Greedy

left - small
Right - large

* Optimal binary Search tree :- 'OBST'

Seq.	1	2	3	4
Keys :	10	20	30	40
frequency :	4	2	6	3

} Succenful Search Probability

i \ j	0	1	2	3	4
0	0	4	8^1	20^3	26^3
1		0	2	10^3	16^3
2			0	6	12^3
3				0	3
4					0

Case : I $j - i = 0$

$0 - 0 = 0$	$C[0,0] = 0$
$1 - 1 = 0$	$C[1,1] = 0$
$2 - 2 = 0$	$C[2,2] = 0$
$3 - 3 = 0$	$C[3,3] = 0$
$4 - 4 = 0$	$C[4,4] = 0$

∴ Time Complexity - $O(n^3)$ — Three level of nested loop

Case: II $j-i=1$ [take one element]

$1-0=1$	$C[0,1]=4$
$2-1=1$	$C[1,2]=2$
$3-2=1$	$C[2,3]=6$
$4-3=1$	$C[3,4]=3$

Case: III $j-i=2$ [take two elem

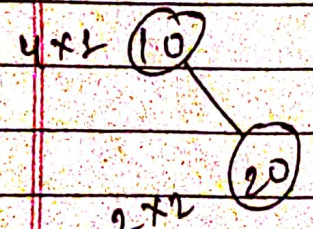
$2-0=2$	$C[0,2]=8$ $r=1$
$3-1=2$	$C[1,3]=10$ $r=3$
$4-2=2$	$C[2,4]=12$ $r=3$

$C[0,2]$

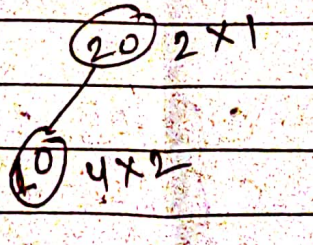
1 2 (Root)

10 20

4 2



$=4+4=8$



$=2+8=10$

without formula

Note: weight of $w(0,4)=\sum_{i=1}^{4} f(i)$ [formula]

For $w[0,2]=f_1+f_2$ [frequency]

$=4+2=6$

Now,

$$c[0,2] = \min \begin{cases} c[0,0] + c[1,2] + w[0,2] \\ c[0,1] + c[2,2] + w[0,2] \end{cases}$$

$$= \min \begin{cases} 0 + 2 + 6 = 8 \leftarrow \\ 4 + 0 + 6 = 10 \end{cases}$$

$$c[1,3] = \min \begin{cases} c[1,1] + c[2,3] + w[1,3] \\ c[1,2] + c[3,3] + w[1,3] \end{cases}$$

$$= \min \begin{cases} 0 + 6 + 8 = 14 \\ 2 + 0 + 8 = 10 \leftarrow \end{cases}$$

$$c[2,4] = \min \begin{cases} c[2,2] + c[3,4] + w[2,4] \\ c[2,3] + c[4,4] + w[2,4] \end{cases}$$

$$= \min \begin{cases} 0 + 3 + 9 = 12 \leftarrow \\ 6 + 0 + 9 = 15 \end{cases}$$

Case: IV $j - i = 3$

$3 - 0 = 3$
$4 - 1 = 3$

$C[0,3] = 20 \quad r = 3$
$C[1,4] = 16 \quad r = 3$

$$C[0,3] = \min \begin{cases} C[0,0] + C[1,3] + \omega[0,1] \\ C[0,1] + C[2,3] + \omega[0,2] \\ C[0,2] + C[3,3] + \omega[0,3] \end{cases}$$

$$= \min \begin{cases} 0 + 10 + 12 = 22 \\ 4 + 6 + 12 = 22 \\ 8 + 0 + 12 = 20 \leftarrow \end{cases}$$

$$C[1,4] = \min \begin{cases} C[1,1] + C[2,4] + \omega[1 \\ C[1,2] + C[3,4] + \omega[1 \\ C[1,3] + C[4,4] + \omega[1 \end{cases}$$

$$= \min \begin{cases} 0 + 12 + 11 = 23 \\ 2 + 3 + 11 = 16 \leftarrow \\ 10 + 0 + 11 = 21 \end{cases}$$

Case: V $j-i=4$

$4-0=4$ | $C[0,4] = 26$ $k=3$

$$C[0,4] = \min \begin{cases} C[0,0] + C[1,4] + w[0,4] \\ C[0,1] + C[2,4] + w[0,4] \\ C[0,2] + C[3,4] + w[0,4] \\ C[0,3] + C[4,4] + w[0,4] \end{cases}$$

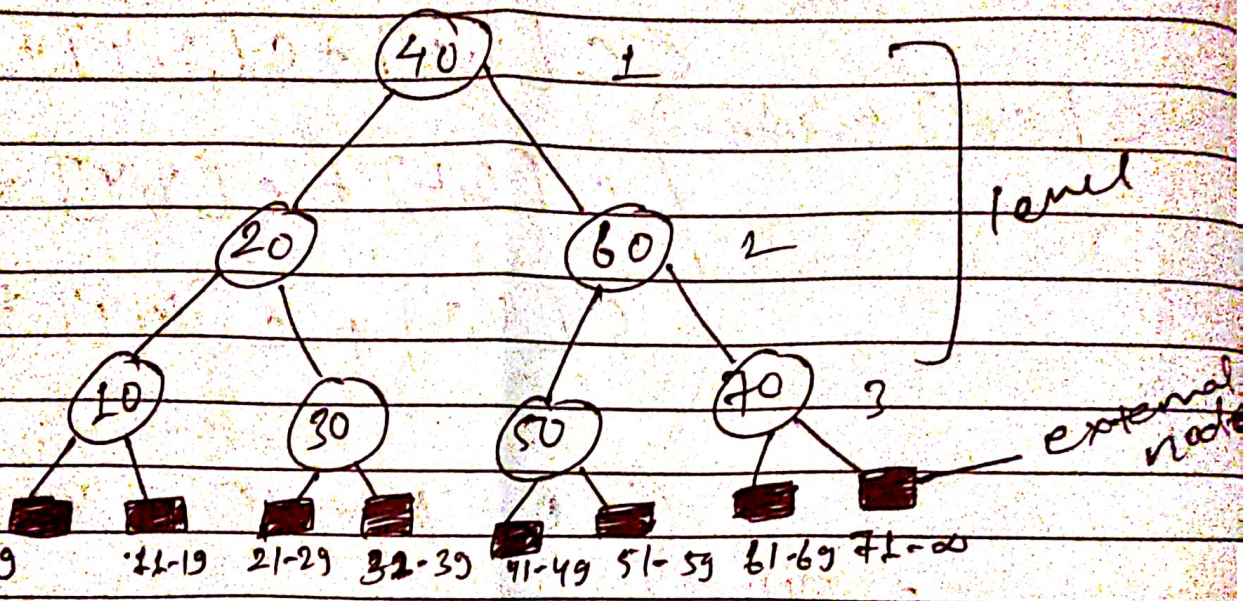
$$= \min \begin{cases} 0 + 16 + 15 = 31 \\ 4 + 12 + 15 = 31 \\ 8 + 3 + 15 = 26 \\ 20 + 0 + 15 = 35 \end{cases}$$

Formula:

$$C[i,j] = \min_{i<k\le j} \{ C[i,k-1] + C[k,j] \} + w[i,j]$$

c) Succesful and unsuccesful Search probability :- OBST / dynamic programming approach

$$T(n) = \frac{^{2n}C_n}{n+1}, \quad n = \text{no. of node}$$



d) No. of Comparisons required in binary tree = height of a binary tree.

e) Search may be succesful or unsuccesful

f) The idea is for the same set of keys when you can have multiple binary search trees one of the binary tree may be giving you best results that is minimum no. of Comparisons.

Eg:

	1	2	3	4
Keys	10	20	30	40
p_i :	0.1	0.2	0.1	0.2
q_i :	0.1	0.05	0.15	0.05	0.05
	q_0	q_1	q_2	q_3	q_4

$$c[0,4] = \sum_{i=1}^{n} p_i * \text{level}(a_i) + \sum_{i=0}^{n} q_i * (\text{level}(e_i) - 1)$$

, e_i = external nodes
a_i = keys

p = probability of succes
q = probability of unsucces

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

$w[0,2] = q_0 + p_1 + q_1 + p_2 + q_2$

$w[0,3] = q_0 + p_1 + q_1 + p_2 + q_2 + p_3 + q_3$

So, generalized this, we get the formula,

$$w[\underset{i}{0},\underset{j}{3}] = w[0,2] + p_j + q_j$$

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

Now find, cost, weight and keys (root) of the binary tree.

$$c[i,j] = \min_{i<k\le j} \{ c[i,k-1] + c[k,j] \} + w(i,j)$$

Eg.

	0	1	2	3	4
keys		10	20	30	40
p_i		3	2	1	1
q_i	2	3	1	1	1

	0	1	2	3	4
$j-i=0$	$w_{00}=2$ $c_{00}=0$ $r_{00}=0$	$w_{11}=3$ $c_{11}=0$ $r_{11}=0$	$w_{22}=1$ $c_{22}=0$ $r_{22}=0$	$w_{33}=1$ $c_{33}=0$ $r_{33}=0$	$w_{44}=1$ $c_{44}=0$ $r_{44}=0$
$j-i=1$	$w_{01}=8$ $c_{01}=8$ $r_{01}=1$	$w_{12}=7$ $c_{12}=7$ $r_{12}=2$	$w_{23}=3$ $c_{23}=3$ $r_{23}=3$	$w_{34}=3$ $c_{34}=3$ $r_{34}=4$	
$j-i=2$	$w_{02}=12$ $c_{02}=19$ $r_{02}=1$	$w_{13}=9$ $c_{13}=12$ $r_{13}=2$	$w_{24}=5$ $c_{24}=8$ $r_{24}=3$		
$j-i=3$	$w_{03}=14$ $c_{03}=25$ $r_{03}=2$	$w_{14}=11$ $c_{14}=19$ $r_{14}=2$			
$j-i=4$	$w_{04}=16$ $c_{04}=32$ $r_{04}=2$				

At first fill $j-i=0$ table, then fill all the w values by using formula.

$$c[0,1] = \min\{c[0,0] + c[1,1]\} + w[0,1]$$

$$= 0 + 0 + 8 = 8$$

$$c[1,2] = \min\{c[1,1] + c[2,2]\} + w[1,2]$$

$$= 0 + 0 + 7 = 7$$

Similarly, for

$$c[2,3] = w[2,3] = 3$$

$$c[3,4] = w[3,4] = 3$$

Now,

$$c[0,2] = \min\begin{Bmatrix} c[0,0] + c[1,2] \\ c[0,1] + c[2,2] \end{Bmatrix} + w[0,2]$$

$$= \min\begin{Bmatrix} 0+7 \\ 8+0 \end{Bmatrix} + 12$$

$$= 7 + 12 = 19$$

$$c[1,3] = \min\begin{Bmatrix} c[1,1] + c[2,3] \\ c[1,2] + c[3,3] \end{Bmatrix} + w[1,3]$$

$$= \min\{0+3\} + 9 = 12$$

$$c[2,4] = \min \begin{Bmatrix} c[2,2] + c[3,4] \\ c[2,3] + c[4,4] \end{Bmatrix} + w[2,4]$$

$$= \min \begin{Bmatrix} 0+3 \\ 3+0 \end{Bmatrix} + 5$$

$$= 8 \quad \text{(both are equal)}$$

Again,

$$c[0,3] = \min \begin{Bmatrix} c[0,0] + c[1,3] \\ c[0,1] + c[2,3] \\ c[0,2] + c[3,3] \end{Bmatrix} + w[0,3]$$

$$= \min \begin{Bmatrix} 0+12 \\ 8+3 \\ 19+0 \end{Bmatrix} + 14$$

$$= 25$$

$$c[1,4] = \min \begin{Bmatrix} c[1,1] + c[2,4] \\ c[1,2] + c[3,4] \\ c[1,3] + c[4,4] \end{Bmatrix} + w[1,4]$$

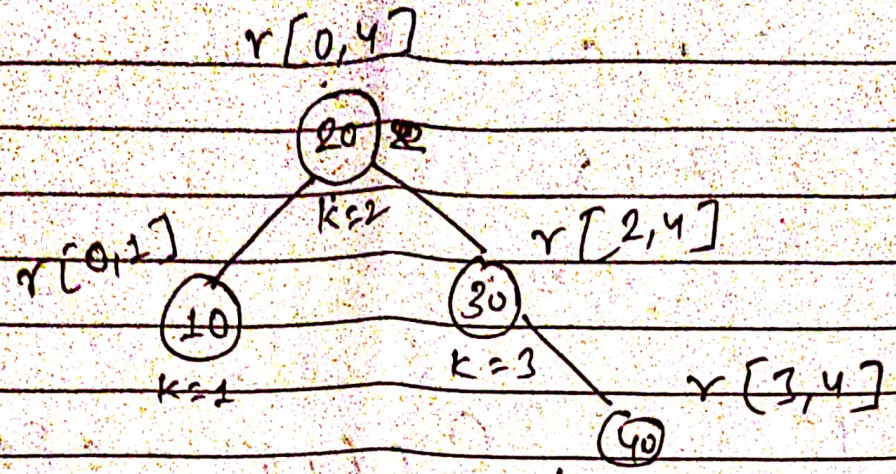
$$= \min \begin{Bmatrix} 0+8 \\ 7+3 \\ 12+0 \end{Bmatrix} + 11$$

$$= 19$$

$$c[0,4] = \min \begin{Bmatrix} c[0,0] + c[1,4] \\ c[0,1] + c[2,4] \\ c[0,2] + c[3,4] \\ c[0,3] + c[4,4] \end{Bmatrix} + w[0$$

$$= \min \begin{Bmatrix} 0+19 \\ 8+8 \\ 19+3 \\ 25+0 \end{Bmatrix} + 16$$

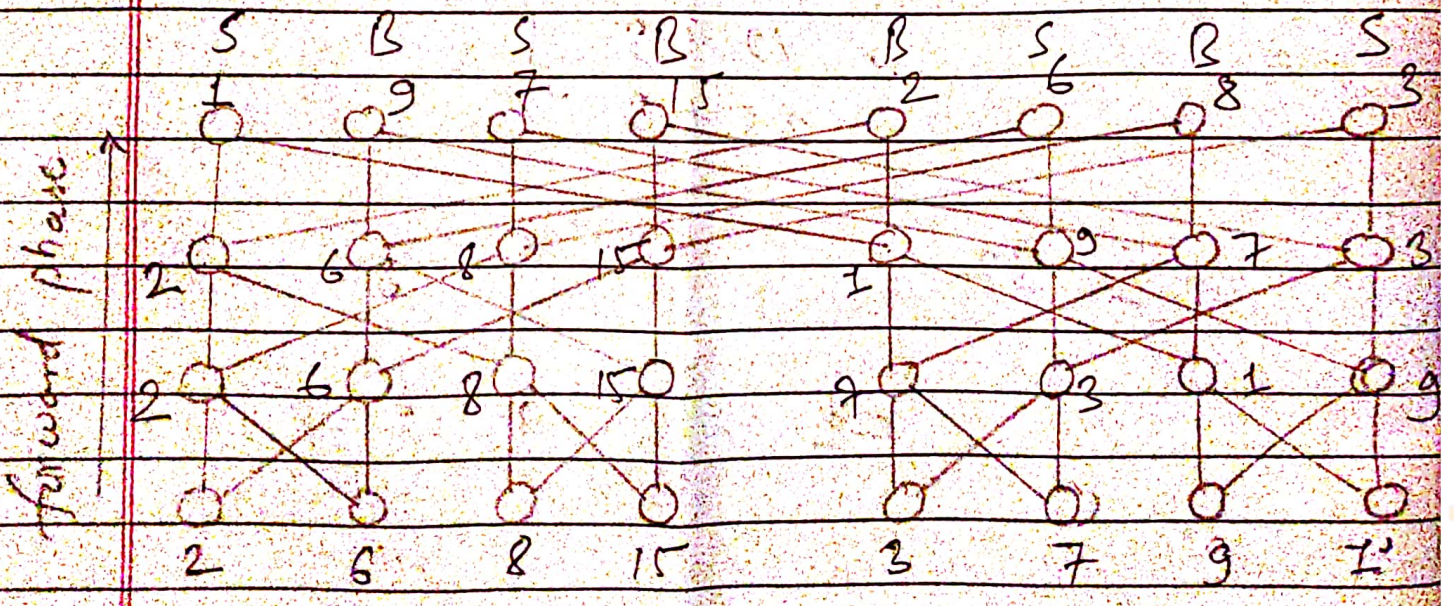
$$= 32$$



This is the optimal binary tree.

✱ Butterfly Network:

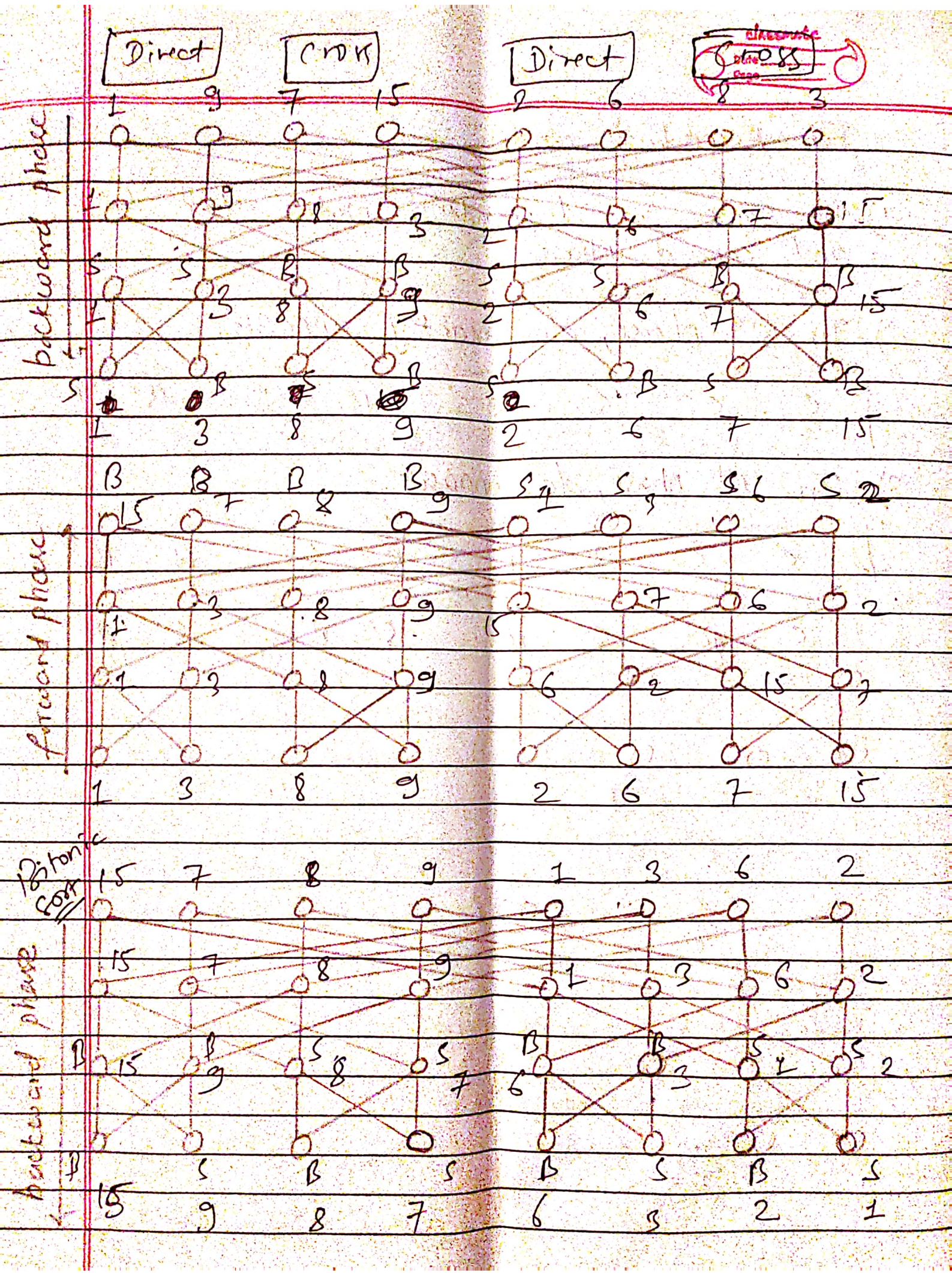
— Let $X_1 = 2,6,8,15$ and $X_2 = 3,7,9,1$. Perform odd even merge sort in a Butterfly Network



$(d+1)2^d$ → no. of process (32)

$d \cdot 2^{d+1}$ → no. of links (48)

$d = 3$


Direct
Cross
Direct
Cross
1 9 7 15
2 6 8 3
backward phase
forward phase
Bitonic
Comp
ascending phase
descending phase

For Ascending order;

1) Do same as descending order for first and second phase.

2) In 3rd phase, do exactly opposite,

↑ S S S S B B B B

3) In 4th phase, do opposite,

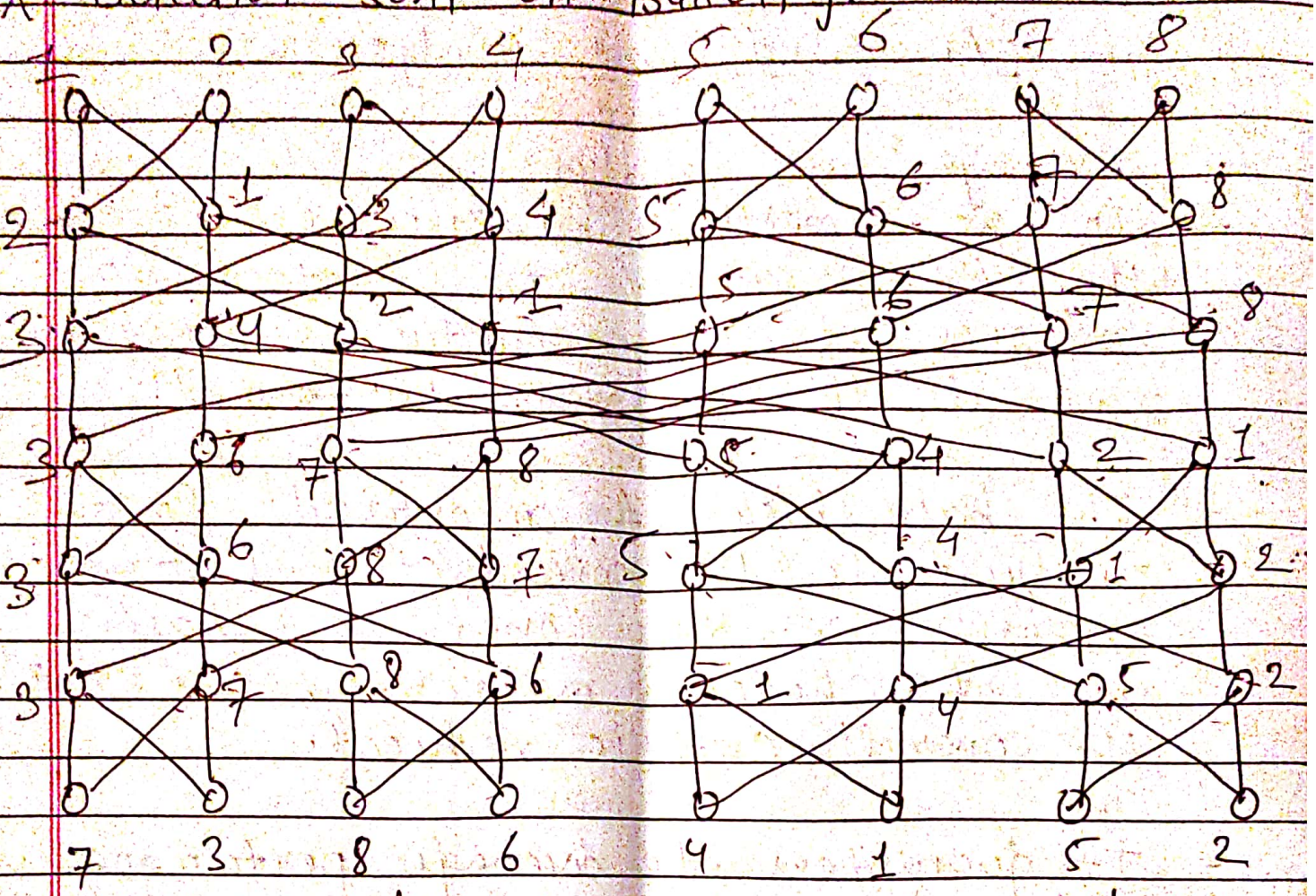
↓ S S B B S S B B

S B S B S B S B

1, 2, 3, 4, 5, 6, 7, 8

classmate
Date
Page

* Selection Sort on Butterfly:



"Amortized Cost Analysis"

- Average Cost analysis
- In an amortized analysis, we average the time required to perform a sequence of data-structure operations over all the operations performed.
- The basic idea of amortized Cost analysis is to spread the cost of expensive operations over a larger no. of cheaper operations, so that the overall cost is reduced.
- It guarantees the average performance of each operation in the worst case.
- Amortized Cost analysis is often used to analyze the time complexity of data structu like dynamic arrays (e.g. ArrayList in Java) and dynamic tables (e.g. hash tables).

There are three common methods used for amortized Cost analysis:

(i) Aggregate method
(ii) Accounting method
(iii) Potential method

(i) Aggregate :→ It involves simply computing the total cost of all the operations and dividing by the number of operations to obtain the average cost.

(ii) Accounting :→ When an operation's amortized cost exceeds it's actual cost, the difference is assigned to specific objects in the data structure as credit. Credit can be used later on to help pay for operations whose amortized cost is less than their actual cost.

One must choose the amortized costs of operations carefully. The total credit in the data structure should never become negative.

(iii) Potential :→ This method involves defining a potential function that associates a potential value with the state of the data structure after each operation.

The amortized cost of an operation is the actual cost of the operation plus the change in potential energy.

potential function.

$$\hat{c}_i = c_i + [\Phi(D_i) - \Phi(D_{i-1})] \quad \{ \geq 0 \text{ for all } i$$

$\hat{c}_i$: amortized cost

c_i : actual cost

$\Phi(D_i) - \Phi(D_{i-1})$: Change in potential

* Aggregate Method:

Example; Hash Table.

1	insert →	1	Copy →	1	Copy →	1	Copy →	1
2	insert →			2	Copy →	2	Copy →	2
3	insert →					3	Copy →	3
4	insert →					4	Copy →	4
5	insert →							5
6	insert →							6
7	insert →							7
8	insert →							8

i	1	2	3	4	5	6	7	8	9	10
size i	1	2	4	4	8	8	8	8	16	16
Actual Cost Ci	1	2	3	1	5	1	1	1	9	1

Amortized Cost per operation :

$$\frac{\text{Total Cost for all operations}}{\text{Number of operations.}}$$

C_i := Amortized Cost

C_i = [(1+1+1+1+1+...) + (1+2+4+...)]

$\therefore$ Cost of i^{th} insertion = $\begin{cases} i, & \text{when } (i-1) \text{ is an exact power of 2} \\ 1, & \text{otherwise} \end{cases}$

$$C_i = \frac{\left[\underbrace{(1+1+1+1+1+\ldots)}_{n} + (1+2+4+\ldots)\right]}{n}$$

$$= \frac{n + \sum_{j=0}^{j=\lfloor \log_2(n-1) \rfloor} 2^j}{n}$$

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

$$= 3n - 3, \text{ which is } O(n).$$

Let $\hat{C_i}$ be an amortize Cost

$$\hat{C_i} = O(n) = \frac{O(n)}{n} = \frac{n}{n} = 1$$

Amortized Cost is Constant. i.e., $O(1)$.

* Accounting Method:

Considering the cost is added or reduced from the bank. And Bank ≥ 0.

Let $\hat{C_i}$ be the amortized Cost and c_i be the actual cost.

S.t. $\hat{C_i} \geq c_i$

Let $\hat{c_i} = 3$, Considering every 1 operation Cost as 3, we get the tabulated as below:

c_i	1	2	3	1	5	1	1	1	9	1
$\hat{c_i}$	3	3	3	3	3	3	3	3	3	3
Bank	2	3	3	5	3	5	7	9	3	5

If every 1 operation cost is 3.
Then n operation cost is 3n.

Therefore,

$$O(n) = \frac{3n}{n} = 3$$

Hence, $O(n) = 1$, which is Constant.

* Potential method:

Suppose D_i be the data structure. c_i be the actual cost and $\hat{c_i}$ be the amortized cost.
we define a potential function $\phi(D_i)$

$\phi(D_i) = 2i - 2^{\lceil \log i \rceil}$ Remember

We know,

Amortized Cost = Actual Cost + Change in potential

$\hat{c_i} = c_i + [\phi(D_i) - \phi(D_{i-1})]$

≥ 0 for all i

$$c_i = \begin{cases} i, & \text{when } (i-1) \text{ is an exact power of 2} \\ 1 & \text{otherwise} \end{cases}$$

Case - I

$\hat{c_i} = i + [\phi(D_i) - \phi(D_{i-1})]$

$= i + 2i - 2^{\lceil \log i \rceil} - [2(i-1) - 2^{\lceil \log (i-1) \rceil}]$

$= i + 2i - 2^{\lceil \log i \rceil} - 2i + 2 + 2^{\lceil \log (i-1) \rceil}$

$= i + 2 - 2^{\lceil \log i \rceil} + 2^{\lceil \log (i-1) \rceil}$

$= i + 2 - 2(i-1) + 2(i-1)$

$= i + 2 - (i-1)$

$= i + 2 - i + 1 = 3$ ∎

$2^{\lceil \log(i-1) \rceil} = 2^{\lceil \log 7 \rceil}$

$= 2^3 = 8$

Rough work

4) Eg. $i = 9$

$(i-1) = 8$

$2^{\lceil \log i \rceil} = 2^{\lceil \log_2 9 \rceil} = 2^4 = 16 \Rightarrow 2(i-1)$

$2^{\lceil \log(i-1) \rceil} = 2^{\lceil \log_2 8 \rceil} = 2^3 = 8 \Rightarrow (i-1)$

Case - II

$\hat{c_i} = 1 + [\phi(\mathcal{D}i) - \phi(\mathcal{D}i-1)]$

$= 1 + 2i - 2^{\lceil \log i \rceil} - [2(i-1) - 2^{\lceil \log(i-1) \rceil}]$

$= 1 + 2i - 2^{\lceil \log i \rceil} - 2i + 2 + 2^{\lceil \log(i-1) \rceil}$

$= 1 + 2 - 2^{\lceil \log i \rceil} + 2^{\lceil \log(i-1) \rceil}$

When $(i-1)$ is not an enact power of 2,

$2^{\lceil \log i \rceil} = 2^{\lceil \log(i-1) \rceil}$

$= 1 + 2 = 3$

AC

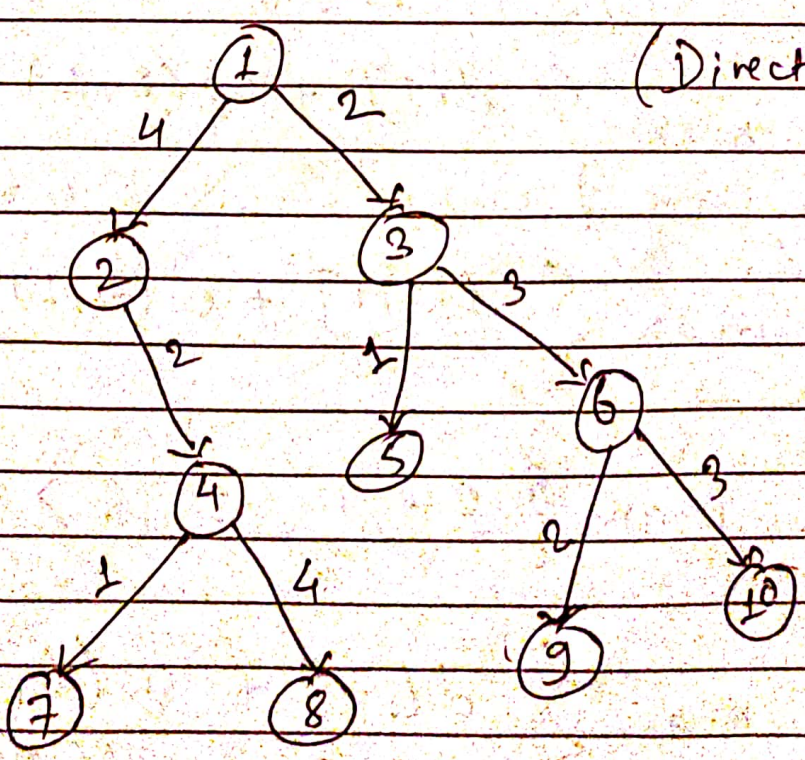
Numericals

* Greedy Algorithms:

(i) Tree vertex splitting
(ii) Job sequencing with deadlines

(i) Tree vertex splitting:

Let, $T = (V, E, W)$ and δ as tolerance,
V = vertex
E = Edge
W = Weight



(Directed Binary Tree)

Fig: Directed Graph

For all leaf nodes, delay = 0

$\therefore d[5], d[7], d[8], d[9], d[10] = 0$

We have, $d[v]$ & Suppose $\delta = 5$, then

$$d[4] = \max \begin{cases} d[4] + w(2,4) \\ d[4] + w(2,4) \end{cases}$$

$$= \max \begin{cases} d(4,7) + w(2,4) \\ d(4,8) + w(2,4) \end{cases} \} \delta \text{ split}$$

$$= \max \begin{cases} 1 + 2 = 3 \\ 4 + 2 = 6 \end{cases} = 6 > \delta \text{ (split)}$$

$$d[2] = \max \begin{cases} d[2] + w(2,1) \\ \end{cases}$$

$$= \max \begin{cases} d[2,4] + w(2,1) \\ \end{cases}$$

$$= \max \{ 2 + 4 = 6 > \delta \text{ (split)}$$

$$d[6] = \max \begin{cases} d[6] + w(6,3) \\ d[6] + w(6,3) \end{cases}$$

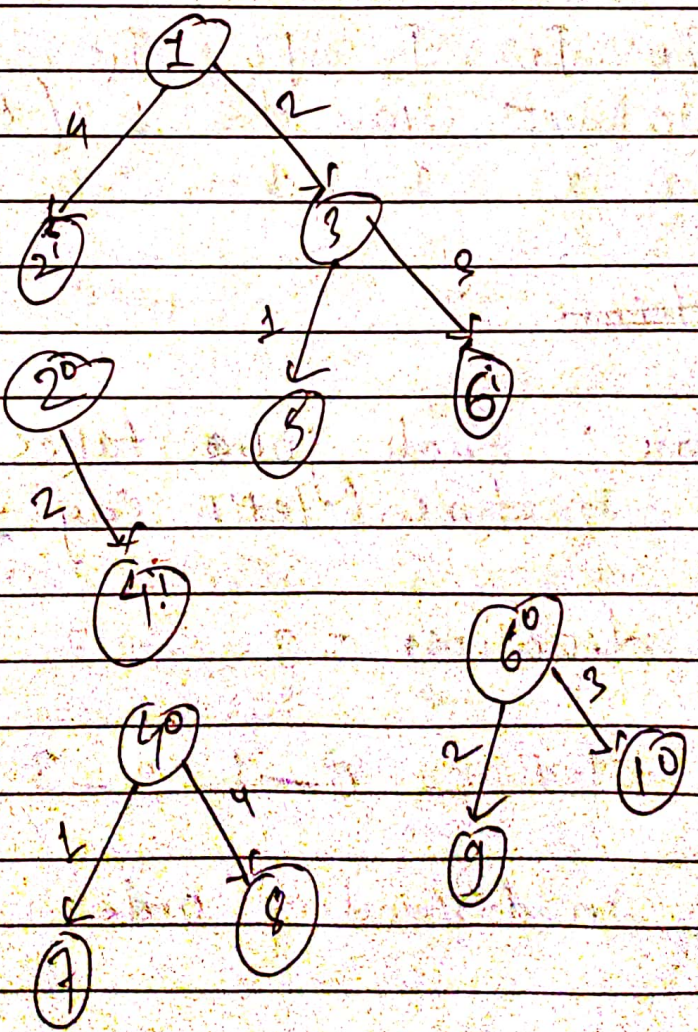
$$= \max \begin{cases} d[6,9] + w(6,3) \\ d[6,10] + w(6,7) \end{cases}$$

$$= \max \begin{cases} 2+3 \\ 3+3 \end{cases} = 6 > 5 \quad (\text{Split})$$

$$d[3] = \max \begin{cases} d[3] + w(3,7) \\ d[3] + w(3,1) \end{cases}$$

$$= \max \begin{cases} d[3,5] + w(3,1) \\ d[3,6] + w(3,1) \end{cases}$$

$$= \max \begin{cases} 1+2 = 5 = 5 \quad (\text{don't split}) \\ 3+2 \end{cases}$$



Analysis:

Visit each vertex — n

In each vertex there is constant time required for calculating and comparing it with δ.

∴ O(n) = n.

(ii) Job sequencing with deadlines:

n = 5

Ji	Jobs	J1	J2	J3	J4	J5
Pi	profits	20	15	10	5	1
di	deadlines	2	2	1	3	3

(Already sorted)

~~Each machines~~

Assumptions: Each jobs takes one unit of time (let's say 1 hr.)

Here,
maximum deadlines = 3

Available slot : 0 – 1 – 2 – 3.

Here, Ji in descending order w.r.to Pi.

	Job Consider	Slot assign	Solution	Profit
	J_1	[1,2]	J_1	20
	J_2	[0,1] [1,2]	J_1, J_2	20+15
X	J_3	[0,1] [1,2]	J_1, J_2	20+15
	J_4	[0,1] [1,2] [2,3]	J_1, J_2, J_4	20+15+5 = 40

Analysis:

Case: I

Each time slot calculated is vacant. So, just insert is required, which is required to the no. of iterations.

$\therefore O(n) = n.$

Case: II

Time slot calculated is not vacant. So, requires another iteration to find the vacant slot.

$O(n) = n^2.$

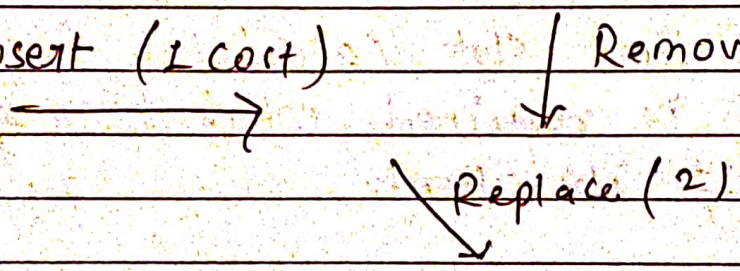
* Dynamic Programming: 'Useful for optimization problem'

(i) String Editing
(ii) Optimal BST

(i) String Editing (Minimum edit distance)

Problems: What is the minimum number of edit operations (i.e. change, delete, insert) to change one string to another?

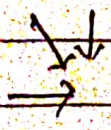
The problem can be solved by using dynamic programming.



(i) If row == column

Copy the diagonal

(ii) If row =/= column



minimum + 1

	Null	a	b	c	f	g
Null	0 →	1 →	2 →	3 →	4 →	5
a	1	0 →	1 →	2 →	3 →	4
d	2	1	1 →	2 →	3 →	4
c	3	2	2	1 →	2 →	3
e	4	3	3	2	2 →	3
g	5	4	4	3	3	2

abcfg → adceg

* Backtracking:

(i) Knapsack Problem
(ii) Sum of Subsets

(i) Knapsack Problem: 'Fraction'

n = 7
m = 15

Objects	1	2	3	4	5	6	7
Profit	10	5	15	7	6	18	3
weight	2	3	5	7	1	4	1

$\left(\frac{P}{W}\right)$

5	1.3	3	1	6	4.5	3
x_1	x_2	x_3	x_4	x_5	x_6	x_7
2nd	(2/3) fraction	4th	X	1st	3rd	5th

$\frac{\text{profit}}{\text{weight}}$

15 - 1 = 14
14 - 2 = 12
12 - 4 = 8
8 - 5 = 3
3 - 1 = 2
2 - 2 = 0

$\sum x_i w_i = 1\times2 + \frac{2}{3}\times3 + 1\times5 + 0\times7 + 1\times1 + 1\times4 + 1\times1$

$= 2 + 2 + 5 + 1 + 4 + 1 = 15$

$\sum x_i p_i = 1\times10 + \frac{2}{3}\times5 + 1\times15 + 0\times7 + 1\times6 + 1\times18 + 1\times3 = 54.6$

0/1 Knapsack problem:

(Capacity) m = 8 $p = \{1, 2, 5, 6\}$ Profits
$n = 4$ $w = \{2, 3, 4, 5\}$ weights

$f_i w_i$ → Weight

	0	1	2	3	4	5	6	7	8
W 0	0	0	0	0	0	0	0	0	0
1 2	0	0	1	1	1	1	1	1	1
2 3	0	0	1	2	2	3	3	3	3
3 4	0	0	1	2	5	5	6	7	7
4 5	0	0	1	2	5	6	6	7	(8)

$x_1\ x_2\ x_3\ x_4$
$0\ 1\ 0\ 1$

$8 - 6 = 2$
$2 - 2 = 0$
Go from last object

Time Complexity:

$O(nW)$

Where, n = no. of items
W = Capacity of knapsack

(ii) Sum of Subsets :

- Backtracking is a general algorithmic technique that involves exploring all possible solutions to a problem by incrementally building a solution and then undoing certain steps if they lead to a dead end.

- The Subset Sum problem is a classical Computational problem where the goal is to determine if there exists a subset of a given set of numbers that adds up to a specific target sum.

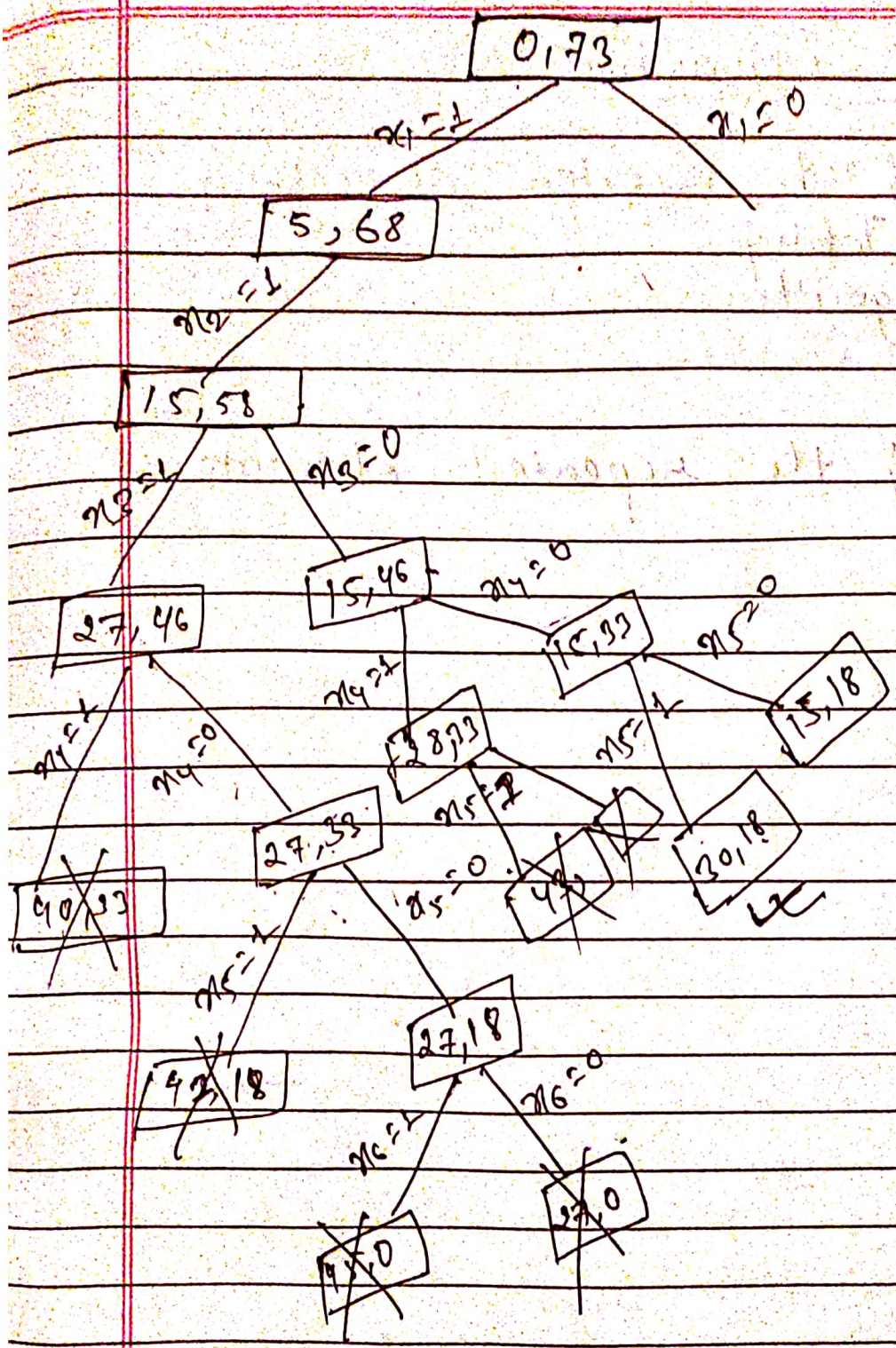
E.g: $m = 30$ ⇒ Sum of Subsets
$n = 6$

$$w = \{5, 10, 12, 13, 15, 18\} = 73$$
$$x_1 \; x_2 \; x_3 \; x_4 \; x_5 \; x_6$$

$x_1\ x_2\ x_3\ x_4\ x_5\ x_6$ = 73

SOS: 30

(depth first search)



∴ Time Complexity $O(2^n)$

(ii) Primality Testing:

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

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

    prime

else

    not prime

where,

n = no. to be checked

a < n (+ve numbers)

Let n = 2

a = 1

$1^{2-1} \% 2 = 1 \% 2 = 1$ ∴ (prime)

Let n = 3;

a = 1, 2

$1^{3-1} \% 3$ $\quad$ $2^{3-1} \% 3$

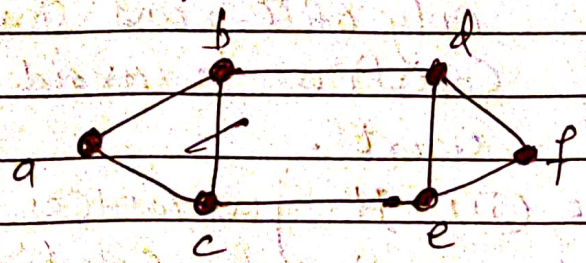
$= 1 \% 3$ $\quad$ $= 4 \% 3$

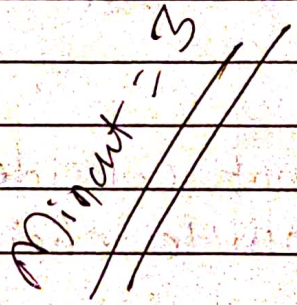
$= 1$ $\quad$ $= 1$ (Prime)

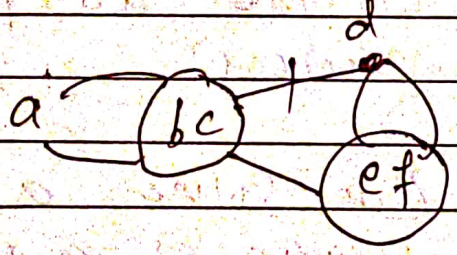
(iii) Kargers algorithm:

- It is a randomized algorithm used to find the minimum cut in a graph. It works by repeatedly Contracting random edges of the graph until only two nodes remain. The edges between those two nodes represent the minimum cut.

Eg







$V_1 \cap V_2 = \phi$

Time Complexity = $O(V^2)$

$\therefore V$ = no. of vertices

* Greedy vs Dynamic:

Greedy	Dynamic
(i) It involves making locally optimal choices at each step with the hope that it leads to a globally optimal solution.	(i) It involves breaking down a problem into smaller sub problems, solving each subproblem only once, and stores the solutions to avoid redundant work.
(ii) It's fast and simple but may not always yield the best result.	(ii) It guarantees an optimal solution but can be more complex and time consuming.
(iii) May not consider all possible choices.	(iii) Consider all possible choices.
(iv) Works well when the problem has the greedy choice property.	(iv) Can solve a wide range of problems.
(v) E.g: Huffman Coding, Fractional Knapsack.	(v) E.g. Fibonacci sequence, 0/1 knapsack

* List Ranking: 'n processor environment'

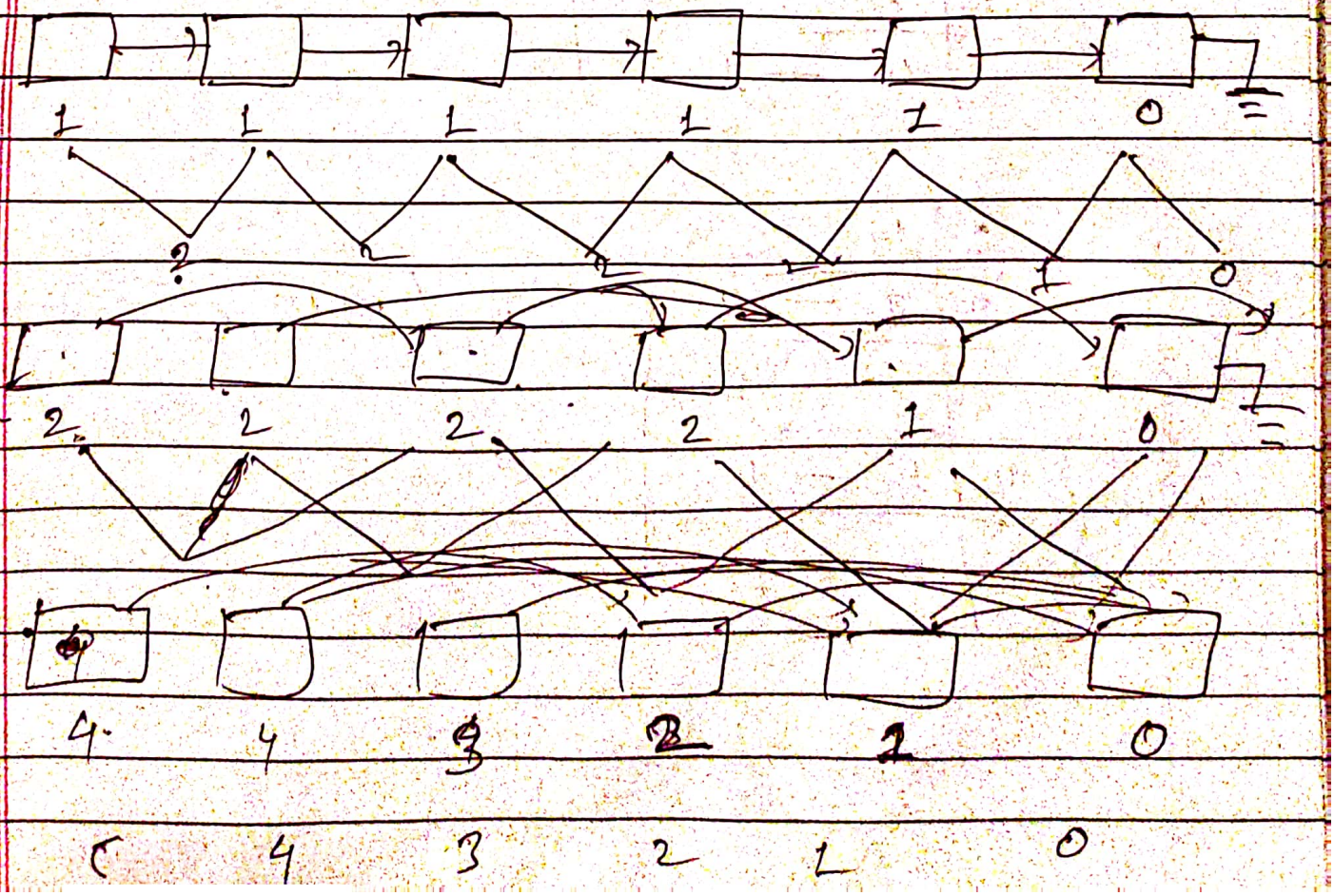
- Determining the position or rank of each item in a linked list.

Initialization

```
For (each processor i):
  if (next[i] == 0)
      Rank[i] = 0
  else
      Rank[i] = 1
```

For finding Rank:

```
While (next[i] != NULL):
For (each processor i)
  if (next[i] != NULL)
      next[i] = next[i] +
                next(next[i])
      Rank[i] = Rank[i] +
                Rank(next[i])
```



PRAM

Selection : (Max Element) 'n-processor'

E.g. 8, 12, 14, 15

2	8
2	4 — 0
2	2 — 0
2	1 — 0
	0 — 1

$= (1000)_2$

2	12
2	6 — 0
2	3 — 0
2	1 — 1
	0 — 1

$= (1100)_2$

2	14
2	7 — 0
2	3 — 1
2	1 — 1
	0 — 1

$= (1110)_2$

2	15	
2	7	— 1
2	3	— 1
2	1	— 1
	0	— 1

$= (1111)_2$

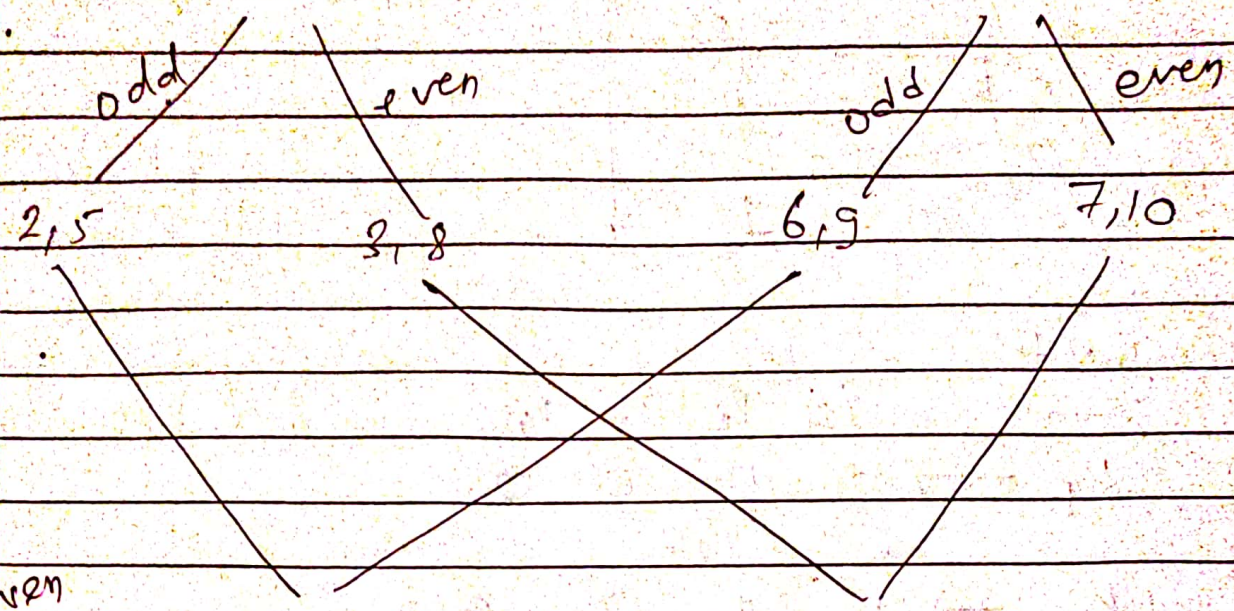
Input

	P_1	P_2	P_3	P_4	
$K_1 \rightarrow$	1	0	0	0	
$K_2 \rightarrow$	1	1	0	0	
$K_3 \rightarrow$	1	1	1	0	
$K_4 \rightarrow$	1	1	1	1	$\therefore K_4 = \max$ element $= 15$
	no elimination	K_1	K_1, K_2	K_1, K_2, K_3	

Sorted arrays.

* Odd Even Merge Sort: (w.r.to position)

x_1 : 2 3 5 8 $\quad$ x_2 : 6 7 9 10



odd even merge: 2,5,6,9 $\quad$ 3,8,7,10

Shuffling: [2], [3,5], [8,6], [7,9], [10] ↺

If required interchange,

2,3,5,6,~~7~~,~~8~~,9,10 → Sorted

* Mesh Algorithm:-

A mesh is an $a \times b$ grid in which there is procesor at each grid point. The edges correspond to communication links and are bidirectional. Each procesor ~~can perform any~~ of the mesh can be labeled with a tuple (i,j) where,

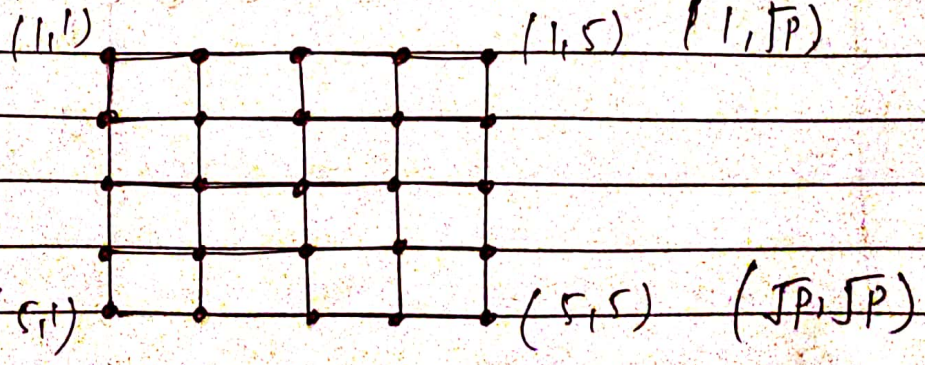
$1 \le i \le a$
$1 \le j \le b$.

Each procesn has some memory & can perform basic operations. $(+, -, \times, *)$

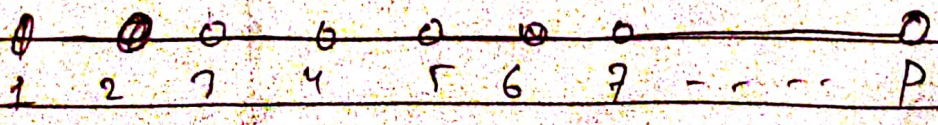
We Concider Square meshes. (ie. $a = b$)

A $\sqrt{p} \times \sqrt{p}$ mesh is shown in the figure

A linear array consicts of p procesor.



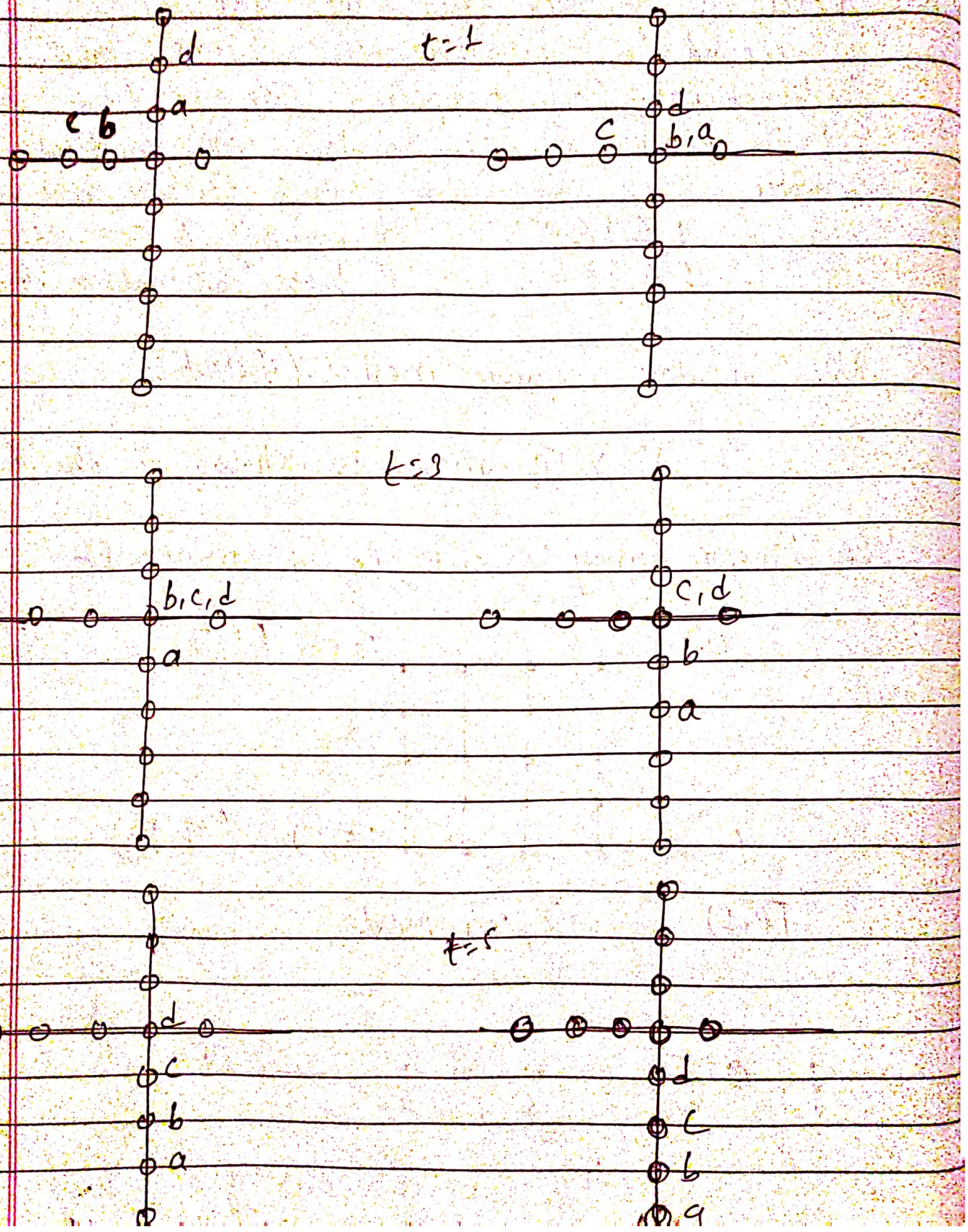
fig(a) Mesh



fig(b) Linear array

Q.9)

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



This Concept is known as packet routing.
(Inter processor Communication operation).
Since it needs (P-1) steps to complete the linear array, the worst case of linear array is
O(P-1).

- A packet Contains:

data + Source processor + destination processor

- A link Can handle only one packet at one unit time.

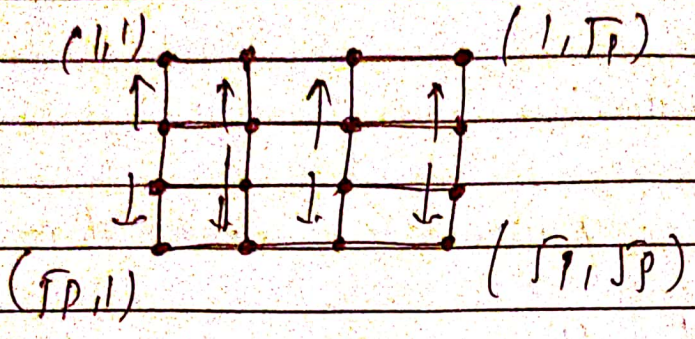
- A processor may receive multiple packets (from different links) and send multiple packets (to different links) at the same time

- Partial Permutation Routing (PPR) is a Special Case of general routing problem. In PPR, each processor is the origin (and destination) of at most one packet.

Broadcasting: 'On mesh'

Information reaches to all the nodes (processors)

$$O(2\sqrt{p} - 1) \approx O(\sqrt{p})$$



* PRAM Prefix Computation:

E.g. Let $n = 8$, $p = 8$

Let the input be $\Sigma = 12, 3, 6, 8, 11, 4, 5, 7$

Step 1

Processor P1 to P4 Computes the prefix sum of 12, 3, 6, 8 to arrive at

12, 15, 21, 29

and

Processor P5 to P8 Computes the prefix sum of 11, 4, 5, 7 to arrive at

11, 15, 20, 27

Step 2

Processor P1 to P4 sits idle.

Processor P5 to P8 will update their results by adding 29 to every prefix to obtain

40, 44, 49, 56

Output will be: 12, 15, 21, 29, 40, 44, 49, 56

Analysis

Step 1, takes $T(n/2)$

Step 2 takes $O(1)$

$\therefore T(n) = T(n/2) = O(\log n)$ ∎

Let there be 100 no. to be added.

- How much time will a person take to add?

1 99 100

- How much time will two person take to add?

1 49 50 51 49 100

1

i.e. 50

∴ Asymptotic Speed up = 99/50 = 1.98 ≃ 2

- Asymptotic Speed up : $\frac{S(n)}{T(n,p)}$

Where,

n = no. of inputs

S(n) = Sequential (run time of of best known Sequential algorithm).

p = no. of processors

T(n,p) = parallel (time taken by p processors using parallel algorithm).

∴ Total work done $O(S(n)) = p * T(n,p)$

∴ Efficiency, $\Theta = S(1)/(p * T(n,p)) = \frac{S(n)}{O(S(n))}$

To be work optimal it requires the efficiency to be $O(1)$.

E.g.

Let a be A n procener parallel algorithm that sorts n keys in $O(\log n)$ time. Let B be n^2 procedor algorithm that sorts all n keys in $O(\log n)$ time.

Find out which algorithm is work optimal.

Assumption: optimal run time to sort n keys $= O(n \log n)$

Speed up of A $= O(n)$
Speed up of B $= O(n)$

Work done of A $= n \cdot O(\log n)$
work done of B $= n^2 \cdot O(\log n)$

Efficiency of A $= 1$
Efficiency of B $= 1/n$

Since efficiency of A is 1, we can say that algorithm A is work optimal.

Linear Sorting:

Odd - Even Transposition Sort (Total digit should be power of 2)

7 5 3 2 7 8 4 6

i = 1

1 5 2 3 7 8 4 6

i = 2

1 2 5 3 7 4 8 6

i = 3

1 2 3 5 4 7 6 8

i = 4

1 2 3 4 5 6 7 8 Sorted

Time Complexity = O(P)

Shear sort (Mesh) n-processor

i=1

1	4	5	6
7	8	3	2
9	11	13	17
24	23	0	12

i=2

1	4	5	6
8	7	3	2
9	11	13	17
24	23	12	0

i=3

1	4	3	0
8	7	5	2
9	11	12	6
24	23	13	17

Fig 3:

0	1	3	4
8	7	5	2
6	9	11	12
24	23	17	13

Fig 4:

0 → 1 → 3 → 2

6 → 7 → 5 → 4

8 → 9 → 11 → 12

24 → 23 → 17 → 13

Fig 5:

0	1	2	3
7	6	5	4
8	9	11	12
24	23	17	13

Stated

$\log_2 24 + 1 = 5$

Step 8

For (i = 1 ; i <= log n + 1 ; i--)

{

if i = odd

Apply snake sort row-wise

if i is even

Apply Column sort

}

Time complexity = $O(\sqrt{p} \log p)$

* Prefix Computation on Butterfly:-

In first diagram;

Step 1: d+cc +cdd fercoard phase

Step 2: ddd+ dd+d

Step 3: Direct

In 2nd diagram;

backward phase;

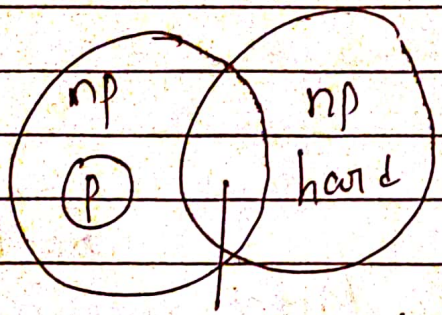
Step 1 and step 2: direct

Step 3: ddc+ ccd+

* Complexity Theory:-

- It Concerned with the resources, such as time ~~Complexity~~, space needed to solve Computational problem.

Complexity classes:



np-Complete.

(a) In a Computer Science, there exist some problems, whose solutions are not yet found, the problems are divided into classes known as Complexity classes.

(i) P-class: 'polynomial' Eg Merge, Quick Sort

- P is the class of decision problems that can be solved by a deterministic Turing machine in polynomial time. In other words, these are problems for which an algorithm exists that can find a solution in a time bound that is a polynomial function of the problem's input size.

↳ Finding Solution in exponential time.

(ii) NP-Class: 'Nondeterministic Polynomial'

- NP is the class of decision problems for which a proposed solution can be verified in polynomial time. While finding a solution may be hard, verifying it is relatively easy.

NP problems are those where the solution can be checked quickly, but there might not be an efficient algorithm to find the solution. ↳ Eg: Travelling Salesman problem.

(iii) NP-Hard :- Eg: Boolean Satisfiability problem (SAT)

- A problem is NP-hard if every problem in the NP class can be reduced to it in polynomial time. In essence, NP-hard problems are at least as hard as the hardest problems in NP, but they may or maynot be in NP themselves.

(iv) NP-Complete :- Eg: The Knapsack problem

- A problem is NP-Complete if it is both in NP and NP-hard. If you can solve an NP-Complete problem in polynomial time, you can solve all problems in NP in polynomial time.

* Load Balancing:

Problem Statement:

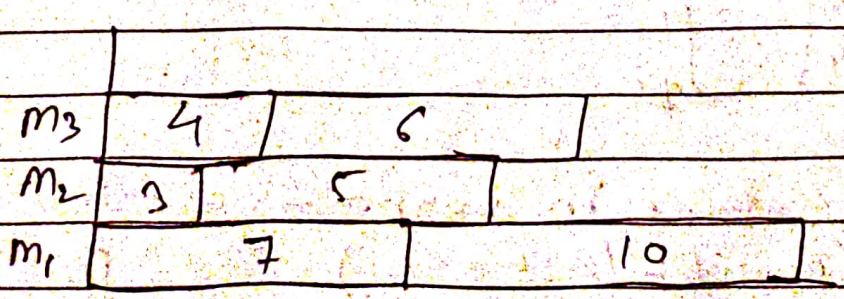
- A set of m identical machines.
- A sequence of jobs with processing times $p_1 \ldots$
- Each job must be assigned to one of the machines.
- When job j is scheduled, we don't know how many $1 \ldots p$, additional jobs, we are going to have and what are their processing times.

Goal: Schedule the jobs on machines in a way that minimizes the makespan.

List Scheduling:

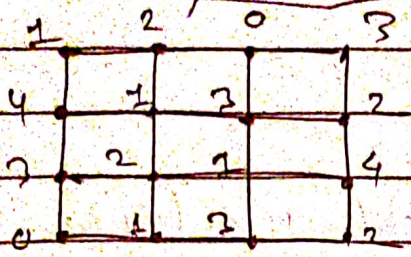
Greedy algorithm → always Schedule a job on the least loaded machine.

E.g: $m = 3$ $6 = 7, 3, 4, 5, 6, 10$

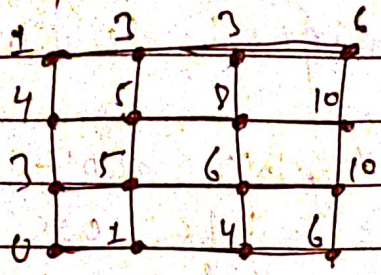


Makespan = 17

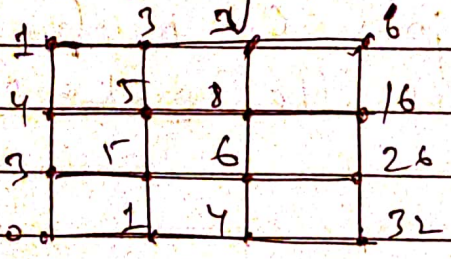
* Prefix Computation on Mesh:



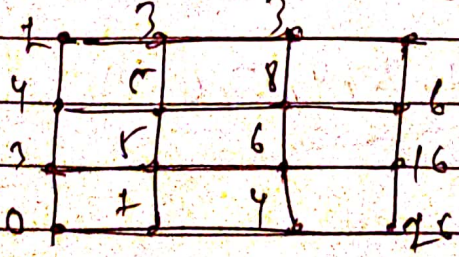
Step 1: Rowwise Computation : $\sqrt{p}$



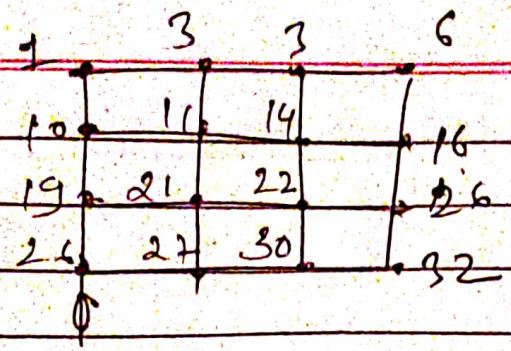
Step 2: Column-wise Computation $\sqrt{p}$
(only last column)



Step 3: Shifting (last column one down)
(Shift + update) → 2 (1+1) → 2



Step 4: Broadcasting in respective rows
(Second last row) → $\sqrt{p}$



∴ Time Complexity = $O(3 \cdot 3\sqrt{p} + 2)$

$= O(\sqrt{p})$

$r[0,9]$

$k=2$

$r[0,4]$

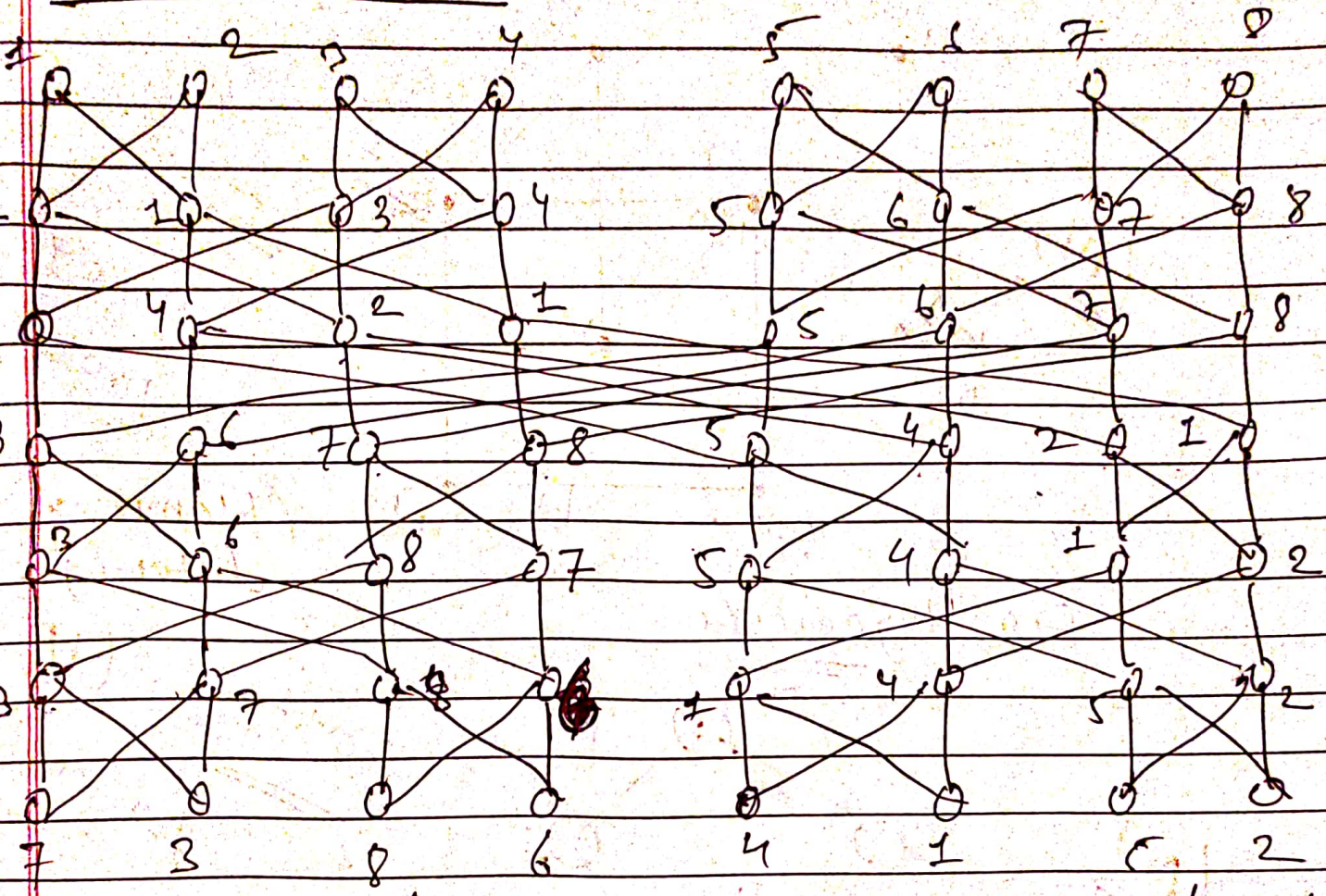
$k=2$

$r[0,1]$ $r[2,4]$

$k=1$

* Selection Sort:

Sorted

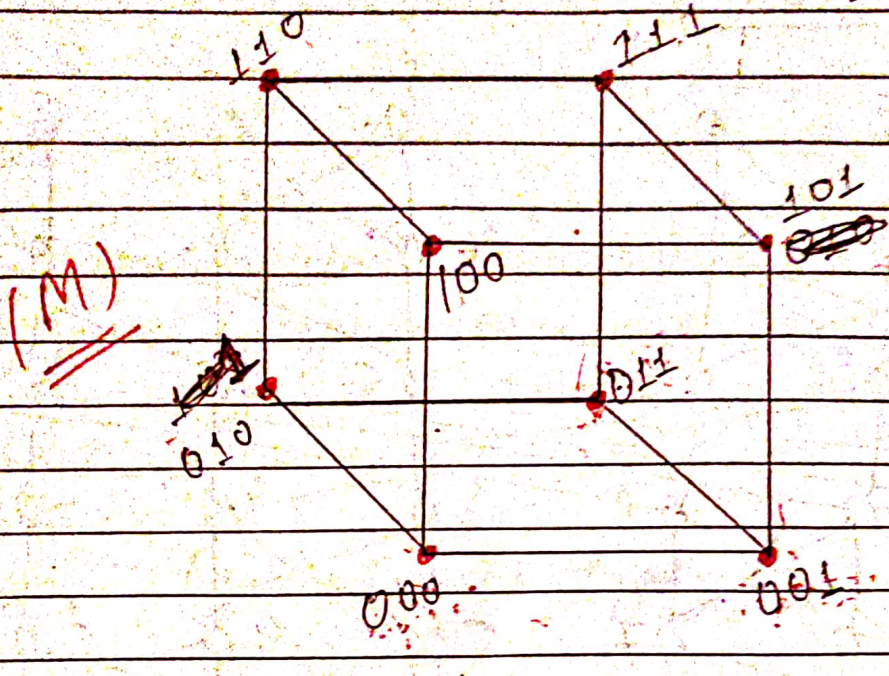


	cross	direct	cross	direct
(1) ✓	S S	B B	B B	S S
(2)	S B	S B	B S	B S
(3)	S S	S S	B B	B B
(4)	S S	B B	S S	B B
(5)	S B	S B	S B	S B

* Embedding of Hypercube to tree structure:

H_3

3 → dimensional(d)



(M)

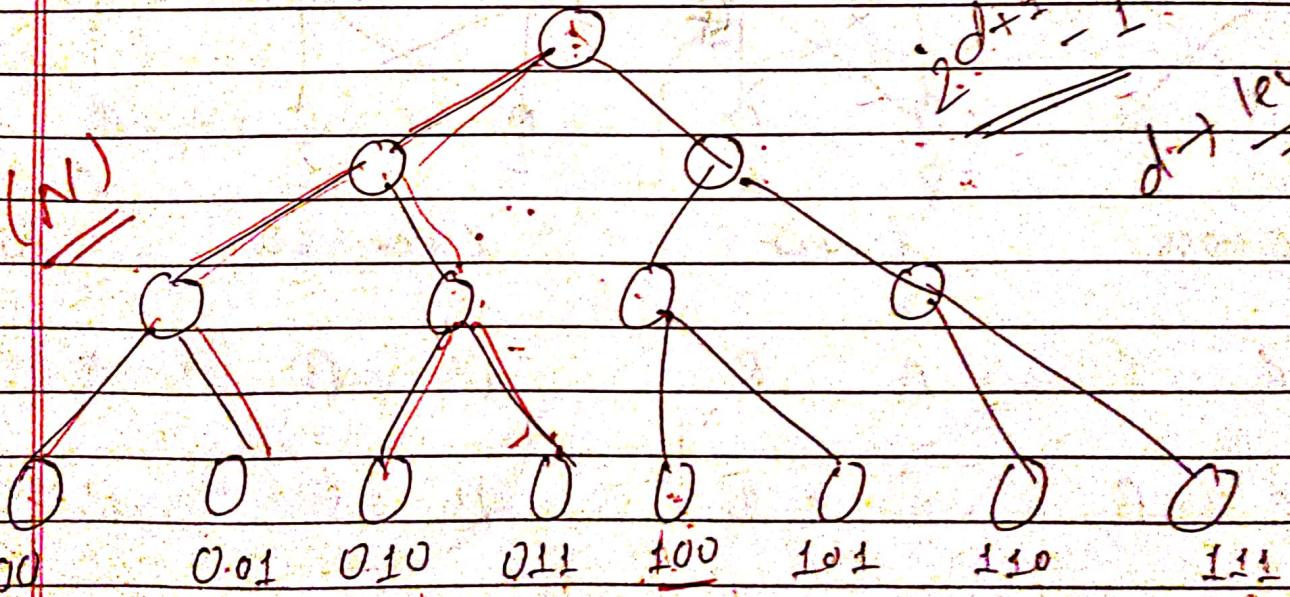
$2^d = 2^3 = 8$

No. of processors

(Hamming's distance)

Fig: Hypercube

(N)



$2^{d+1} - 1$ (15)

d → level

Expansion: 15/8

Dilation: 3 (Max length)

Congestion: 7 (Max nodes or processor)

* Embedding of Hypercube to Tree Structure:

- Changing one network (M) to another network (N) structure.
- Perform Calculation of network structure (N).
- Result obtained is mapped to the original network structure (M).

1. Expansion:

$$= \frac{|V_2| \rightarrow \text{Vertices of } B}{|V_1| \rightarrow \text{Vertices of } A}$$

2. Dilation:

- Max length of path of B that needs to be ~~mapped~~ traversed to map path of A.

3. Congestion:

- Max no. of times any node of B involved while mapping path of A with B.

* Probabilistic Analysis:

- Probabilistic analysis is the use of probability in the analysis of problems. Most commonly, we use probabilistic analysis to analyze the running time of an algorithm.

- We must know or make assumptions about the distribution of inputs.
- The expected cost of running the algorithm is over this distribution.
- The probabilistic analysis will give us average case running's time.

Traditional (Worst-case or average-case) analysis assumes a specific input distribution or a fixed input size to evaluate the performance of an algorithm. However, in many real-world scenarios, the input data is not uniform, and it may vary significantly, leading to different performance outcomes. Probabilistic analysis addresses this by incorporating randomness into the analysis, providing more realistic and often more informative results.

E.g. Hiring cost in procedure HIRE_ASSISTANT

* Randomized Algorithm :-

- A randomized algorithm is one that makes use of a randomizer (such as a random number generator).

- 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 :
(i) Las vegas algorithm
(ii) Monte Carlo algorithm.

	Las vegas	Monte Carlo
(i)	This algorithm always produce the same (correct) output for the same input.	(i) Monte Carlo is the algorithm whose output might differ from run to run for the same input.
(ii)	It guarantees the correctness of the solution.	(ii) It doesn't guarantee the correctness of the solution.
(iii)	Upper bound cannot be fixed (Continue running until they find a valid solution)	(iii) Upper bound can be fixed. (Fixed running time)

(iv) The execution time depends on the output of randomizer.

(v) Eg. Randomized Quick Sort

*** Finding Index of repeat:

```
K = RandInt(n)
if A[K] = 1, return K
```

(iv) It may generate the wrong answer sometimes that depends on the output of the randomizer.

(v) Eg: Primality Testing.

*** an array that Contains 1 *** repeat 300 times.

```
K = RandInt(n)
if A[K] = 1, return K
return "failed"
```

* Online Vs Offline Algorithm:

- Online algorithms are algorithms that need to make decisions without full knowledge of the input.
- Online algorithms may produce the results that are not optimal as it doesn't have the Complete input
- It procen it's input piece-by-piece in a serial fashion.
- Application: Stock Market, Secretary problem
  E.g: Least Recently Used (LRU).
- Offline algorithms are algorithm which solves the problem with cohole problem data from the beginning.
- Offline algorithms can produce the optimal solution as it is given the Complete input
- It procenes all the input at same time, as all data are initially available.
- Application: Sorting in the list, Search in array
  E.g: optimal page replacement algorithm

- Because online algorithm doesnot 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 algorithms has focused on the quality of decision-making that is possible in this setting.

Competitive Analysis:

- Competitive analysis formalizes this idea by comparing the relative performance of an online and offline algorithm for the same problem instance.

- An online algorithm A is c-competitive if there is a constant b for all sequences S of operations

$$A(S) < c\ OPT(S) + b$$

Where,

A(S) = Cost of A on the sequence S

OPT(S) = optimal offline cost for the same sequence

- Competitive ratio is a worst case bound.

* Ski Rental Problem:

- It is often used to illustrate the concept of online algorithms and competitive analysis.

- Assume that you are taking ski lessons. 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?

Soln:

- 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 = y guarantees that you never pay more than twice the cost of the offline strategy.

E.g.

* Assume y = 7$, Thus after 6 rents, you buy.

Your total payment = 6 + 7 = 13 $

Setting: K = y

- Proof: When you buy skis in your K^{th} visit, even if you quit right after this time, $t \geq y$.

- Your total payment is $K - 1 + y = 2y - 1$
- The offline cost is $\min(t, y) = y$.
- The ratio is $(2y-1)/y = 2 - (1/y)$

We can say that this strategy is $[2 - (1/y)]$-Competitive.

* Load Balancing:-

- A Computer methodology to distribute workload across multiple Computing resources like Computer cluster, network links, central processing units, disk drives etc.

Goals of load balancing is

(i) Achieve optimal resource utilization
(ii) Maximize throughput
(iii) Minimize response time
(iv) Avoid overload on individual resource
(v) Avoid Crashing.

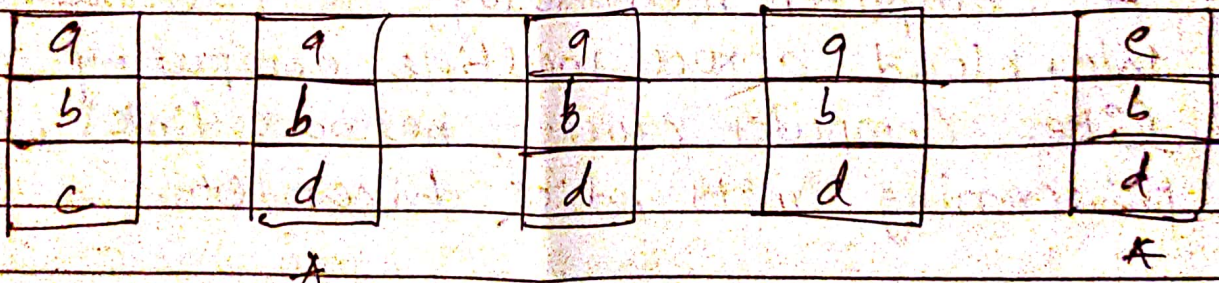
Eg of load balancing algorithm is weighted round robin algorithm.

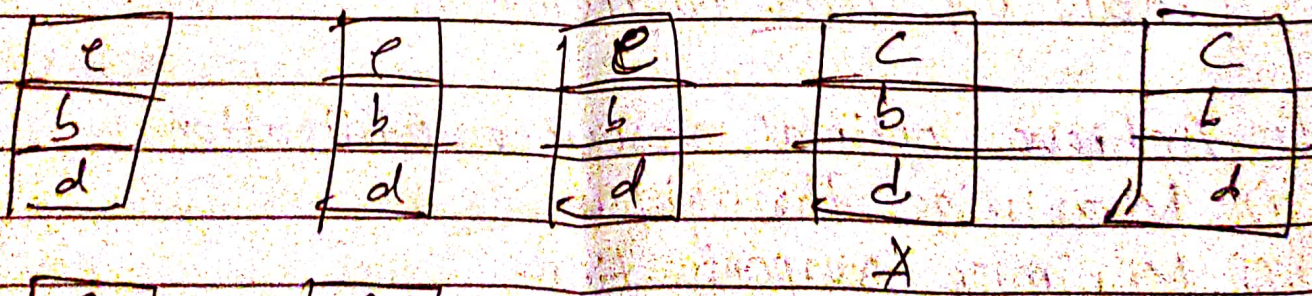
- Load balancing is required for
(i) High Availability
(ii) High Reliability — ARS
(iii) High Scalability
(iv) Ease of maintenance
(v) Resource sharing

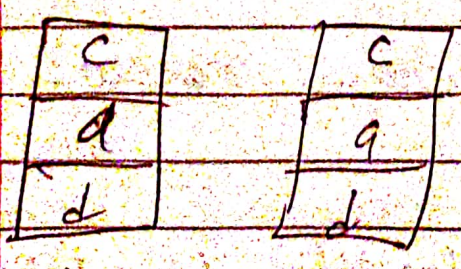
LFD (Longest forward distance)

An optimal offline page replacement strategy. (K=3) m=5

G = a, b, c, d, a, b, e, d, e, b, c, c, a, d.







4 caches misses in LFD.

Online paging Algorithms:

(i) FIFO (First in first out)
(ii) LIFO (Last in first out)
(iii) LRU (Least Recently used)

- LIFO is not Competitive
- FIFO & LRU is k Competitive