇒ Efficiency :- Θ :-

$$\Theta := s(n) / (p * T(n,p))$$

$$= s(n) / \Theta(s(n)).$$

- To be work optimal it requires the efficiency to be $\Theta(1)$. $\Rightarrow 1/$

- ex - optional run time to sort n keys ~~$\Theta(n^2)$~~
 $= \Theta(n \log n)$. Let A be an n -processor parallel algorithm that sorts n keys in $\Theta(\log n)$ time.
 let B be n^2 -processor algorithm that also sorts n keys in $\Theta(\log n)$ time.

find out which algorithm is works optimally?

$$\text{I). (Speed up)}_A = \frac{\Theta(n \log n)}{\Theta(\log n)} = \Theta(n)$$

$$\cdot \quad (\text{Speed up})_B = \frac{\Theta(n \log n)}{\Theta(\log n)} = \Theta(n)$$

$$\text{II). (work done)}_A = n \cdot \Theta(\log n)$$

$$(\text{work done})_B = n^2 \Theta(\log n)$$

$$\text{III). Efficiency } (\text{efficiency})_A = \frac{\Theta(n \log n)}{n \cdot \Theta(\log n)} = 1$$

$$\text{Efficiency } (\text{efficiency})_B = \frac{\Theta(n \log n)}{n^2 \Theta(\log n)} = \frac{1}{n}$$

⇒ Prefix Computation - :

- Let Σ be the domain ~~over~~ in which binary associative operator \oplus is defined.
- Any operator is binary associative if for any of three elements x, y, z from.
- $\Sigma(((x \oplus y) \oplus z) = (x \oplus (y \oplus z)))$ [\oplus can be $+, -, 1, \max, \min, \text{avg}$].

$$\Sigma = (5, 8, -2, \text{f}, -11, 12)$$

$\oplus = \text{minimum}$.

output = 5, 5, -2, -2, -11, -11.

- The prefix computation problem on Σ_i has an input 'n' elements from Σ_i , say $x_1, x_2, x_3, \dots, x_n$.

The problem is to compute the n elements.

$$x_1, x_1 \oplus x_2, \dots, x_1 \oplus x_2 \oplus x_3 \oplus \dots \oplus x_n.$$

The output elements are preferred to be prefixes.

Example:- Let Σ_i be the set of integers and \oplus be the usual addition operation.

$$\text{Input} = 3, -5, 8, 2, 5, 4$$

$$\text{output} = 3, -2, 6, 8, 13, 17$$

Example:- Let $n=8$ and $p=8$

let the input to the prefix computation problem be. $12, 3, 6, 8, 11, 4, 5, 7$, & \oplus be addition.

Step-1

processor P_1 to P_4 computes the prefix sum of $12, 3, 6, 8$ to arrive at $12, 15, 21, 29$.

and,

processor P_5 to P_8 computes the prefix sum of $11, 4, 5, 7$ to arrive at $11, 15, 20, 27$

Step-2

processor P_1 to P_4 sits idle.

processor P_5 to P_8 will update their results by adding 29 to prefix to obtain $40, 44, 49, 56$.

Step-1

12	12, 3	12, 3, 6	12, 3, 6, 8	11	11, 4	11, 4, 5	11, 4, 5, 7
----	-------	----------	-------------	----	-------	----------	-------------

12	12, 15	12, 15, 21	12, 15, 21, 29	11	11, 15	11, 15, 20	11, 15, 20, 27
----	--------	------------	----------------	----	--------	------------	----------------

12, 15, 21, 29

11, 15, 20, 27

Step-2

12, 15, 21, 29

40, 44, 49, 56

\Rightarrow Analysis :-

Step 1 takes $T(n/2)$

Step 2 takes $O(1)$

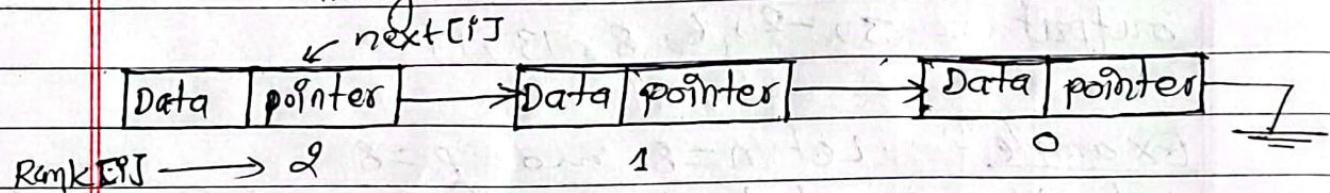
Recurrence relation is,

$$T(n) = T(n/2) + O(1), T(1) \Rightarrow$$

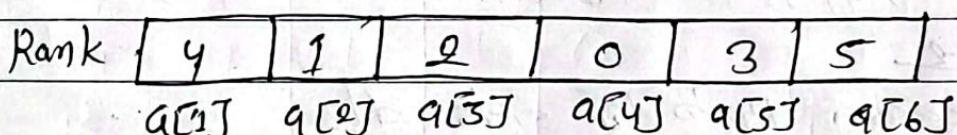
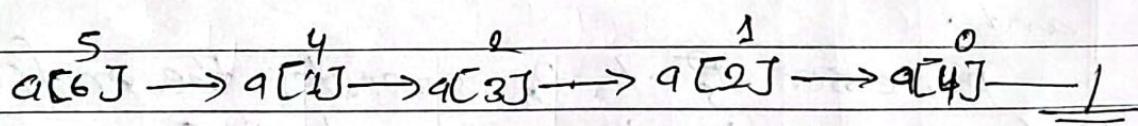
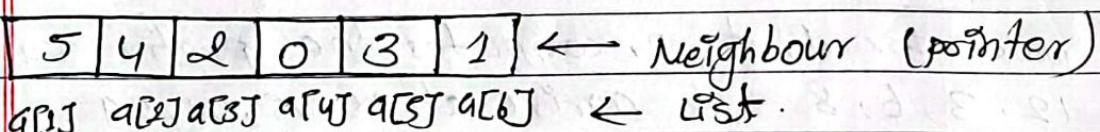
This solves to,

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

\Rightarrow List Ranking :-



pointer jumping.



Algorithm :-

Initialization

for (each processor i)
if ($\text{next}[i] == 0$)

Rank $EiJ = 0$

else

Rank $EiJ = 1$

Finding Rank

while ($\text{next}[i] != \text{null}$)

for (each processor i)
if ($\text{next}[i] == \text{null}$)

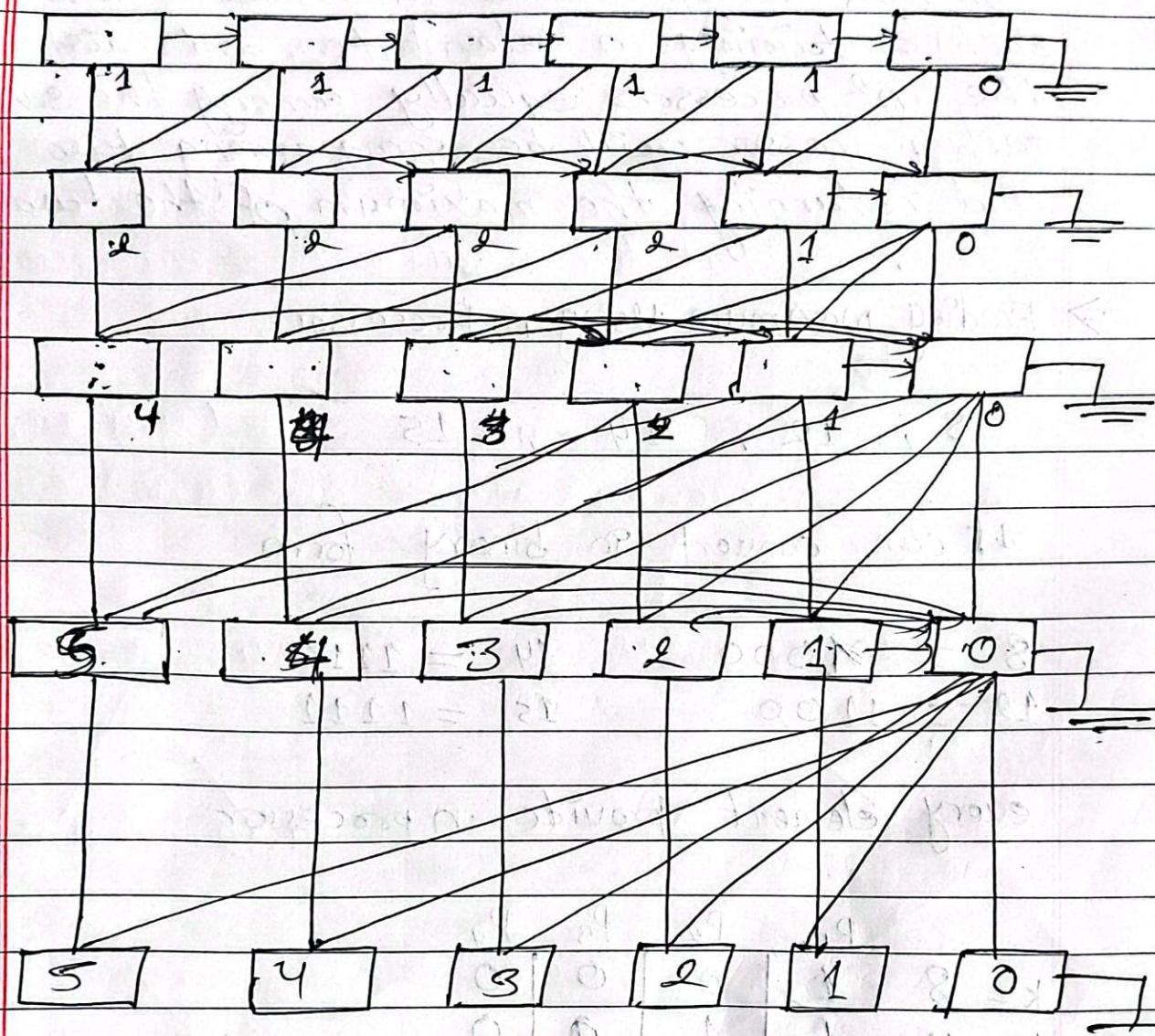
$\text{next}[i] = \text{next}[i] + (\text{next}[\text{next}[i]] - 1)$

$\text{Rank}[i] = \text{Rank}[i] + \text{Rank}[\text{next}[i]]$

$[i]$



Example :-



The sorting rank is 5, 4, 3, 2, 1, 0 (↑)

~~sequential algorithm - $O(n^2)$~~ - total running time = $O(n \log n)$, total work = $O(n \log n)$
 $\text{efficiency} = O(n)/O(n \log n) = 1/\log n$ not work optimal

\Rightarrow maximal selection with n^2 processor +

- maximal selection with n^2 processor refers to a parallel algorithmic problem that involves selecting the maximum element from a given set using a parallel computing system with n^2 processors. The problem assumes that the elements of the set are stored in a distributed manner across the processor.

- It is found the maximum element efficiently.
- It has array divided into ~~two~~ n subarrays, each containing n elements, and distribute the n^2 processors equally among the subarrays. Each processor will be comparing two elements and returning the maximum of the two so-on.

⇒ Finding maximum using n Processor:-

8, 12, 14, 15

It can convert in binary form.

$$8 = 1000$$

$$14 = 1110$$

$$12 = 1100$$

$$15 = 1111$$

every element provide in processor.

	P_1	P_2	P_3	P_4
$K=8$	1	0	0	0
$K=12$	1	1	0	0
$K=14$	1	1	1	0
$K=15$	1	1	1	1

→ $K_1, K_2, \& K_3 \rightarrow$ eliminate
 $K_1 \& K_2$ eliminate.

→ K_1 only eliminate

1 K or $n^{1/2}$ processor
 $n^{1/2}$ elements.

K_1
K_2
K_3
K_4

$$T(n, P) = 1$$

$$T(2) = O(1)$$

which is $T(n) = (\log \log n)$

Note $n = 2^{2^P} \Rightarrow 2^q = \log n \Rightarrow q = (\log \log n)$

Total work not optimal $O(n \log \log n)$

\Rightarrow Merging

- problem: Given 2 sorted sequences $x_1 = k_1, k_2, \dots, k_m$ and $x_2 = k_{m+1}, k_{m+2}, \dots, k_{2m}$.

Assume each sequence has m distinct elements, and m is an integral power of 2.

- The goal: To produce a sorted sequence of $2m$ elements.
- Best sequential algorithm is $O(m)$.
- For each $k_i \in x_1$, we know that it is the rank # i element in x_1 . We allocate a single processor to perform a binary search on x_2 and figure out q (the number of elements in x_2 that are less than k_i). Then we know that k_i is the rank #($q+i$) element in $x_1 \cup x_2$.
- For each element in x_2 , a similar procedure can be used to compute its rank in $x_1 \cup x_2$.
- we can use $2m$ processors, one for each element. An overall rank can be found for each element using binary search in $O(\log m)$ time. Merging can be done in $O(\log m)$ time.

\Rightarrow Odd even merge

- step 1: If $m=1$, merge two sequence with 1 comparison.
- step 2: partition x_1 into their odd and even parts, i.e. $x_1^{\text{odd}} = k_1, k_3, k_5, \dots, k_m$, and $x_1^{\text{even}} = k_2, k_4, \dots, k_m$. Similarly, partition x_2 into x_2^{odd} and x_2^{even} .
- step 3: Recursively merge $x_1^{\text{odd}}, x_2^{\text{odd}}$ (and $x_1^{\text{even}}, x_2^{\text{even}}$) using m processors.

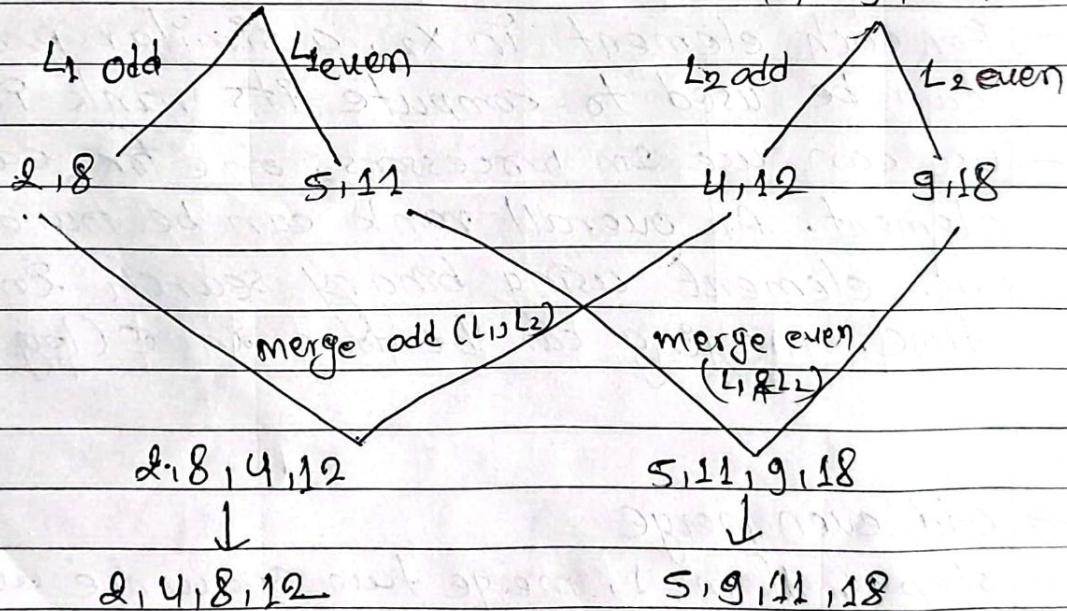
Let $L_1 = U_1, U_2, \dots, U_m$ ($U_2 = U_{m+1}, U_{m+2}, \dots, U_{2m}$)
be the result.

- Step 4: Form a sequence $L = U_1, U_{m+1}, U_2, U_{m+2}, \dots, U_{m+3}, \dots, U_m, U_{2m}$ Compare every pair (U_{m+i}, U_{i+1}) , p.e. $(U_{m+1}, U_2), (U_{m+2}, U_3), \dots$
Interchange elements if they are out of order. Output the resultant sequence.

Example :-

Let $X_1 = (2, 5, 8, 11)$ & $X_2 = (4, 9, 12, 18)$
~~so~~ odd = $(2, 8)$, even = $(5, 11)$ & odd = $(4, 12)$, even = $(9, 18)$
 so that

$$X_1 = 2, 5, 8, 11 \quad X_2 = 4, 9, 12, 18$$



Shuffling

$2, [5, 4], [9, 8], [11, 12], 18$

If required interchange, LL

$2, 4, 5, 8, 9, 11, 12, 18$

⇒ The final result = $2, 4, 5, 8, 9, 11, 12, 18$
 is sorted order

⇒ Odd even merge sort - :

- step 1: if $n \leq 1$ return X
- step 2: use n CPU's to partition input X of n elements into two lists X_1 and X_2 . each with $n/2$ elements
- step 3: use $n/2$ CPU's to sort X_1 recursively and $n/2$ CPU's to sort X_2 recursively. Let X_1^* and X_2^* be the result sorted lists.
- step 4: use odd even merge to merge two sorted lists using n CPU's.

⇒ Analysis - :

$$T(n) = O(1) + T(n/2) + O(\log n)$$

$$= \log(n) + \log(n/2) + \log(n/4) + \dots + \log(n/2^k); k = \log_2 n$$

$$= \log(n) + [\log(n) - \log 2] + [\log(n) - \log 4] + \dots + [\log(n) - \log(n)]$$

$$\leq \log(n) * \log(n)$$

~~$T(n) = \log(n) * \log(n)$~~

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

Total work $O(n \log^2 n)$

It is not work optimal.

⇒ Preparata's Algorithm - :

- It is known as the closest pair algorithm, is an algorithm used to find the pair of closest points in a two-dimensional plane. It was developed by Franco Preparata and Michael Ian Shamos in 1975.
- algorithm works by dividing the plane into smaller regions and recursively finding the closest pair of points in each region.
- Short
 - divide into four parts
 - find closest
 - find minimum distance & combine result.

Unit - 4 :- Mesh and Hypercube Algorithms :-

⇒ Mesh Algorithm :-

- A mesh is an $a \times b$ grid in which there is a processor at each grid point. The edges correspond to communication lines and are bidirectional. Each processor of the mesh can be labeled with a tuple (p, q) where $1 \leq p \leq a$ and $1 \leq q \leq b$.
- Every processor has a RAM with some local memory. Hence, each processor can perform any of the basic operations such as subtraction, addition, multiplication, comparison, local memory access and so on. The computation is assumed to be synchronous.

- We consider square meshes, that is meshes for which $a=b$. A $\sqrt{p} \times \sqrt{p}$ mesh is shown in figure(a).
- A closely related model is the linear array. A linear array consists of p processors named $(1, 2, \dots, p)$ connected as follows:

- a) Processor i is connected to the processors $i-1$ & $i+1$ for $2 \leq i \leq p-1$.
- b) Processor 1 is connected to processor p and processor p is connected to processor 1.

processor 1 and p are known as boundary processors. processor $i-1$ and $i+1$ are known as left neighbour (right neighbour) of i .

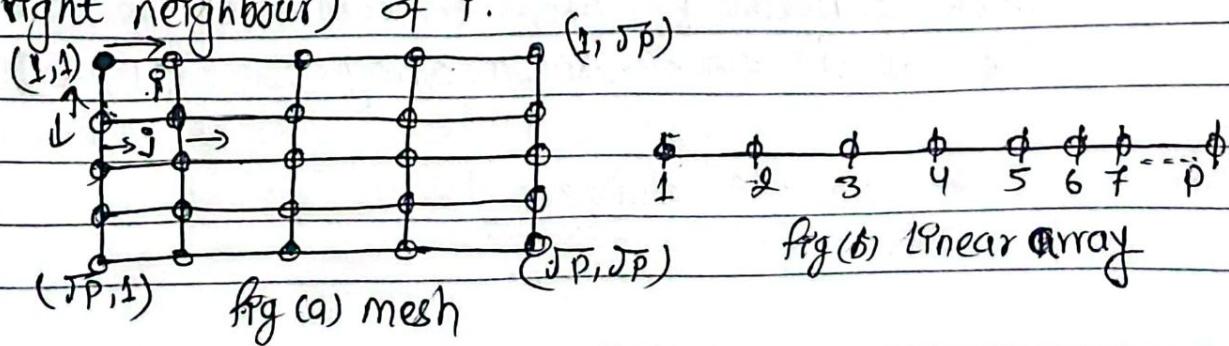
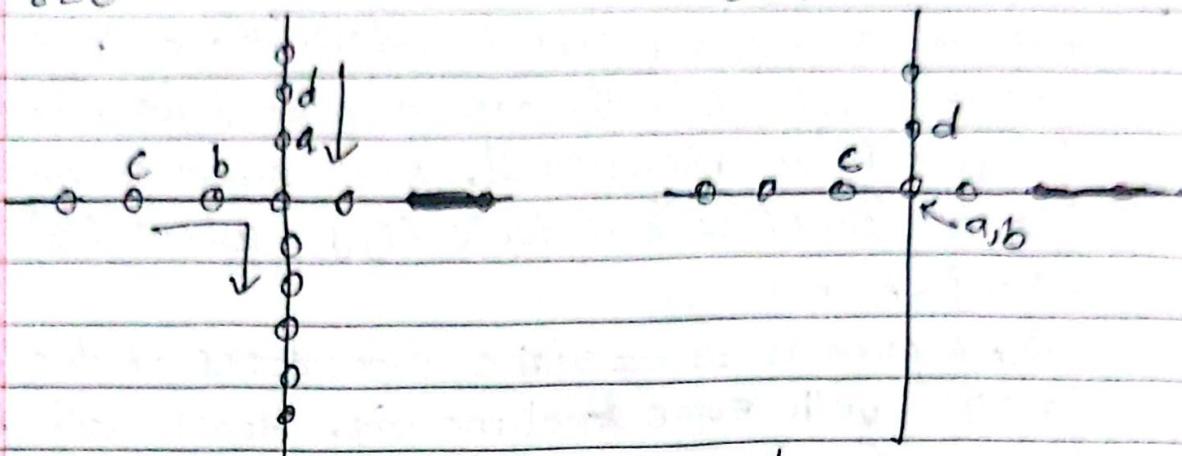


Fig (b) Linear array

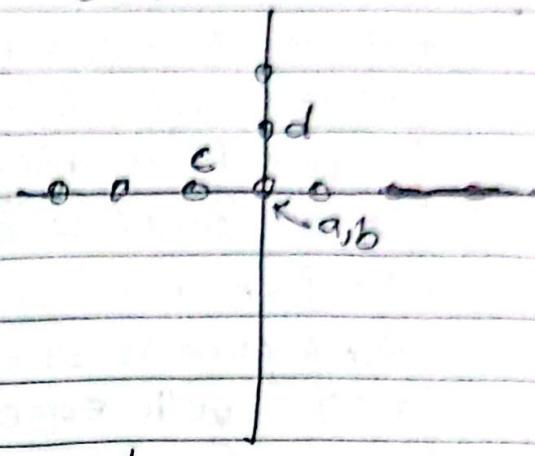
Fig (a) mesh

Consider 4 packets a, b, c and d their final destination are shown in figure below :

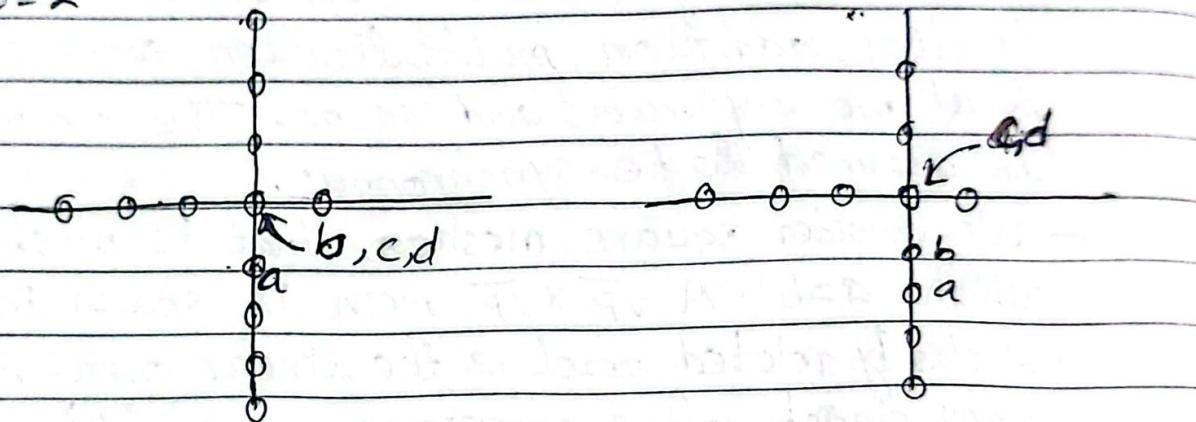
$t=0$



$t=1$

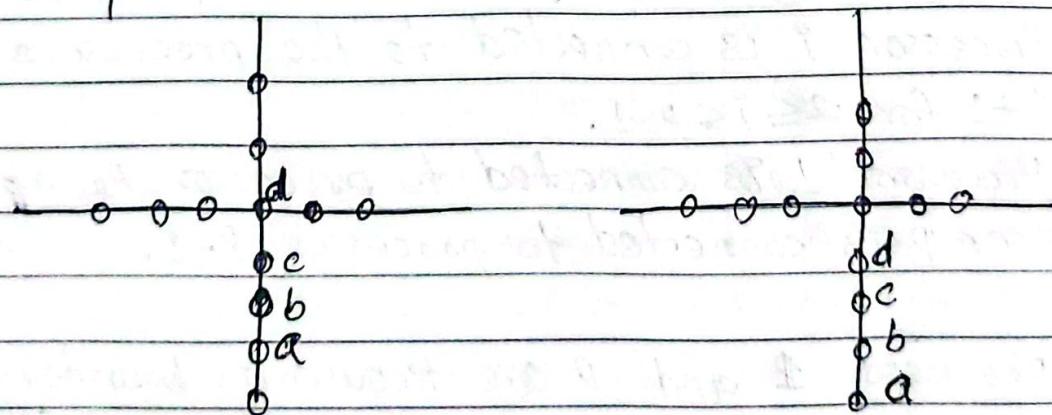


$t=2$



$t=3$

$t=4$

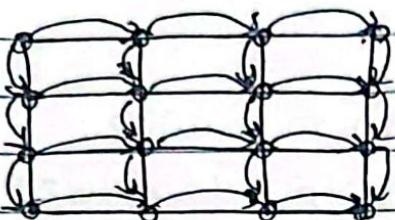


$t=5$

Since it needs $p-1$ steps to complete the linear array,
the worst case of linear array is $\mathcal{O}(p-1)$.

(12 possible changes)

⇒ Broadcast in mesh :-



- In the case of a $\sqrt{p} \times \sqrt{p}$ mesh broadcasting can be done in two ~~bi-directional~~ phases.
 - 1) If (i,j) is the processor of message origin in phase 1, m could be broadcast to all the processors in row j .
 - 2) In phase 2, broadcasting of m is done in each column.
- This algorithm takes $\leq 2(\sqrt{p}-1)$ steps which is expressed as :
 The broadcasting on a p processor linear array can be completed in p steps or less on a $\sqrt{p} \times \sqrt{p}$ mesh, the same can be performed in $\leq 2(\sqrt{p}-1) = O(\sqrt{p})$ time.
- Broadcasting can be done in the following ways :
 - a) Broadcasting without constraints (Redundancy).
 - b) Broadcasting with constraints (No redundancy).

⇒ Prefix computation in mesh :-

- consider a $\sqrt{p} \times \sqrt{p}$ mesh. The problem of computing prefix sum on the mesh can be reduced to three phases in each of which computation is local to the individual rows and columns. This algorithm assumes the row major indexing scheme.

The prefix computation on mesh can be done in following step :

step-1 : prefix computation row-wise.

Row i (for $i = 1, 2, \dots, \sqrt{P}$) computes the prefix of its p elements.

step-2 : prefix computation column wise,

last column only.

only \sqrt{P} column computes the prefix of sum computed in step 1.

step-3 : shift 1 down to column wise last column only & broadcast in its respective row (column-wise) & perform prefix computation.

shifting $\rightarrow (i, \sqrt{P}) \xrightarrow{\text{shifting}} (i+1, \sqrt{P})$

Broadcast : Broadcast (i, p) in row $i+1$ (for $i = 1, 2, \dots, \sqrt{P}-1$) node j in row i updates final results.

or simply

step 1 : Each rows computes its prefix

step 2 : only last column computes its prefix

step 3 : Shift the prefix of last column one down

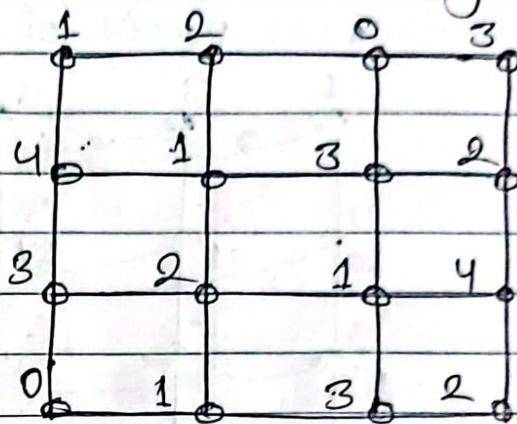
step 4 : for second row to last row broadcast

the prefixes to each row and add with the individual prefix of step 3.

The worst case complexity analysis for prefix computation is $O(2\sqrt{P} - 1)$.

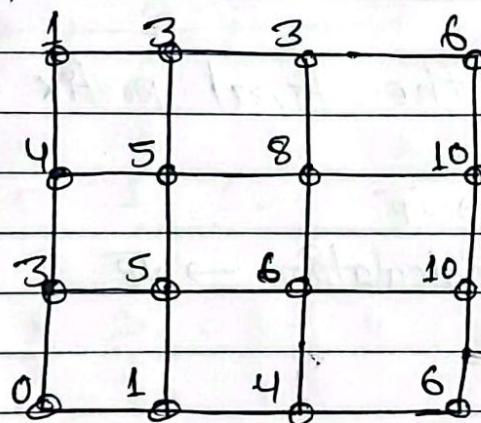
consider the data on the 3×3 mesh and the problem of prefix sums under the row major indexing scheme.

Given example 98 :

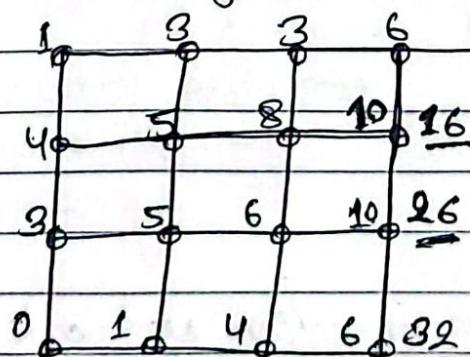


Step 1

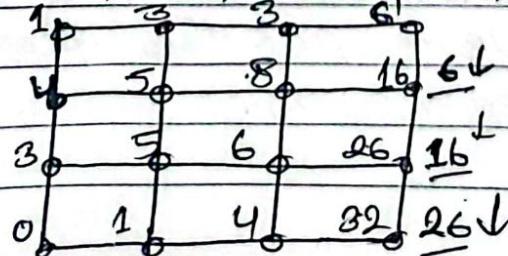
Step-1 :- prefix compute each ~~column~~ rows:



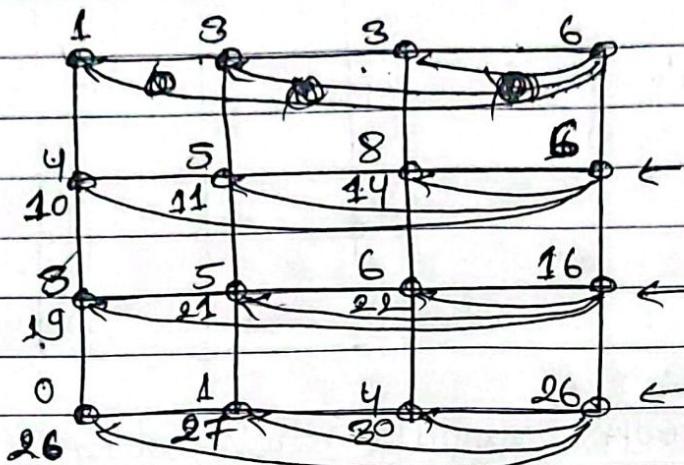
Step-2 :- Only last column computes its prefix.



Step-3 :- Shift the prefix of last column one down.



Step-4:- For second row to last row broadcast the prefixes to each row and add with the individual prefix of step 3.



Hence this is the final prefix computation.

prefix calculation $\rightarrow \sqrt{P}$

last column prefix calculation $\rightarrow \sqrt{P}$

shifting $\rightarrow 1$

updating $\rightarrow 1$

Broadcasting $\rightarrow \sqrt{P}$

$$\begin{aligned} & 3\sqrt{P} + 2 \\ & = O(\sqrt{P}) \quad \# \end{aligned}$$

\Rightarrow Data Concentration :-

- In a p -processor interconnection network. Let there be $d < p$ data items distributed arbitrarily with atmost one data item per processor.
- Problem is to move or pack the given elements into the first d processors.

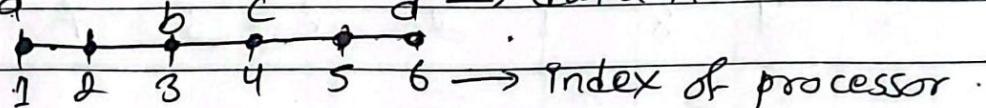
\Rightarrow Linear Array :-

Let L be a p -processor linear array with d data items.

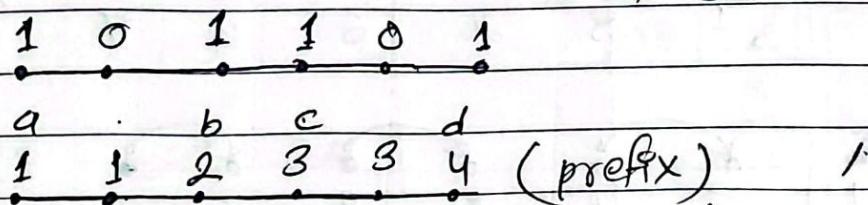
Assumptions:-

- If processor i has a data item, then set $x_i = 1$ otherwise $x_i = 0$, Let prefixes of the sequences.
- x_1, x_2, \dots, x_n be y_1, y_2, \dots, y_n
- If processor i has data item, then the destination of this item is y_p , once calculation is over, there are routed.

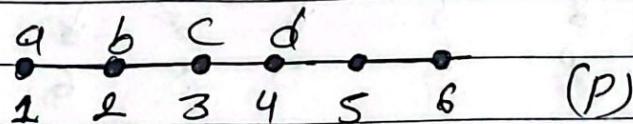
~~eg~~ a b c d \rightarrow data items



$\therefore d < p$ satisfies.



Routing :-



Complexity Analysis :-

prefix calculation :-: P

Routing :-

$$\frac{P}{2P} \quad \therefore O(P) \neq$$

\Rightarrow Sorting :-

- Given a sequence of n -keys, the problem of sorting is to rearrange the sequence in either ascending or descending order.

⇒ Linear Array :-

For ($i=1$; $i \leq p$; $i++$)

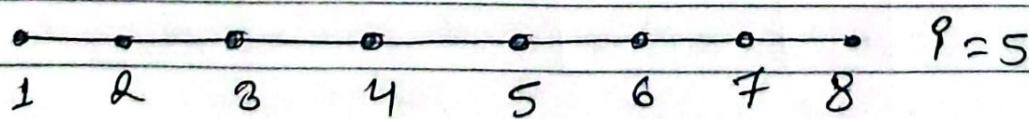
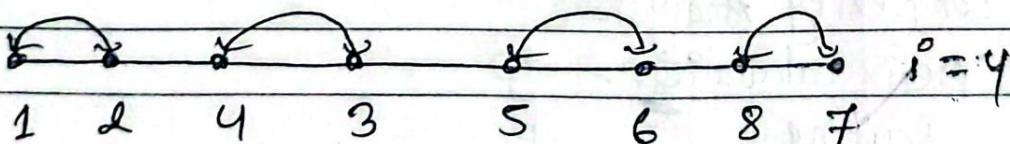
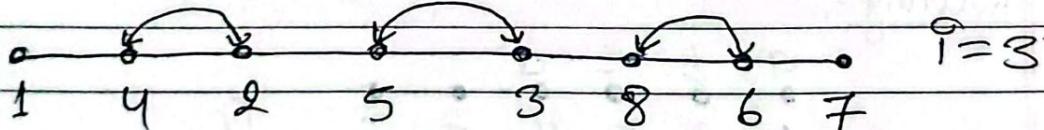
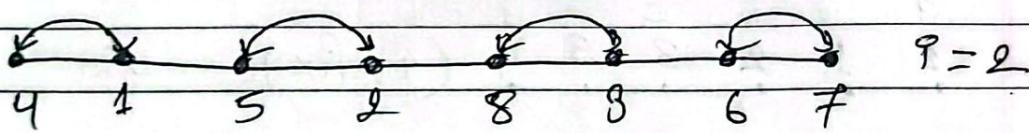
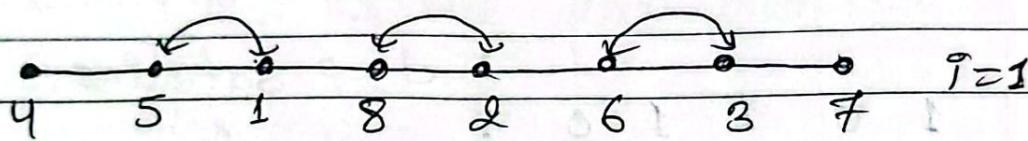
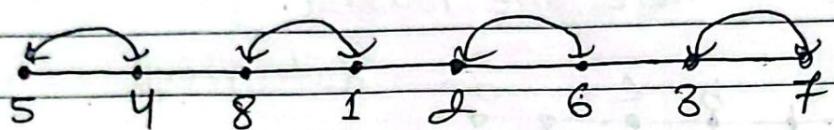
If i is odd, compare and exchange keys at processor $i-1$ and i

else

compare and exchange keys at processor $i-2$ & $i+1$.

This algorithm is also known as odd-even transposition.

Example:-



complexity $O(p)$ as in every execution of 'for loop' the sorting takes 1 unit of time.

⇒ Sorting on mesh :-

Algorithm

for ($i=1$; $i \leq \log p + 1$; $i++$)

↓

If i is even

sort the columns (top to bottom)

else

sort the rows (alternative order)

↓

Example :-

$i=1$

15	12	8	82		8	12	15	82	Asc
7	13	6	17	→	17	13	7	6	Desc
2	16	19	25		2	16	19	25	Asc
18	11	5	3		18	11	5	3	Desc

$i=2$ (Col Asc)

$p=3$

2	11	5	3		2	3	5	11	Asc
8	12	7	6	→	12	8	7	6	Desc
17	13	15	25		13	15	17	25	Asc
18	16	19	82		82	19	18	16	Desc.

$i=4$ (Col Asc)

$p=5$

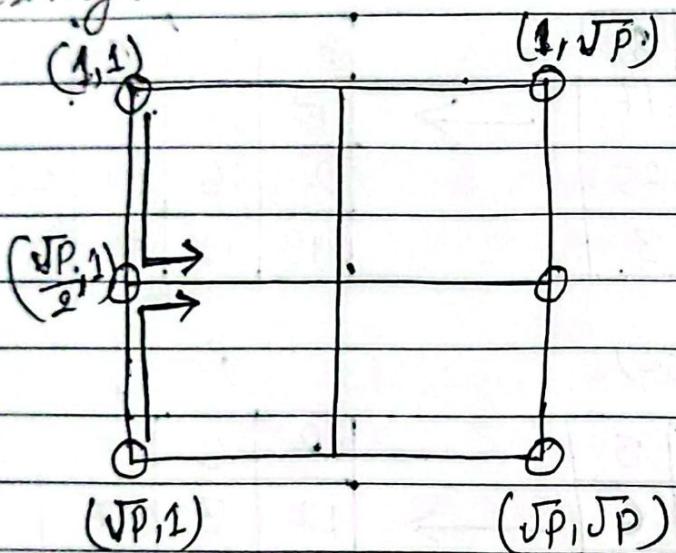
2	3	5	6		2	3	5	6	Asc
12	8	7	11	→	12	11	8	7	Desc
13	15	17	16		13	15	16	17	Asc
32	19	18	25		32	25	19	18	Desc.

↗

complexity : $O(\sqrt{p} \log p)$ #

⇒ A Greedy Algorithm for PPR on a Mesh:

- For the PPR (Parallel Prefix Reduction) problem on a $\sqrt{P} \times \sqrt{P}$ mesh, we see that if a packet at processor $(1,1)$ has (\sqrt{P}, \sqrt{P}) as its destination, then it has to travel a distance of $2(\sqrt{P}-1)$. Hence $2(\sqrt{P}-1)$ is a lower bound on the worst-case routing time of any packet routing algorithm.
- It used to compute the prefix sum of a given input array or list. This is one approach to solve this problem efficiently in a parallel setting.



- It uses packet routing algorithms with linear array.
- Let 'q' be an arbitrary packet with (i,j) as its origin and (k,l) as its destination.
- packet q uses a two phase algorithm.
 - one It travels along column j to row k shortest.
 - two It traverses along row k to its correct destination using shortest path. So on repeatedly.

~~select~~

(n > p)

\Rightarrow Randomized Algorithm for $n = p - 1$ (selection) :-

- Given a sequence of n keys and an integer p , $p \leq p \leq n$, the problem of selection is to find the p th smallest key from the sequence.
- If $X = k_1, k_2, \dots, k_n$ is the input, the algorithm chooses a random sample (call it S) from X and identifies two elements d_1 and d_2 from S . The elements chosen are such that they bracket the element to be selected with high probability and also the number of input keys that are in range $[d_1, d_2]$ is small.
- After choosing d_1 and d_2 we determine whether the element to be selected is in the range $[d_1, d_2]$. If this is the case, we proceed further and the element to be selected is the $(p-1)x_1$ the element of x_2 . If the element to be selected is not in range $[d_1, d_2]$, we start all over again.
- Selection from $n = p$ keys can be performed in $O(\sqrt{p})$ time on a $\sqrt{p} \times \sqrt{p}$ mesh.

\Rightarrow Randomized Selection for $n > p$:-

- when the number of key 'n' is larger than ~~select~~ the network size 'p' or $n > p$.

In a particular, assume that $n = p^c$ for some constant $c > 1$.

Each processor has n/p keys to begin with. The condition for the whole loop is changed to ' $(N > D)$ ' (where D is constant).

In step 1 a processor includes each of

its keys with probability $\frac{1}{N^{1/3}c}$.

so, this step now takes time $\frac{n}{p}$.

The number of keys in the sample is $O(N^{1/3}c) = O(\sqrt{p})$.

Step 2 remains the same and still takes $O(\sqrt{p})$ time.

Since there are only $O(N^{1/3}c)$ sample keys they can be concentrated and sorted in Step 3. In time $O(\sqrt{p})$.

Then each step takes time $O(\frac{n}{p} + \sqrt{p})$.

⇒ 4.2.1 Hypercube Algorithm - I

⇒ Hypercube - I

A hypercube of dimension d , denoted by H_d has $P=2^d$ processors. Each processor in H_d can be labelled with d -bits.

- we define hypercube as following terms:

- Dimension
- Coding of Processors.
- Hamming Distance
- Diameter.

Example

for $d=3$; i.e. dimension = 3

no. of vertex = $2^d = 2^3 = 8$

no. of processors = 8

Coding of processors = 000, 001, 011, 010, 110, 100, 101, 111

- coding should be carried out in a such a way that the difference of code of two adjacent processors should be 1, which is called hamming distance.
- The diameter of a d -dimensional hypercube is d .
- The bisection width of a d -dimensional hypercube is $2^{(d-1)}$.
- The hypercube is highly scalable architecture. Two d -dimensional hypercubes can be easily combined to form a $d+1$ -dimensional hypercube.

⇒ Characteristics -

- Each processor number should differ by 1 bit only.
- Degree of H_d = No of interconnected processor of reference processor.
- calculate by hamming distance with no of bit position change.

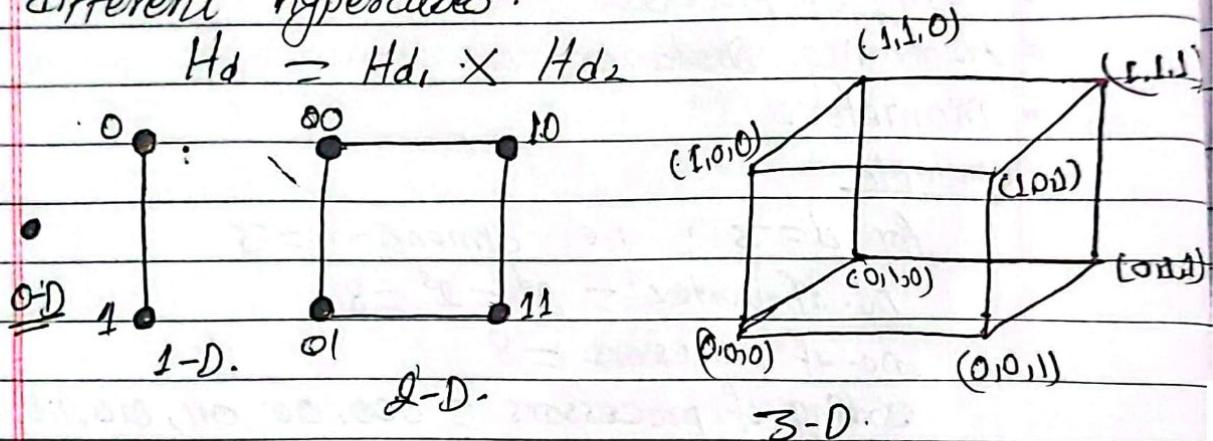
\Rightarrow Variables: ~~H_p, H_c, etc.~~:

- Sequential - at one unit of time a processor can communicate with only one of its neighbour.
- Parallel - at one unit of time a processor can communicate with all of its neighbour.

\Rightarrow Features :-

- The diameter of hypercube is $\log p$.
- A hypercube can be generated by combining different hypercubes.

$$H_d = H_{d_1} \times H_{d_2}$$

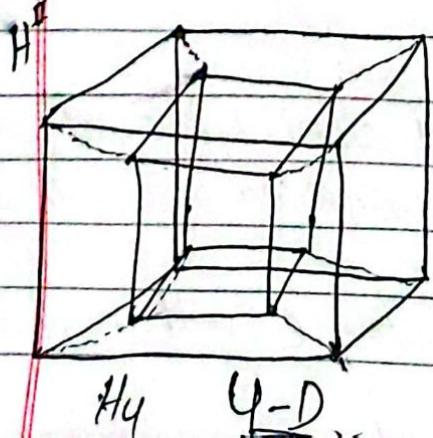


Eg. computational model of hypercube.

eg let's take (0,0,0)

0, 0, 0	0, 0, 0	0, 0, 0
0, 0, 1	0, 1, 0	1, 0, 1

only one bit changed then we can see that all points are hamming distance a part from pp processor.

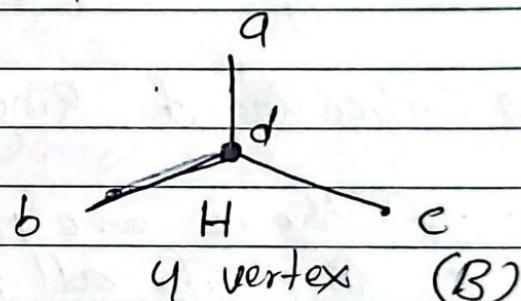
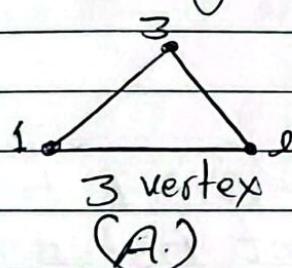


prefix(0) prefix(1)

The hypercube has several variants like butterfly, shuffle-exchange network and cube-connected cycle.

prefix (0)		prefix (1)	
0	0 0 0	1	0 0 0
0	0 0 1	1	0 0 1
0	0 1 0	1	0 1 0
0	0 1 1	1	0 1 1
0	1 0 0	1	1 0 0
0	1 0 1	1	1 0 1
0	1 1 0	1	1 1 0
0	1 1 1	1	1 1 1

⇒ Embedding of Network :-



$$1 \rightarrow b, 2 \rightarrow c, 3 \rightarrow q$$

$$1-2 \rightarrow bd - dc$$

$$1-3 \rightarrow bd - da$$

1) Expansion = $\frac{|V_2|}{|V_1|} = \frac{4}{3} = 1.33$

2) Dialation :- longest path value or max length of path that any link of (A) is mapped.
⇒ Dialation = 2

3) Congestion :- maximum number of times any node of (B) involves while mapping path of A with (B) = 2.

⇒ congestion of bd = 2.

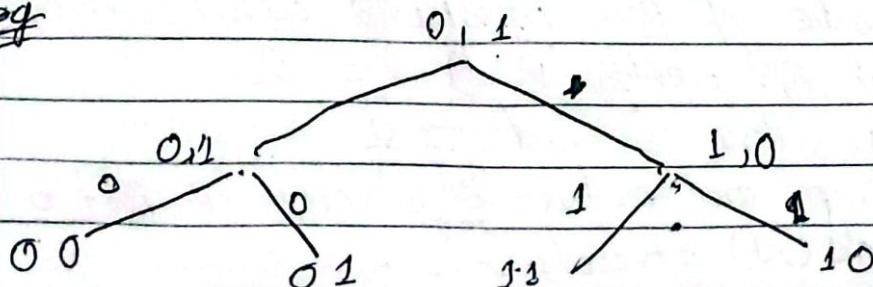
∴ changing one network structure into another network (N) structure

- changing network (m) structure to network (N) structure.
- perform calculations on network structure (N).
- Result obtained is mapped to the original network structure (m).
- It helps in choosing the best alternative structures.
- The algorithm that has been analyzed in one structure can be used in another structure that is transformable without testing.

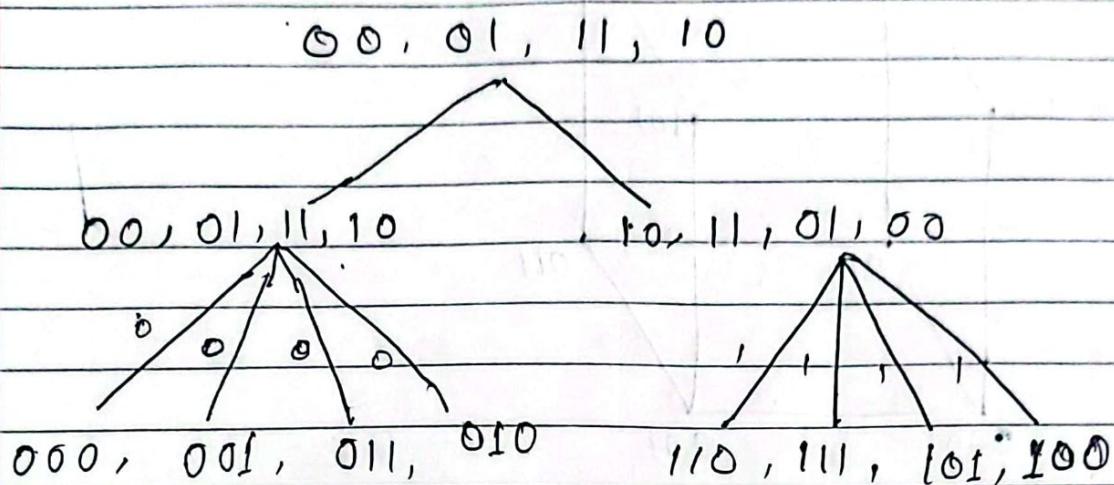
\Rightarrow Embedding of Ring :-

- If there are processors from 1 to p, it's a ring if all subsequent processors are linked and 1 is linked with p. the transformation is carried out with the help of Gray code.
- The graycode of order k, denoted by G_{1k} defines an ordering among all the k bit binary numbers.
- G_{1k} can be defined recursively as
 - G_1 is $0\ 1$
 - G_k is $0[G_{k-1}]^r, 1[G_{k-1}]^r$
 $[G_{k-1}]^r$ is reverse.

eg

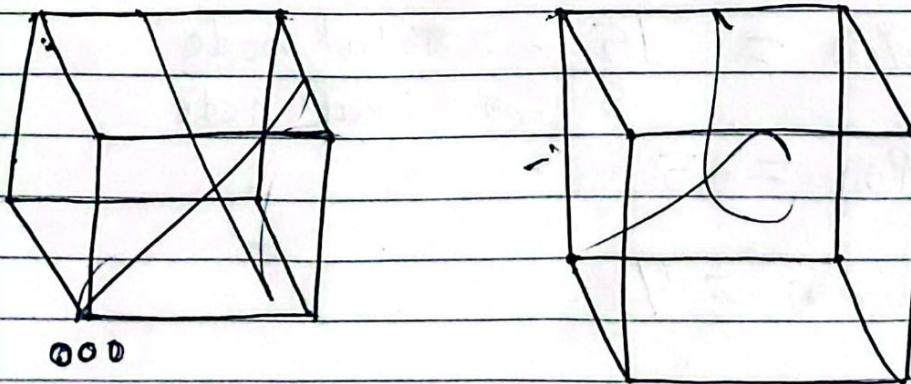


Example :- 00, 01, 11, 10



⇒ Embedding of Binary Tree :-

- A p -leaf ($P = 2^d$) binary tree T can be embedded into H_d .
- more than one processor of T have to be mapped into same processor of H_d .
- If tree leaves are $0, 1, 2, \dots, P-1$ then leaf is mapped to r th processor of H_d .
- Each processor of T is mapped to the same processor of H_d as its leftmost descendant leaf.



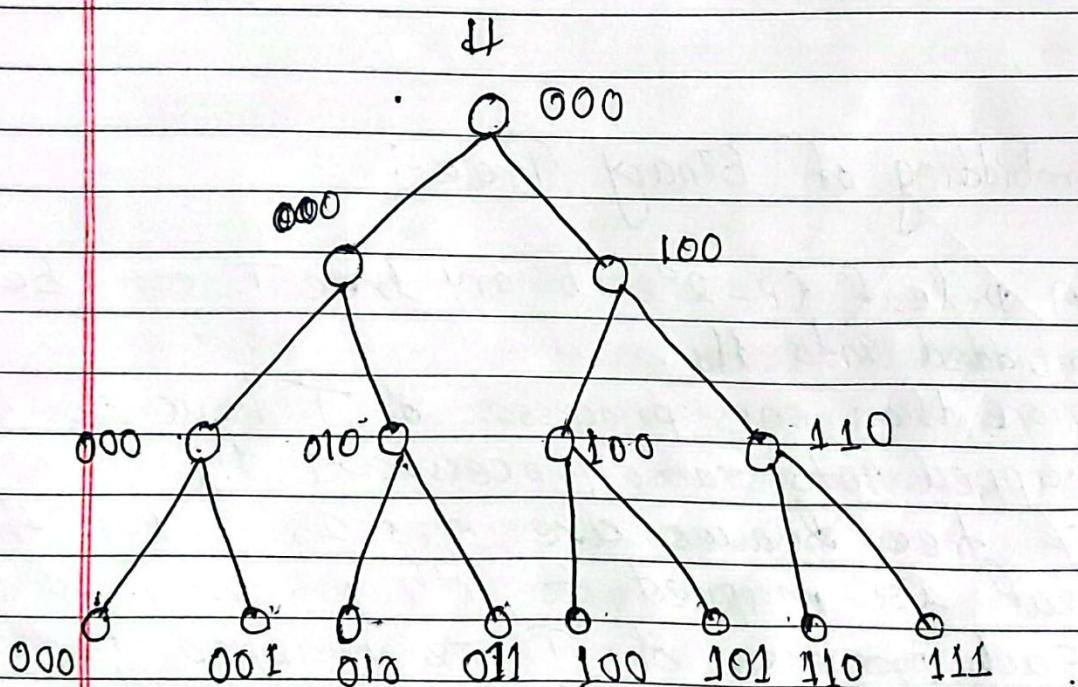
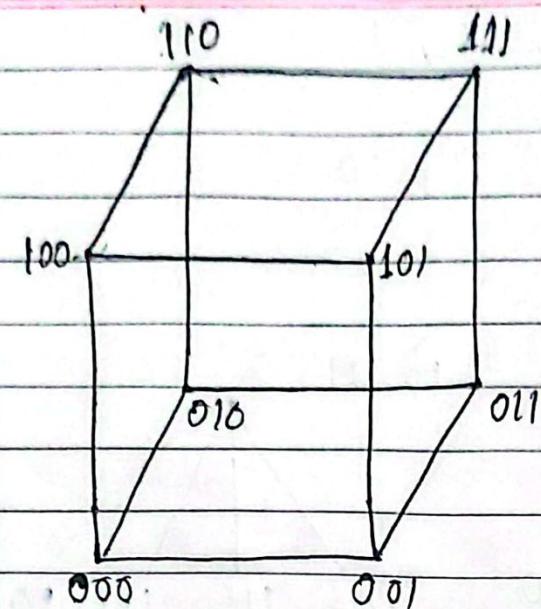


Fig Embedding of hypercube.

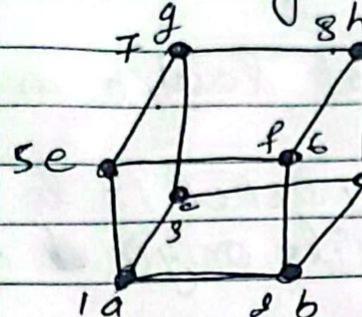
$$\text{Expansion} = \frac{15}{8} \rightarrow \text{Total node}$$

$$8 \rightarrow \text{Leaf node.}$$

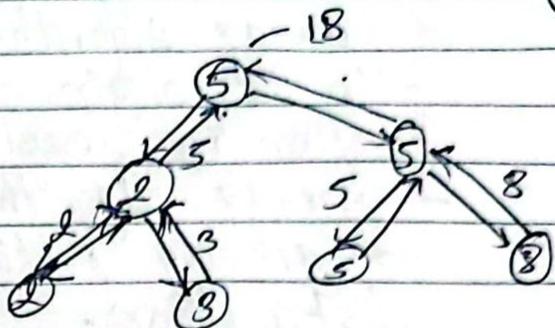
$$\text{Dilation} = 5$$

$$\text{Congestion} = 4$$

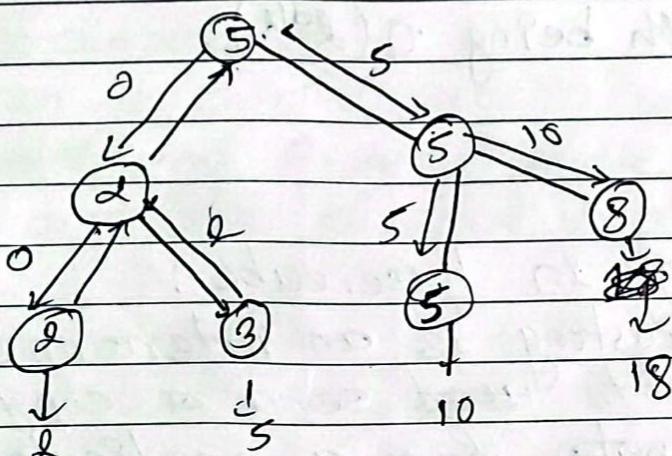
⇒ Embedding of Hypercube to tree Backtracking



⇒

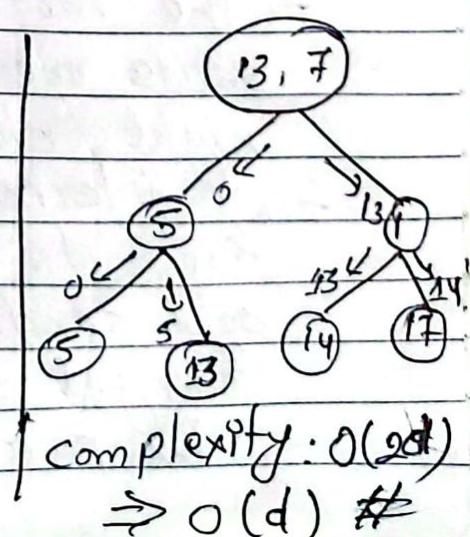
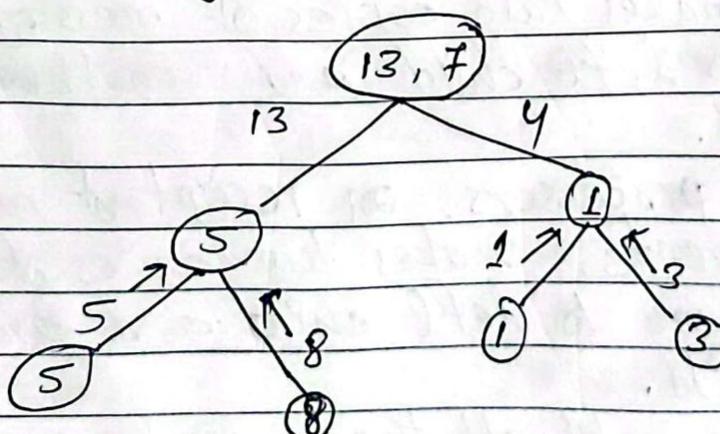


Forward pass



∴ Complexity : $O(\log n) = \log 2 \cdot d$

⇒ perform prefix computation of : 5, 8, 1, 3 on full binary tree representation.



complexity : $O(2d)$
 $\Rightarrow O(d) \#$

⇒ PPR Routing:-

⇒ Greedy Algorithm:-

- In Ba, origin of packet is at level 0 and destination is level d.
- Greedy algorithm for each packet is to choose a greedy path between its origin and destination.
- Distance travelled by any packet is d. Algorithm runs in $O(2^d)$ time, the average queue length being $O(2^{d/2})$.

⇒ Broadcasting in Hypercube:-

- The broadcasting is an interconnection networks. Is to ~~send~~ send a copy of message that originates from a particular processor to a subset of other processors. It is widely used in the design of several algorithms.
- To perform broadcasting on Hd, we employ the binary tree embedding.
- Assume a message m, at root of the tree is to be broadcast.
- The root makes two copies of message and sends one to left child and another to right child.
- On internal processors, on receipt of message from its parents, makes two copies of message and sends one to left child and another to right child.
- This proceed until all the node have a copy of m.

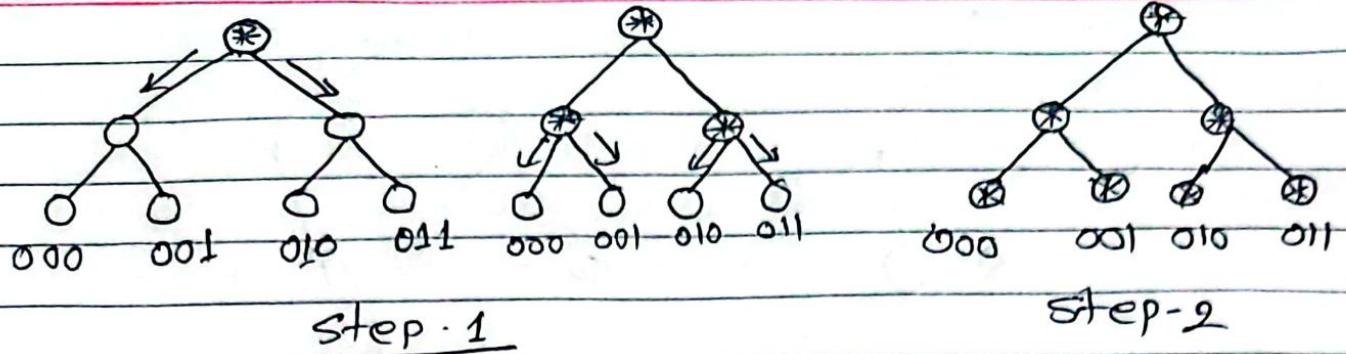


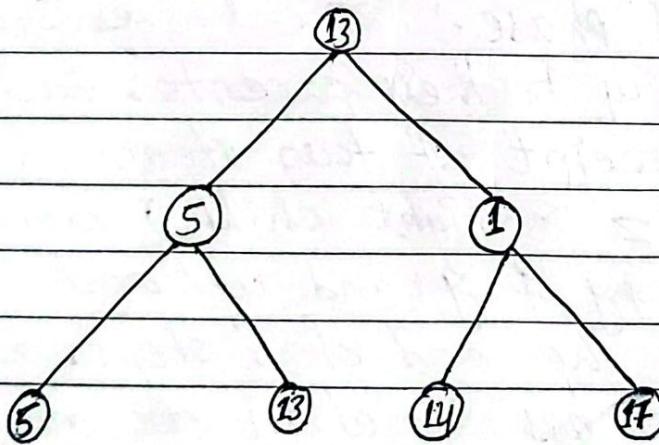
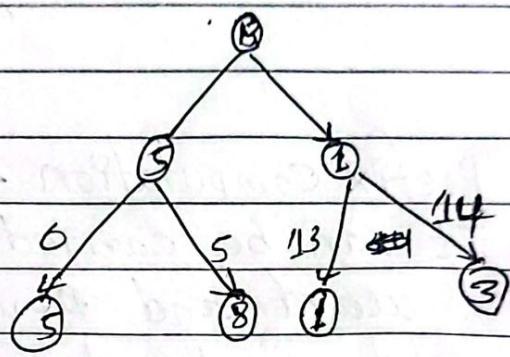
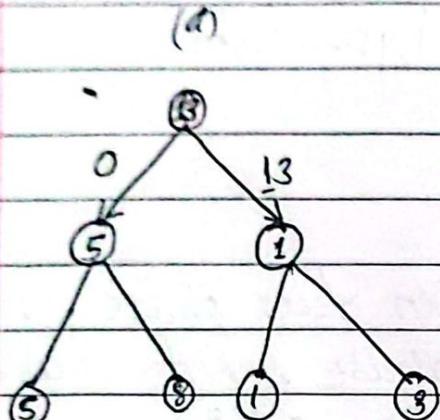
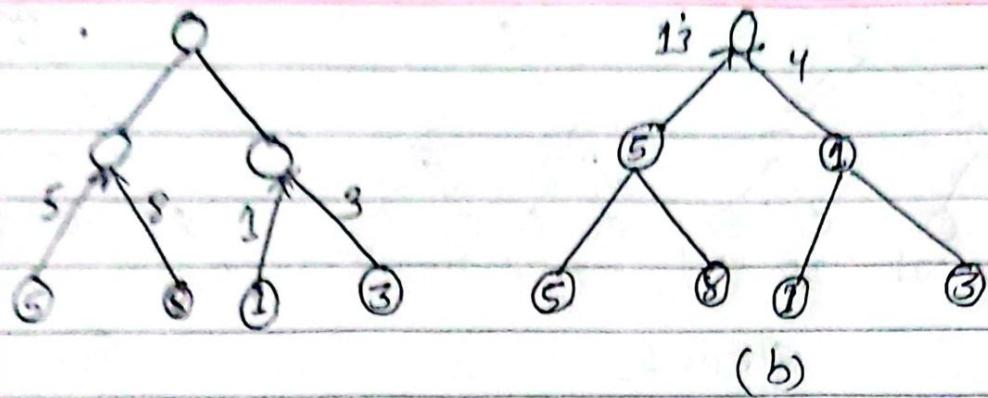
Fig: Broadcasting in hypercube.

⇒ Prefix computation :-

- It can be carried out in two phase. i.e. forward and reverse phase. At each step only one level of tree is active.

Forward phase:- The leaves start by sending their data up to their parents. Each internal processor on receipt of two items (y is left child and z is right child) computes $w = y \oplus z$, stores a copy of y and w and send w to its parent. At the end of d steps, each processor in the tree has stored in its memory the sum of all data items in the subtree rooted at this processor. The root sum of all elements in tree.

Reverse Phase:- The root starts by sending zero to its left child and y to its right child. Each internal processor on receipt of a datum (say q) from its parent q to its left child and $q+y$ to its right child. When the left gets a datum q from its parent, it computes $q+x$ and stores it as final result.



Eg prefix computation on a binary tree.

- prefix computation on 2^d - leaf binary tree
are kept ~~data item~~ as well as ~~1d~~ can
be done in $O(d)$ steps.

\Rightarrow Data concentration :-

- on H_d , there are $k < p$ data items distributed arbitrarily with atmost one data per processor.

The problem of data concentration is to move the data into the processor $0, \dots, k-1$ of H_1 .

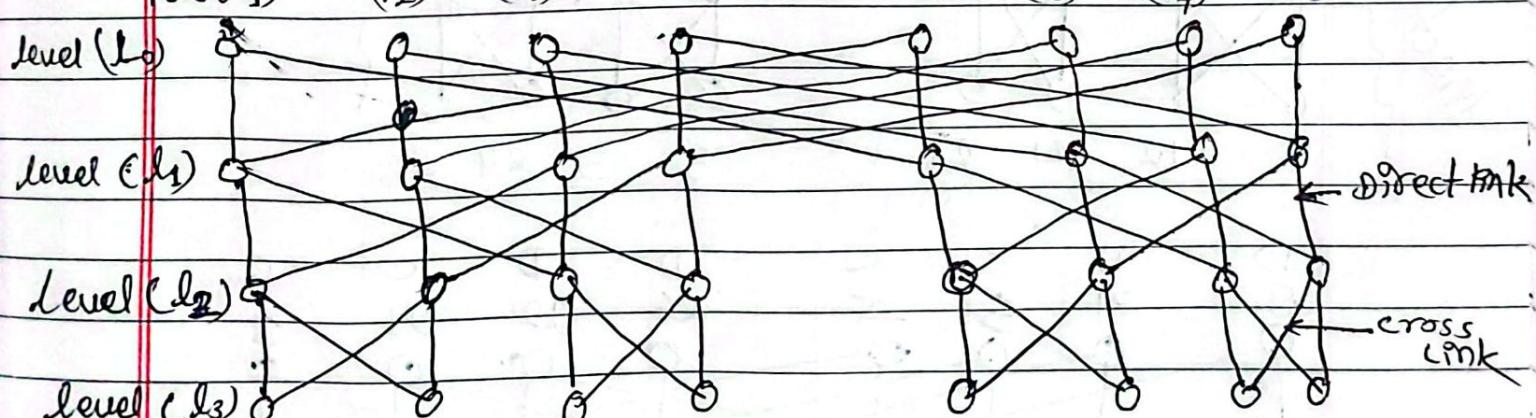
I. prefix computation by using tree (binary).

II. Route by using butterfly network.

\Rightarrow Butterfly Network (B_d) :-

B_d is also one of the embedding technique to solve problems of sorting, data concentration, prefix computation, selection etc. In a massive parallel connected network.

000	001	010	011	100	101	110	111
row(r_1)	(r_2)	(r_3)	(r_4)	(r_5)	(r_6)	(r_7)	(r_8)



1) vertical line

2) cross link

3) $1^{st} \rightarrow 2^{nd}, 3^{rd} \rightarrow 1^{st}$ cross

processor (p) = 32 $\rightarrow (d+1)2^d$

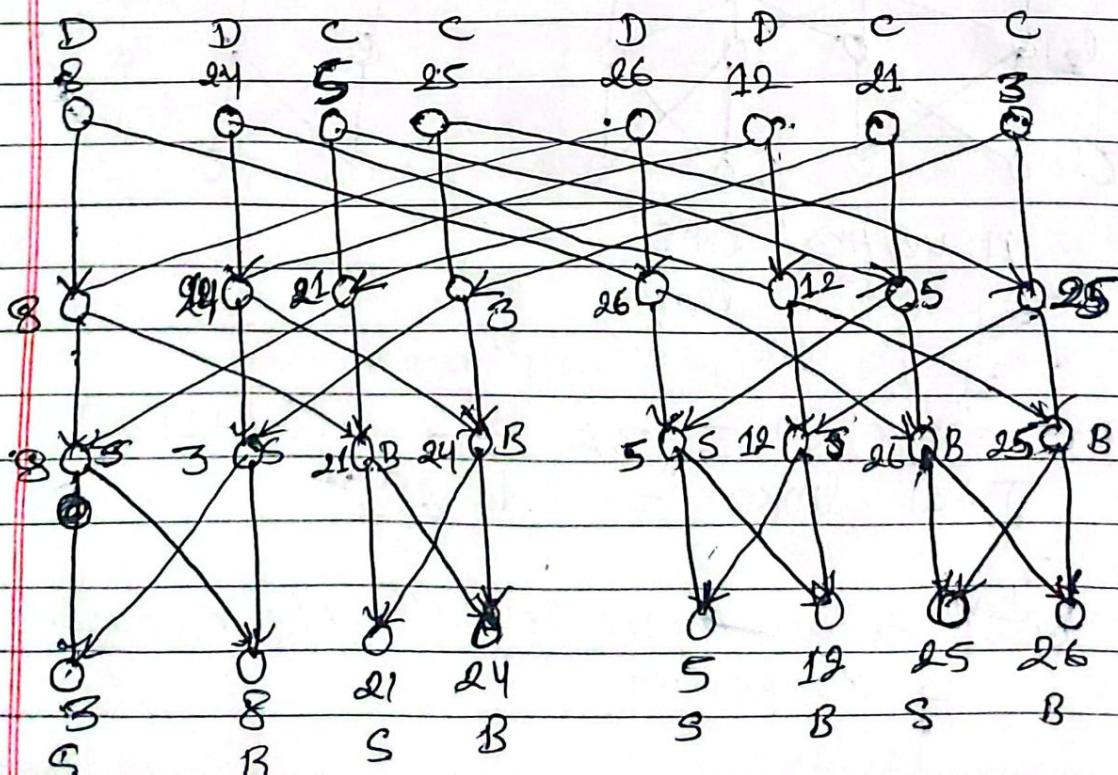
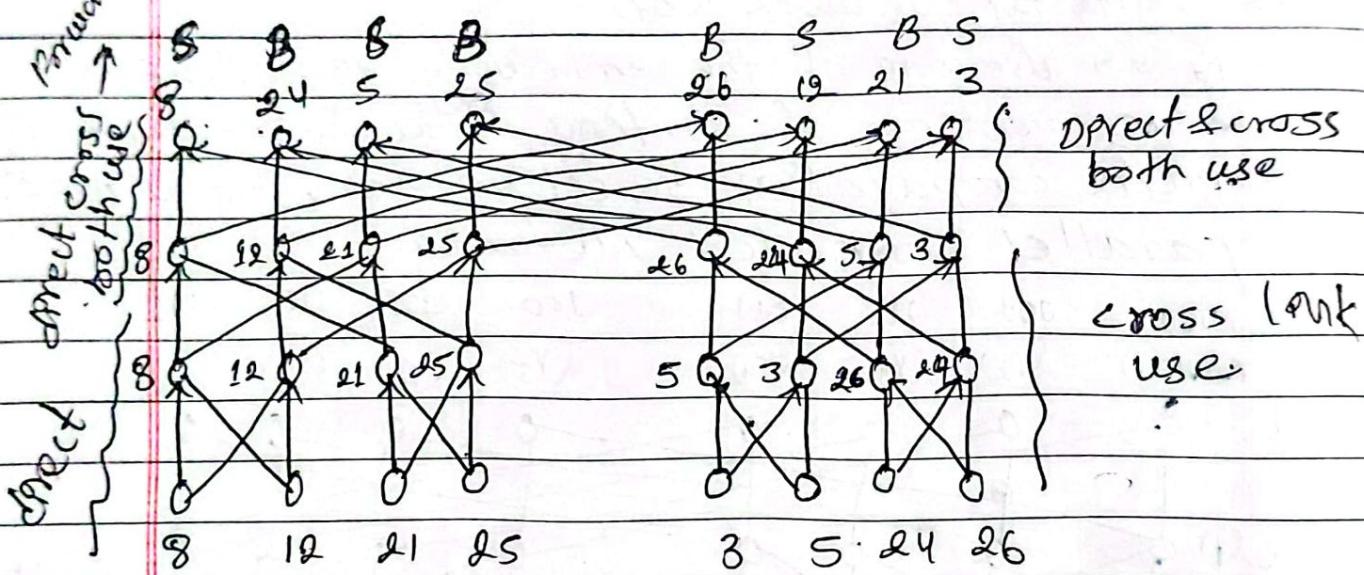
Total links = $d \cdot 2^{d+1}$

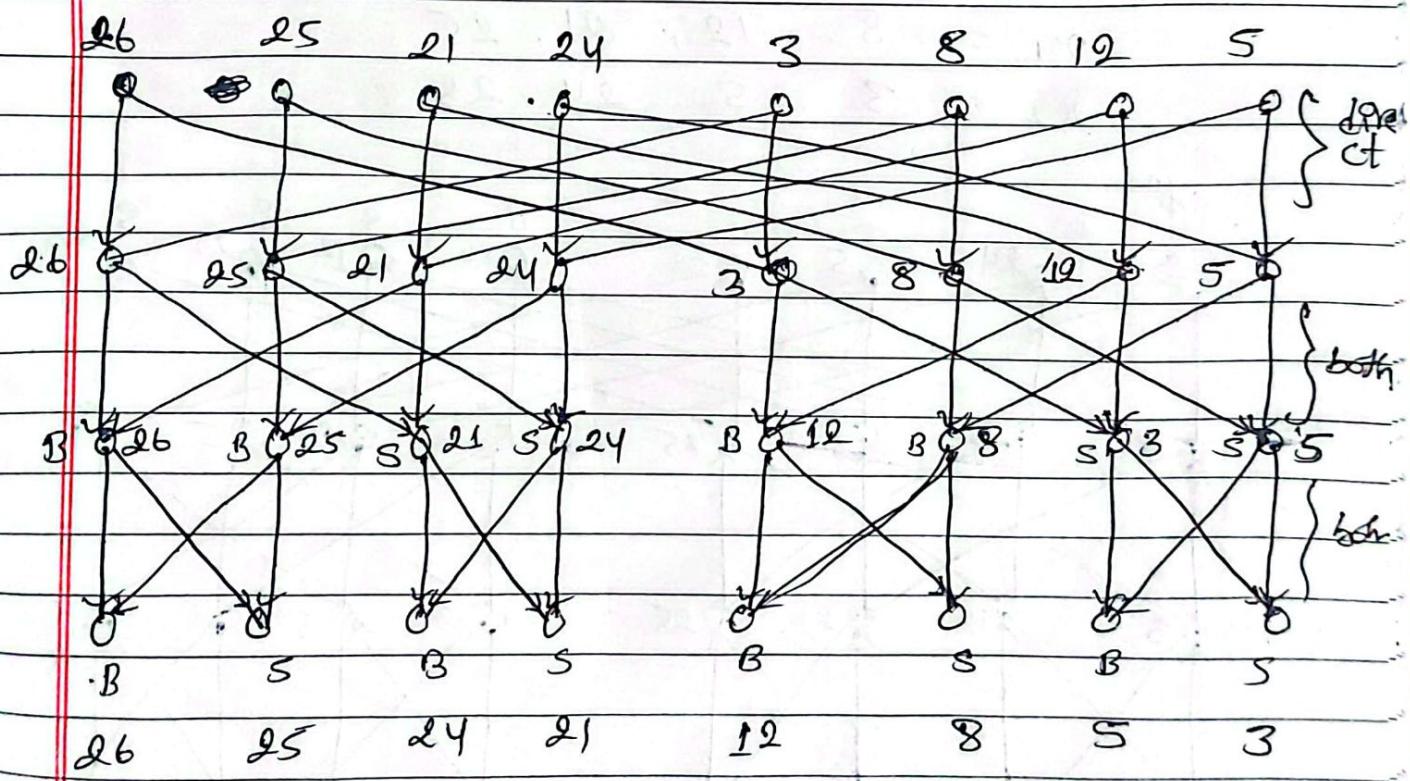
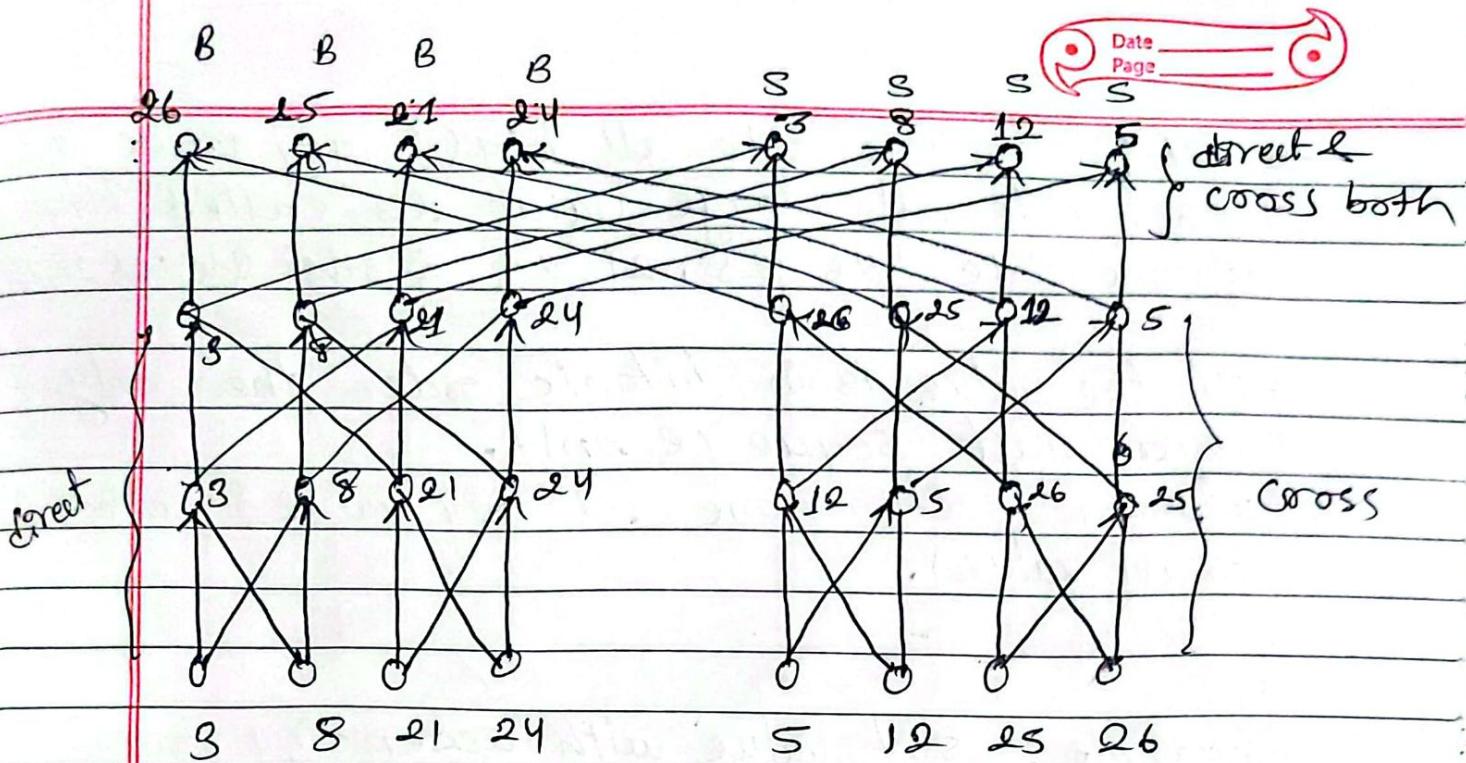
~~Fig~~ Above figure is butterfly network of 3 dimension.

⇒ Odd-even merge sort (Butterfly network)

- It consists of two phases.
- I-phase: separate odd and even into [$x_1 \rightarrow x_2$ given sequence to $O_1; E_1, O_2, E_2$]
- II-phase: Recursively merge O_1 with O_2 and E_1 with E_2 .

Example :- $n_1 = 8 \quad 12 \quad 21 \quad 25$
 $x_2 = 3 \quad 5 \quad 24 \quad 26$





∴ sorted are 18, 26, 25, 24, 21, 12, 8, 5, 3.

⇒ Sorted order data given example trace only last two steps. #

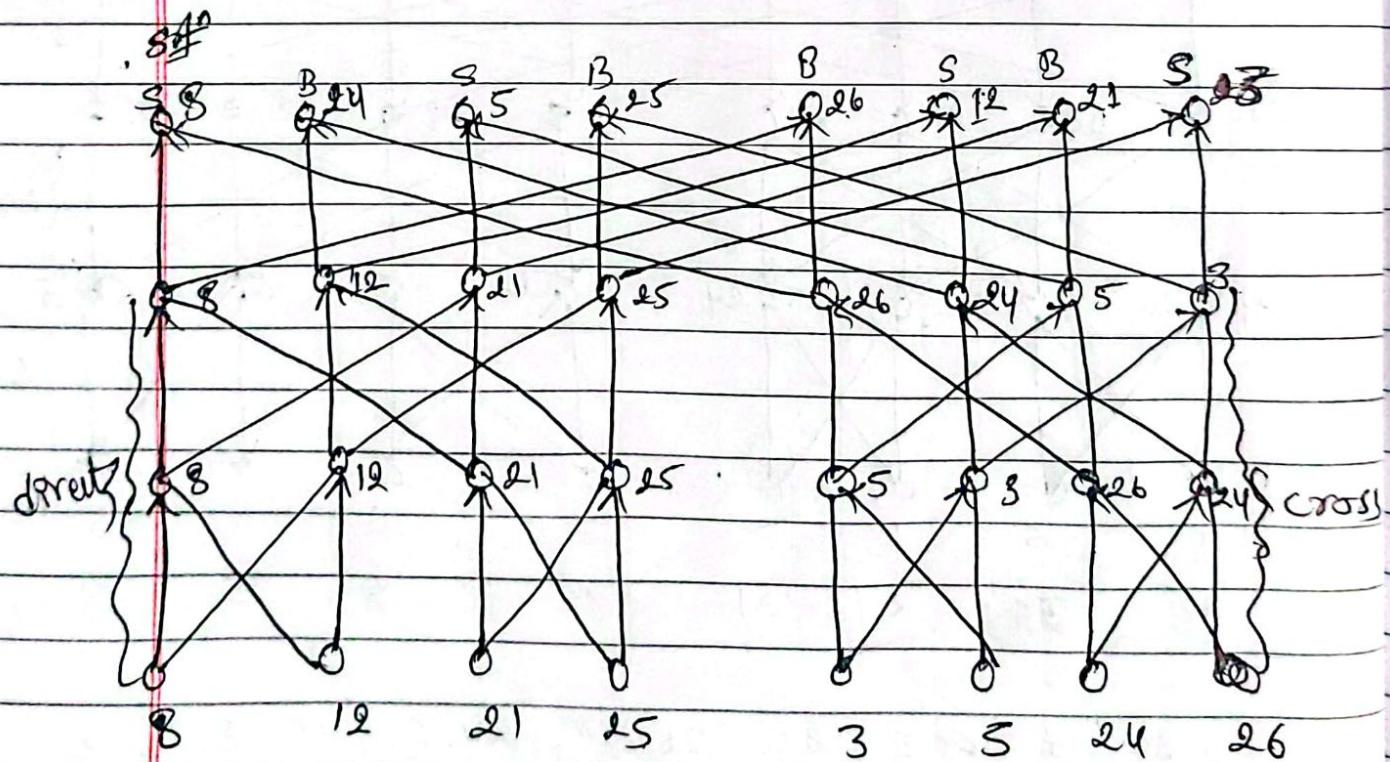
\Rightarrow Bitonic \rightarrow one side all big (large) value and other side all large value is called bitonic.
eg one side 26, 25, 21, 24 & other side 3, 8, 12, 5.

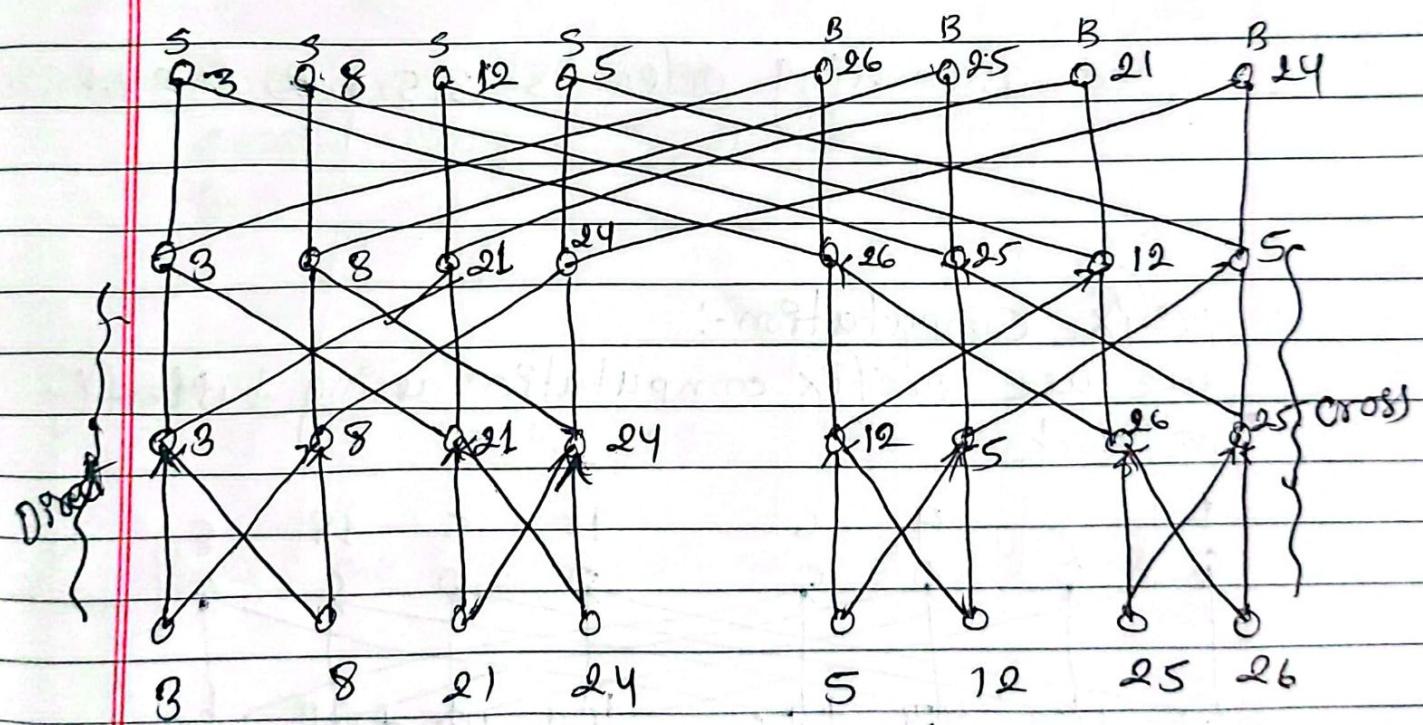
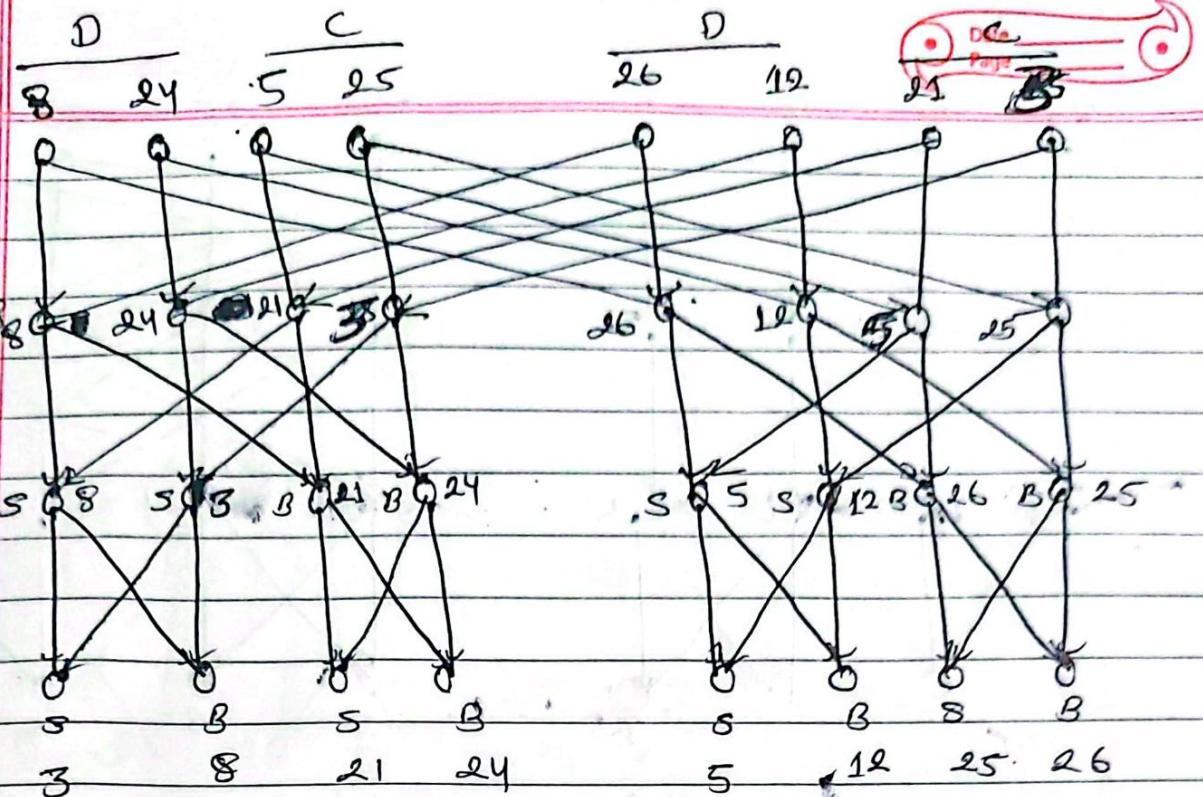
- If the value is in bitonic order then only used last sequence only.
- * (Both section one side big value & other side small value).

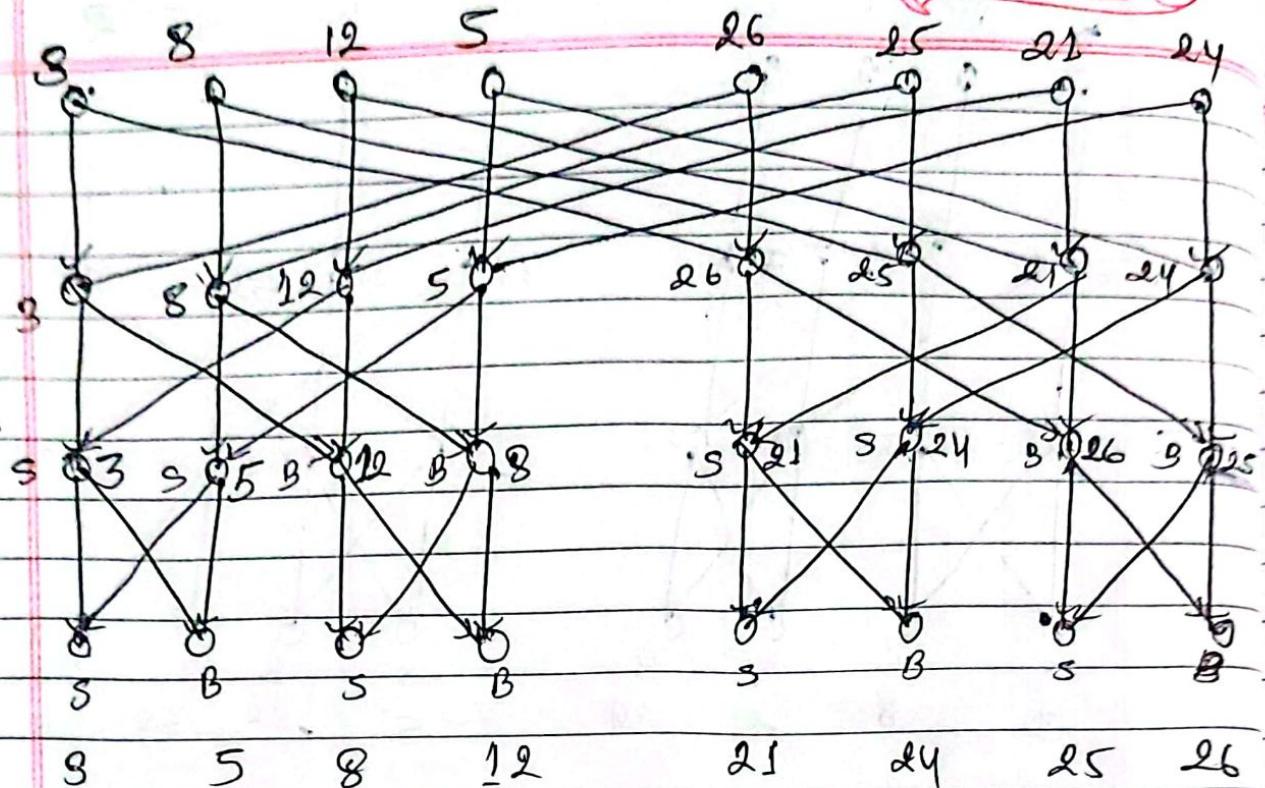
Example: Sort value with ascending order.

$$x_1 = 8, 12, 21, 25$$

$$x_2 = 3, 5, 24, 26$$



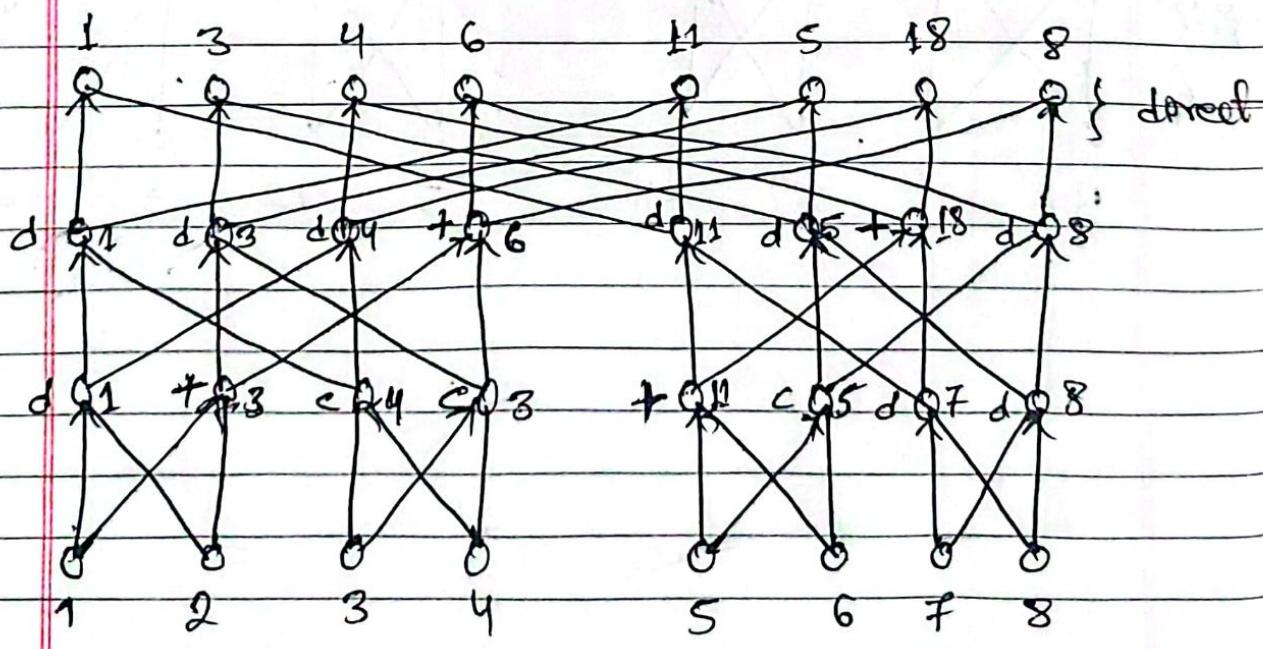


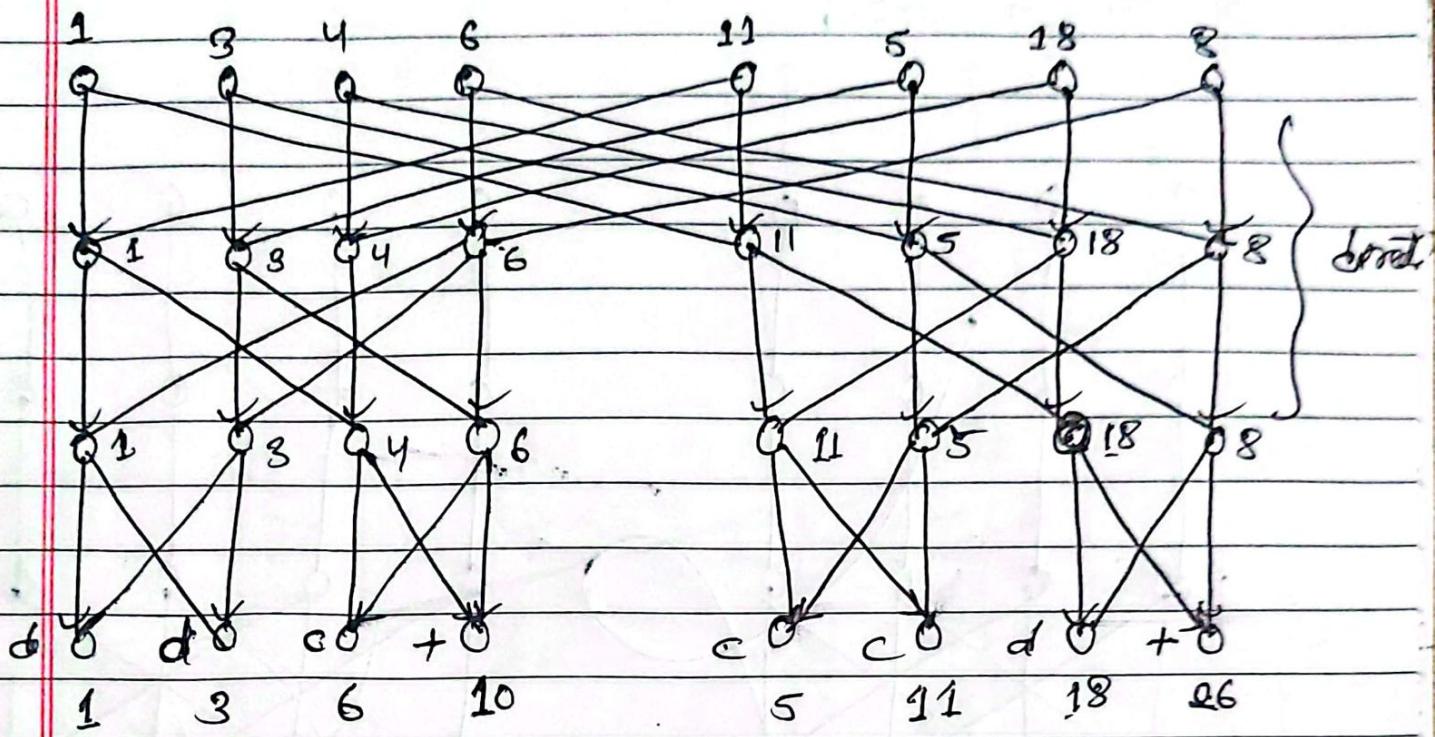


\therefore Sorted ascending order $\{8, 3, 5, 8, 12, 21, 24, 25, 26\}$

\Rightarrow prefix computation:-

we use prefix computation using butterfly network :

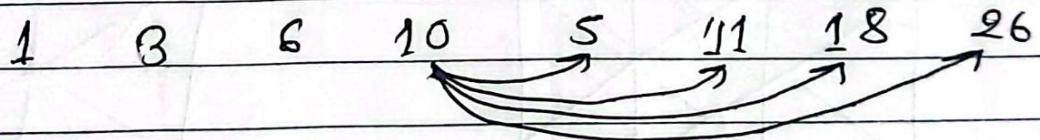




\Rightarrow first part = 1, 3, 6, 10

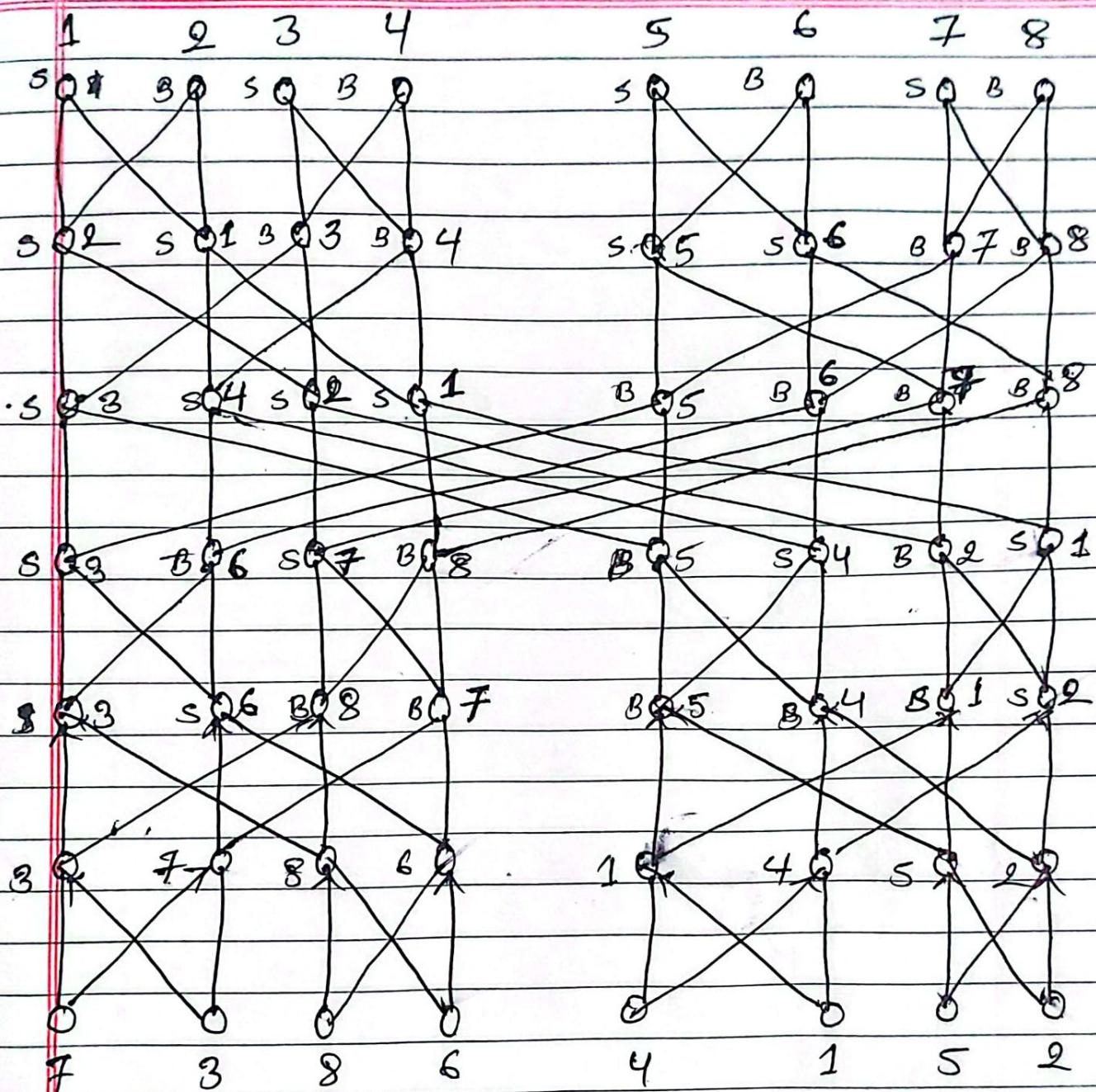
second part = 5, 11, 18, 26

then,



$\Rightarrow 1, 3, 6, 10, 15, 21, 28, 36$

Hence the prefix output ~~is~~ ~~is~~ 1, 3, 6, 10, 15, 21, 28, 36



⇒ Selection sort output = 1, 2, 3, 4, 5, 6, 7, 8