UNIT 1 ADVANCE ALGORITHM ANALYSIS AND DESIGN TECHNIQUES

1.1 Amortized Analysis (Aggregate Analysis, Accounting Method, Potential Method), Probabilistic Analysis, Las Vegas & Monte Carlo Algorithms.

1.2 Greedy Algorithms (Tree Vertex Splitting, Job Sequencing with deadlines), Dynamic Programming (Greedy Vs Dynamic, String Editing; optimal BST), backtracking (Sum of subsets, Knapsack Problem), Randomized Algorithms (Identifying the repeated elements, Primality Testing, karger's algorithm)

Falguni

ALGORITHM

→ Set of finite steps that should generate one or more output.
OR
→ A finite set of instructions that if followed, accomplishes a particular task.
OR
→ A set step-by-step procedure or set of rules to solve a problem or to perform a computation
OR
→ The foundation of computer science used to process data, perform calculations and automate tasks.

1. Algorithm development approach

Linear
Non-linear
Dynamic Programming
Backtracking
Greedy
Divide and Conquer

2. Algorithm Validate

Predicate Calculus

e.g. database → query optimization, relational algebra, Tuple calculu

3. Algorithm Analysis

Time complexity, space complexity

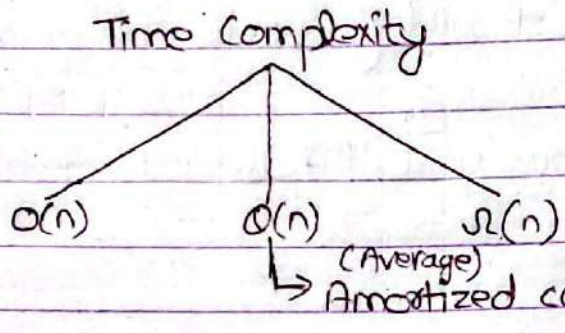
4. Fine Tuning (trying to reduce the step for better result).

NP ↑ Exponential $O(2^n)$ — Factorial – NP
P – high order: Logarithmic $O(\log n)$ — Hay Problem [↑size ↑behavior] (asymptotic behavio
Quadratic $O(n^2)$ — Halting Problem
⋮
Linear $O(n)$
Constant $O(1)$

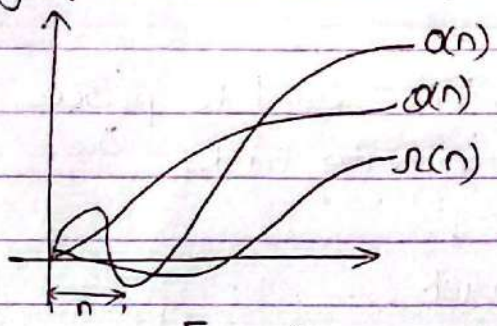
constant step concept
n = 10 → 5 steps
n = 1000 → 5 steps

$n \log n \Rightarrow 8 \log_2 2^3 \Rightarrow 24$ ✱ exact value ⇒ polynomial
else
non polynomial

5. Implement



Asymptotic Behaviour (sudden change in behaviour).

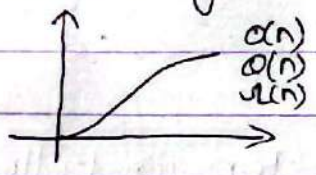


where n ⇒ [no. of ~~experim~~ iteration↓ experiment↓]

$n \to \infty$ [experiment must be carried on unless experiment conclude]

Asymptotic Notation → [O(n), Θ(n), Ω(n)]

☐ Algorithm can have same asymptotic Notation i.e same worstcase, ~~Avgera~~ Average case and best case.



e.g. Hash function O(1)
Mergesort (n log n)

☐ Time Complexity is no. of steps required to solve a problem
It must be independent of machine.

Falgun 21

- used to show that the average cost of an operation is small, even though some operations within the sequence is expensive.

AMORTIZED COST ANALYSIS

- It is also known as average cost analysis.
- It is the method of analysing the costs associated with the data structure that averages the worst operations out over time.
- It is required because it is hard to justify where the algorithm is up to the mark best on bestcase and worstcase.
- It differs from average ~~cost~~ case analysis [No probability involved]
- This analysis guarantees the average performance of each operation in the worst case. Thus worthwhile to implement if an algorithm's average cost is better.
- For example Hash Tables, Disjoint sets, Splay Trees.
- There are three types of amortized analysis :-
  1. Aggregate Analysis
  2. Accounting Method
  3. Potential Method

1. Aggregate Analysis Method

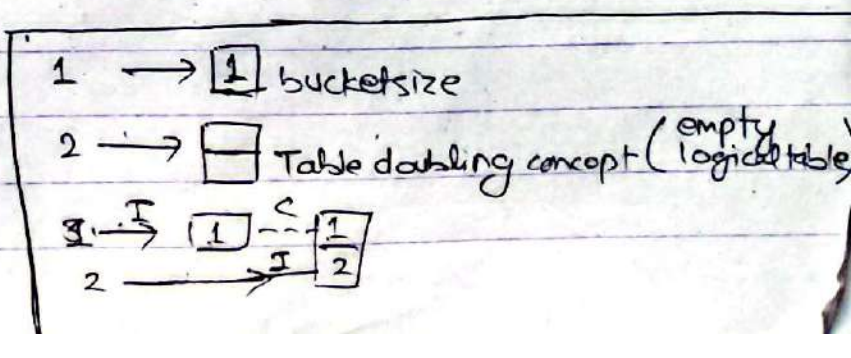
Here, we determine upper bound for a sequence of n operations, say T(n) then amortized cost of n operations is set equal to T(n)/n.

OR

a. First, we must show that sequence of n operations takes T(n) time in worst case.

b. Then we show that each operation takes T(n)/n time on average (avg cost/Amortized cost per operation). So, in aggregate analysis, ~~each~~ all operation have same amortized cost.

Hash Table Concept ⟹



Amortized cost → $\hat{C_i}$
Normal cost → C_i
Insert cost → 1
copy cost → 1
Table doubling cost → x

Example : Hash Table

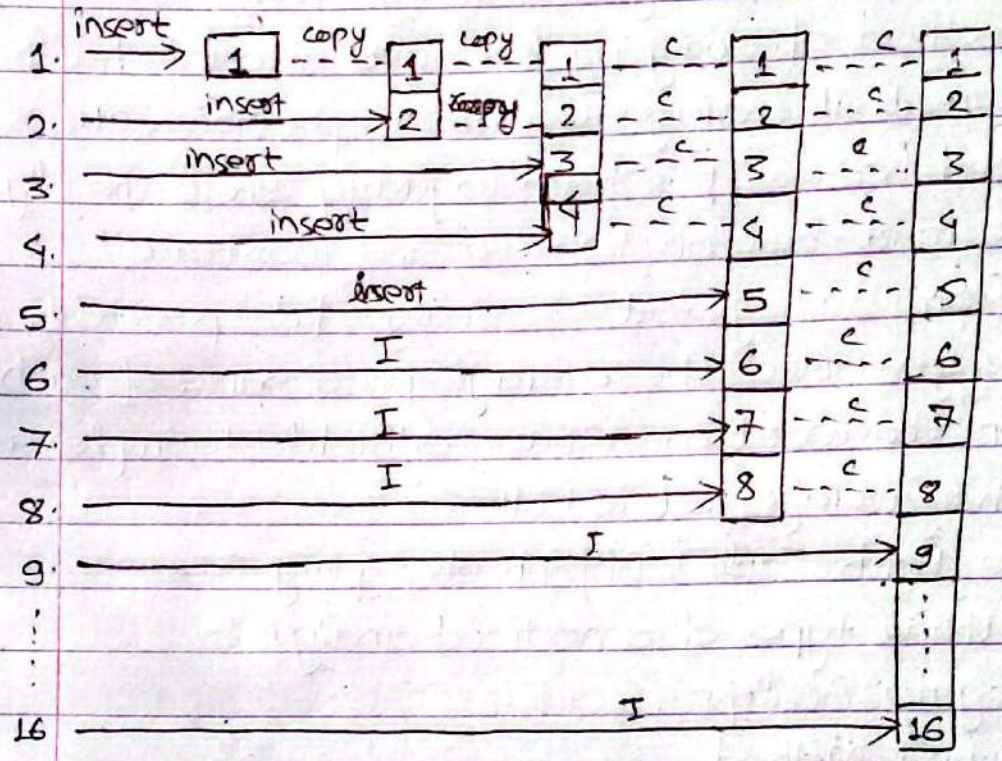


Fig : Table Doubling Concept

Size, insert cost, copy cost & Actual cost are tabulated below:

i	1	2	3	4	5	6	7	8	9
size(i)	1	2	4	4	8	8	8	8	16
insert cost	1	1	1	1	1	1	1	1	1
copy cost	0	1	2	0	4	0	0	0	8
C_i	1	2	3	1	5	1	1	1	9

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

if $i=2$ then $i-1$
$\Rightarrow (2-1)$
$\Rightarrow 1$
$\Rightarrow 2^0$

Generalization of c_i

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

$$c_i = \sum_{i=0}^{n-1} 1 + \sum_{i=0}^{\lceil \log_2 n \rceil} 2^i \quad \text{OR} \quad \sum_{i=1}^{n} 1 + \sum_{i=0}^{\lceil \log_2 n \rceil} 2^i$$

Here, $n = 9$

insert + copy

$= 9 + (1+2+4+8)$

$= 9 + 15$

$= 24$

$\lceil \log_2 n \rceil = \lceil \log_2 9 \rceil = 4$

$$c_i = \sum_{i=1}^{n} 1 + \sum_{i=0}^{\lceil \log_2 n \rceil}$$

$= n + 2n - 3$

$= 3n - 3$

$= O(3n)$

Here, $O(n) = 3n$

Now,

Amortized cost $(c_i) = O(n) = \frac{O(n)}{n \text{ (window size)}} = \frac{3n}{n} = 3 = 1$

∴ Amortized cost is constant i.e $O(1)$

Use aggregate method in hash table where following assumptions are made.

1. Insertion cost = 1
2. Copy cost = 1
3. Table doubling cost irrespective of size = 1

Sol:

i	1	2	3	4	5	6	7	8	9
size (i)	1	2	4	4	8	8	8	8	16
insert cost	1	1	1	1	1	1	1	1	1
copy cost	0	1	2	0	4	0	0	0	8
Table doubling cost	0	1	1	0	1	0	0	0	1
c_i	1	3	4	1	6	1	1	1	10

Generalization of c_i

$$c_i = \sum_{i=1}^{n} 1 + \sum_{i=0}^{\lceil \log_2 n \rceil} 2^i + \sum_{i=1}^{\lceil \log_2 n \rceil} 1$$

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

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

$$= 4n - 3$$

Here,

$$O(n) = 4n$$

Now,

$$\text{Amortized cost } (c_i') = On = \frac{O(n)}{n} = \frac{4n}{n} = 4$$

12/03
Sunday

2 Accounting Method (Bank balance ≥ 0)

- It is also known as Banking Method as it borrows ideas from finance. Thus, this method is easy to find out the optimal cost.
- Here, "each operation is assigned a charge" or the amount of we charge an operation is called its amortized cost.
- Some operations are charged more or less than they actually cost
- If an operation's amortized cost exceeds its actual cost, the difference is assigned as credit.
- Credit can never be negative in any sequence of operations.
- otherwise the total amortized cost would not be an upper bound on total actual cost.
- This method requires setting of the amortized cost such that $B \geq 0$ i.e. starts with the guess of $\hat{c}_i$.

- For the same hash table, assume $\hat{c}_i = 3$

i	1	2	3	4	5	6	7	8	9
size i	1	2	4	4	8	8	8	8	16
c_i	1	2	3	1	5	1	1	1	9
		1	0	2	-2	+2	2	2	-6
$\hat{c}_i$	3	3	3	3	3	3	3	3	3
Bank balance	2	3	3	5	3	1	7	9	3

- Assume $\hat{c}_i = 2$ for the first and $\hat{c}_i = 3$ for the rest.

i	1	2	3	4	5	6	7	8	9
size i	1	2	4	4	8	8	8	8	16
c_i	1	2	3	1	5	1	1	1	9
		1	0	2	-2	2	2	2	-6
$\hat{c}_i$	2	3	3	3	3	3	3	3	3
Bank balance	1	2	2	4	2	4	6	8	2

Q Can you find c_i' so that the bank balance is exactly 0 but not below.

Sol:

Assume $c_i' = 2$ for the first three and the rest $c_i' = 3$

i:	1	2	3	4	5	6	7	8	9
size i	1	2	4	4	8	8	8	8	16
c_i	1	2	3	1	5	1	1	1	9
c_i'	2	2	2	3	3	3	3	3	3
Bank balance	1	1	0	2	0	2	4	6	0

if guess = 2

c_i'	2	2	2	2	2	2	2	2	2
c_i'	1	1	0	-1					

↳ invalid

3. Potential Method

- It is known as Physicist's Method which uses a potential function "ϕ" to measure "stored" work done by the data structure.
- It track the difference between the actual cost and the amortized cost through potential function so that total work is distributed more evenly over the operations.
- The amortized cost of an operation is actual cost plus change in potential i.e

$$\hat{c_i} = c_i + \Delta\phi$$

where, $\Delta\phi = \phi(D_i) - \phi(D_{i-1})$

D_i = potential after i^{th} operation has been executed

D_{i-1} = potential before i^{th} operation

Major advantages of using potential Method

a. It provides a more systematic and a mathematically rigid approach via defining a precise potential function that captures the "stored work" in the system.

b. It works well for dynamic and non-uniform operations via adjusting based on system states (e.g dynamic arrays, self-adjusting data structures, Fibonacci heaps etc).

c. It handles worst-case ~~a~~ costs more smoothly by balancing expensive operations without needing credit assignments.

d. It ~~avoids the need for~~ is easier to generalize and apply to complex data structures & via fine-grained analysis that provides only a general bound.

e. It handles variable operation sequence, better via adapting to any sequence of operations (e.g online algorithms where future operations are unknown). and visualizing resource usage & internal state of data structure. and data structure behaviour

12/04 Monday

Here, we have to define potential function (ϕ)

$$\phi(D_0),\ \phi(D_i),\ \phi(D_{i-1})$$

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

lets assume : For hash table (table doubling)

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

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

we have,

Amortized cost $(c_i^{\wedge}) = c_i + \phi(D_i) - \phi(D_{i-1})$

where,

$$\phi(D_i) - \phi(D_{i-1}) \geq 0$$

we know,

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

$$\therefore c_i^{\wedge} = \begin{cases} i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{otherwise} \\ 2i - 2^{\lceil \log i \rceil} - \left[2(i-1) - 2^{\lceil \log (i-1) \rceil}\right] \end{cases}$$

Case I : $c_i = i$ [i is exact power of 2]

$$c_i^{\wedge} = c_i + \phi(D_i) - \phi(D_{i-1})$$

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

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

$$= i + 2i - 2i + 2 - [2i - 2 - i + 1]$$

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

$$= 2i - 2i + 3$$

$$= 3$$

Case II : $c_i = 1$ [$i-1 \neq$ exact power of 2]

$$\hat{c_i} = c_i + \phi(D_i) - \phi(D_{i-1})$$
$$= 1 + 2i - (i-1) + [2(i-1) - (i-1)]$$
$$= 1 + 2i - i + 1 - [2i - 2 - i + 1]$$
$$= 2 + i - [i-1]$$
$$= 2 + \cancel{i} - \cancel{i} + 1$$
$$= 3$$

OR

$$\hat{c_i} = 1 + \left[2i - 2^{\lceil \log i \rceil} - \left[2(i-1) - 2^{\lceil \log (i-1) \rceil}\right]\right]$$
$$= 1 + \left[\cancel{2i} - \cancel{2^{\lceil \log i \rceil}} - \cancel{2i} + 2 + \cancel{2^{\lceil \log (i-1) \rceil}}\right] \quad \left[\because 2^{\lceil \log i \rceil} = 2^{\lceil \log (i-1) \rceil}\right]$$
$$= 3$$

12/08
Friday

Randomized Algorithms

- An algorithm that employs randomness (e.g random numbers, probabilistic choices) as part of its logic to influence computatio
- Also called probabilistic algorithm as it uses random choices to achieve results.
- This algorithm may exhibit variability in ~~not~~ runtime, correctness or both depending on random decisions made during execution.
- It can be categorized into:
  - i. Las Vegas algorithm
  - ii. Monte Carlo algorithm

  ] - concept come from casino gambling.

i. Las Vegas Algorithm

- Type of randomized algorithm that always produces the correct result, but its runtime is a random variable. (it can vary unpredictably. (unbounded)
- It prioritizes correctness over efficiency.
- for example quicksort, randomized Prim's Algo.
- often implemented using recursive techniques but is tricky.
- ~~Might be slow in~~

ii Monte Carlo Algorithm

- Type of randomized algorithm that may produces incorrect results with a certain (typically small probability).
- Its runtime has a bounded or fixed runtime.
- It prioritizes efficiency over absolute correctness.
- for example Monte carlo integration, Primality Test, simulation.
- often implemented using iterative techniques.

Primality Testing

To ~~create~~ check whether the given number is prime or not.
The complexity of primality testing is $O(n)$.

a. Brute Force Method

```
Read a number (n)
For i=2 to n
  n % i
if n % i =0
  n =i
  prime
else
  not prime
```

e.g

n = 5

n	i	Remainder
5	% 2	1
5	% 3	2
5	% 4	1
5	% 5	0

∴ prime

e.g

n = 4 , 4 % 2 = 0

since, n ≠ i, so not prime.

b. Randomization (Monte-Carlo)

```
if a^p % n ==1      where, p<n. ; n is the no. to be checked,
  prime                        a <n
else
  not prime
```

This concept will not always provide correct answer.
Complexity = $O(1)$

e.g

n = 5

a = 1, 2, 3, 4

p = 1, 2, 3, 4

$3^2 \% 5 \Rightarrow 9 \% 5 = 4$ so, not prime

let, $1^5 \% 5 \Rightarrow 1 \% 5 = 1$, so, prime

let, Randomized (a) : 3

Randomized (p) : 2

Greedy Algorithm

a. Huffman coding $o(n \log n)$

b. Shannon-Fano

c Job Sequencing

a. Huffman-coding [Here, shorter codes are assign to more frequent character & longer codes are for less frequent. uses variable length code to represent character in message]

- Lossless data compression technique
- Reduces the average bit length
- provides lower bit-length to maximum occuring character and vice-versa.
- generates prefix code.
- uses in zip files & multimedia

example :

a : 0.15

b : 0.15

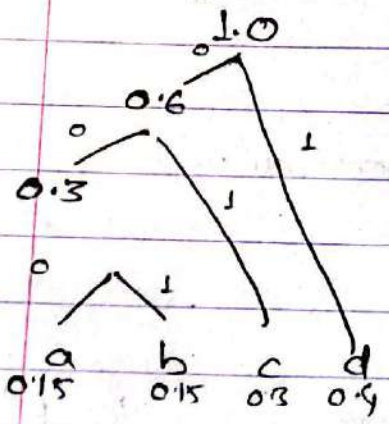
c : 0.3

d : 0.4

abcabcdbc	
a : 2	00
b : 3	01
c : 3	10
d : 1	11

$$\text{Average bit length} = \frac{\sum_{i=1}^{n} P_i * \text{bit length}}{n}$$

$= O(n \log n)$



P_i		bit length (prefix code)
a : 0.15	000	3
b : 0.15	001	3
c : 0.3	01	2
d : 0.4	1	1

Average bit length $= (3 \times 0.15 + 3 \times 0.15 + 2 * 0.3 + 1 * 0.4)/4$

$= 1.9/4$

$= 0.475$

~~sort~~ in descending order
~~sort~~

b. Shannon-Fano (lowest to highest arrange bit length)

	~~step1~~	~~step2~~	~~step3~~	bit length (n)
a : 0.25	0	0	0	3
b : 0.25	0	0	1	3
c : 0.3	0	1		2
d : 0.4	1			1

$$\text{Entropy}(H) = \sum_{n=1}^{n} P_n \log_2 \frac{1}{P_n}$$

$$= (0.25)\log_2 \frac{1}{0.25} + (0.25)\log_2 \frac{1}{0.25} + (0.3)\log_2 \frac{1}{0.3} + (0.4)\log_2 \frac{1}{0.4}$$

$$=$$ bits

$$\text{Avg length}(L) = \sum_{n=1}^{n} P_n m_n$$

$$= (0.25 \times 3) + (0.25 \times 3) + (0.3 \times 2) + (0.4 \times 1)$$

$$=$$ bits

$$\text{Efficiency}(\eta) = \frac{H}{L} =$$

$$\text{Redundancy} = 1 - \eta =$$

Greedy Algorithms

- Simple and straightforward algorithms that take decision on the basis of current available information without worrying about the effect of decision in future.
- It chooses the option that looks best at the moment with the hope that leads to a globally optimal solutions.

Features

- At each steps, it makes greedy choices
- No backtracking
- More efficient in time than eg dynamic programming.
- Doesnot always produce optimal solutions.

- For eg 0/1 knapsack, Huffman coding, shannon-Fano. etc.

Greedy Algo = cost minimization - profit Maximization

Both are lossless data compression algos that uses variable-length codes based on symbol frequencies.

#1. Which algorithm is better, Shannon-Fano or Huffman coding?
2 Find the complexity of both.

⇒ Huffman Coding is generally considered better as it guarantees optimal results and optimal compression efficiency.

	Shannon-Fano	Huffman Coding
a	It is a heuristic-based algorithm works by sorting the symbols via probability or frequency.	a. It is a greedy-based algorithm that repeatedly combines the two least frequent symbols first.
b.	It is top to bottom approach [which splits symbols into groups] trying to balance the probabilities which can lead to uneven/suboptimal code.	b. It is bottom-up approach which merges least frequent nodes.
c.	It may is not always produces an optimal prefix code.	c. It always produces an optimal prefix code (minimum redundancy
d	Time complexity : $O(n \log n)$ (sorting)	d. Time complexity : $O(n \log n)$ (priority queue)
e.	It is simpler to implement but less efficient & outdated.	e. It is complex to implement but more efficient.
f	Tree construction is built by recursive partitioning.	f. Tree construction is built using priority queue.
g.	Uses only for historical context or theoretical understanding & rarely used.	g. Uses for real world compression ; compression standards (ZIP, JPEG, MP3 etc

3. Generate a random no. using las-vegas concept that always bring correct answers.

```
impoat random
def las_vegas_random():
        while True:
            num = random.randint(1, 15)
            if 1<=num<=10
            return num
print("Las Vegas random no:(1-10):", las_vegas_random())
```

Output:

Las vegas random no:(1-10): 7

Baisakh 8th sunday

Job Sequencing with Deadlines

Each job is represented by J_i
Each job J_i will have profit P_i and each job has deadline d_i.
To get the profit (P_i) each job needs to be completed.

Assumption

Only one processor
At any instant of time, one job will be serviced.
Each job requires 1 unit of time to complete.

Procedures

1. Arrange job J_i w.r.t P_i in descending order. ($P_1 > P_2 > P_3 \ldots > P_n$)
2. Allocate the time slots.

Time slots = max of d_i

OR

$\max(d_i)$

Calculation:

$J_i = \min(d_i, \max(d_i))$

Analysis

a. Time slot calculations $\Rightarrow n$-step
b. constant step calculation (k) : 1
c. To insert J_i in S_i [true case] $\Rightarrow 1$
d. To insert J_i in S_i [false case] $\Rightarrow n^2$

Time complexity $= n + 1 + 1 + n^2$

$= n^2 + n + 2$

$\therefore O(n) = n^2$

Algorithm

1 Arrange all job in decending order w.r to P_i

2 Allocate time slot w.r.t max^n deadline

3 $k_i = \min$ of $\{$ max of deadline $\}$, $d[J_i]$) slotposition.

4 Allocate time slot for J_i on the basis of k.

5. If the slot is already occupied search for position before it.

6 If all slots are pre-occupied, drop off the job J_i

7. Max^n deadline & n = total Jobs

total time = n x m

eg/ Find proper job sequence for following scenario.

Jobs	J_1	J_2	J_3	J_4
profit	80	30	50	70
deadlines	1	2	3	1

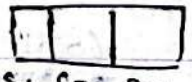
solution

Arranging in descending order of profit

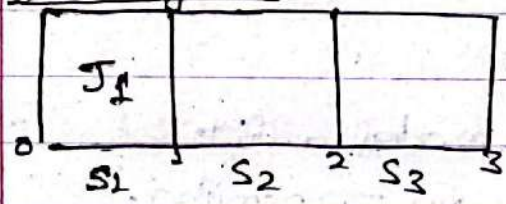
Jobs	J_1	J_4	J_3	J_2
profit	80	70	50	30
deadlines	1	1	3	2

Time slots = max of deadlines

= 3



Inserting J_1



For J_1 = min (d_1 , max(d_1))

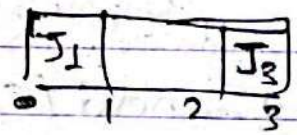
= min (1 , 3) = 1

For J_4 = min (d_4 , max(d_4))

= min (1 , 3) = 1 ✗ not selected

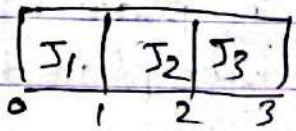
For J_3 = min (d_3 , max(d_3))

= min (3 , 3) = 3



For J_2 = min (d_2 , max(d_2))

= min (2 , 3) = 2



Job sequence : J_1 , J_2 , J_3

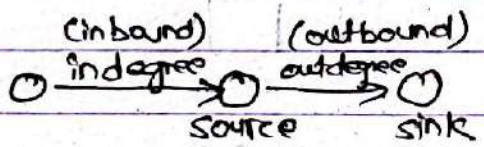
Total profit = $P_1 + P_2 + P_3$

= 80 + 30 + 50

= 160 //

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 $\langle i, j \rangle \in E$
- Source vertex has in-degree '0' and sink vertex has out degree.



- A weighted tree can be used to model a distribution network in which electrical signal are transmitted.
- Transmission of power from ~~root to noleaf~~ one network to another may result in some loss.
- with the loss, each edge in tree is labeled while traversing that edge.
- The network maynot be able to tolerate losses beyond its threshold (δ).
- So, in place where loss exceeds the threshold, boosters have been placed i.e splitting is done.

⇒ TVS problem determine an optimal splitting via greedy approach.

- Compute for each node $u \in V$
- compute max^n delay $d(u)$ from u to any other node in subtree
- Compare with threshold. and splitting is performed on necessity.

- used in ~~to~~ irrigation canal, power transmission line, signal booster.

Calculation:

$$d(u) = \max_{v \in d(u)} \{ d(v) + w(u,v) \}$$

where,

u → node

V → set of all edges

d(u) → delay of node.

d(v) → connecting node of u.

w(u,v) → distance

δ → threshold.

Analysis

If n is the total leaf node; node traverse = n-1

Generating Max value = 1

Compare with δ and split if necessary = 1

$$\Sigma = n-1+1+1$$

$$= n+1$$

$$\therefore O(n) = n \quad \text{(linear)}$$

↑nodes ↑steps

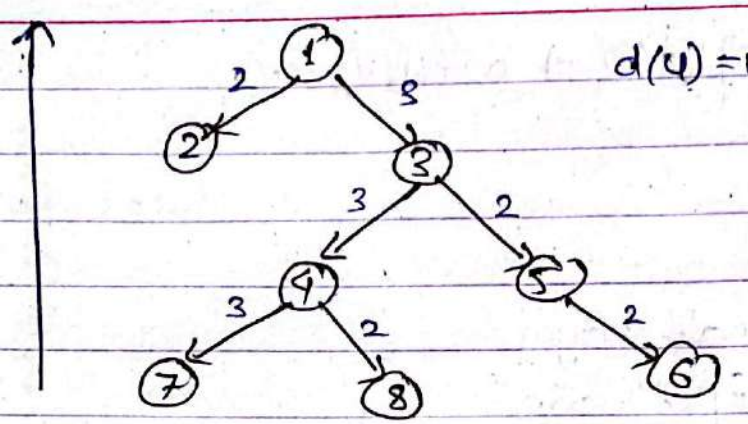
Dynamic Programming

- It is an optimization problem based on the concept of solving ~~all possible~~ sequences of sub-modules (i.e it tries to figure out all possible solutions.
- It provides the optimal solution.
- For e.g knapsack problem, Fibonacci seq, Longest common subsequ etc.

01/08
Monday

Follow Bottom up approach
Splitting factor > 8

eg



$$d(u) = \max_{v \in d(u)} \{ d(v) + w(u,v) \}$$

If $\delta = 5$. Perform TVS on given tree.

Step 1:
Leaf nodes, $d(u) = 0$ i.e
$d(2) = d(7) = d(8) = d(6) = 0$

For intermediate node 4:

$c(4) = \{7, 8\}$

$$d(4) = \max_{v \in 7,8} \begin{cases} d(7) + w(4,7) \\ d(8) + w(4,8) \end{cases}$$

$$= \max \begin{cases} 0+3 \\ 0+2 \end{cases}$$

$\therefore d(4) = 3$

Here $d(4) = 3 < 5$ (No splitting)

For Node 5

$c(5) = \{6\}$

$$d(5) = \max_{v \in 6} \{ d(6) + w(5,6) \}$$

$= 0+2 = 2 < 5$ (No splitting)

For Node 3

$c(3) = \{4, 5\}$

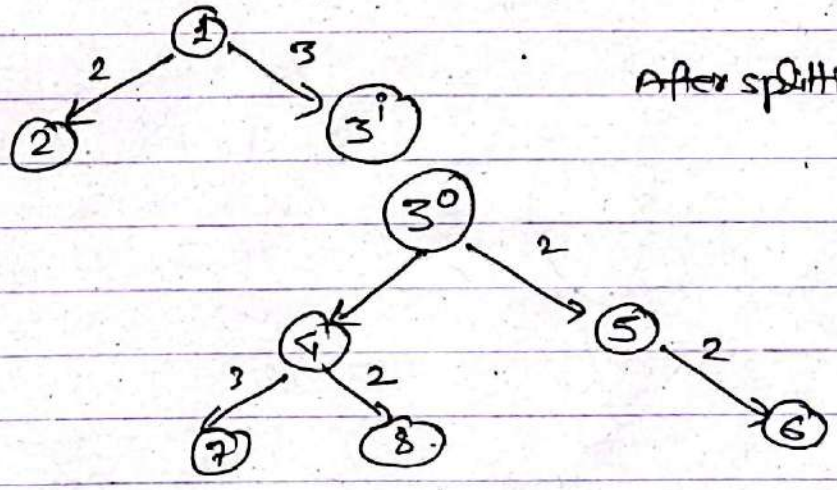
$$d(3) = \max_{v \in 4,5} \begin{cases} d(4) + w(3,4) \\ d(5) + w(3,5) \end{cases}$$

$$= \max \begin{cases} 3 + 3 \\ 2 + 2 \end{cases}$$

$$= \max \begin{cases} 6 \\ 4 \end{cases}$$

$= 6 > 5$

Node 3 must be splitted.



After splitting $d(3) = 0$

For Node 1

$c(1) = \{2, 3\}$

$$d(3) = \max_{v \in 2,3} \begin{cases} d(2) + w(1,2) \\ d(3) + w(1,3) \end{cases}$$

$$= \max \begin{cases} 0 + 2 \\ 0 + 3 \end{cases}$$

$= 3 < 5$ (No splitting)

01/10
Wednesday

String Editing

Q.9

delete cost = 1
insert cost = 1
copy cost = X

source : bbacdb
Target : adbb

Method 1

source	b	b	a	c	d	b	b	
delete	1	1	0	1	0	0	1	
copy	0	0	1	0	1	1		
insert	0	0	0	0	0	0	1	

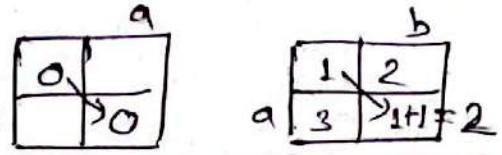
⇒ b b a c d b b
D D C D C C I
cost : 4

Method 2

a d b b c c c b
I I C C D D D D
cost : 6

Method 3

delete all and insert all
b b c c c b c d b b
D D D D D D I I I I
cost = 10



e.g 2

source: a a a b b c

target: a a b c

→ insert
↓ remove (Delete)
↘ Replace [Copy]

		NULL	a	a	b	c
		→	→	→	→	↓
NULL		0	1	2	3	4
a	↓ 1	1	0	1 →	2 →	3
a	↓ 2	2	1	0 →	1 →	2
a	↓ 3	3	2	1	1 →	2
b	↓ 4	4	3	2	1 →	2
b	↓ 5	5	4	3	2	2
c	↓ 6	6	5	4	3	2

No. of operations to be performed = 2

operations: C C D C C D

e.g 3

source: b b a c d b

target: a d b b

		NULL	a	d	b	b
		→	→	→	→	↓
NULL		0	1	2	3	4
b	↓ 1	1 →	2	2	2	3
b	↓ 2	2	2 →	3	2	2
a	↓ 3	3	2 →	3	3	3
c	↓ 4	4	3	3 →	4	4
d	↓ 5	5	4	3 →	4 →	5
b	↓ 6	6	5	4	3	4

No. of operation to be performed = 4

operation: C I C D C D D

Optimal Binary Search Tree (OBST)

Suppose, we have keys $k_1 < k_2 < k_3 < \dots < k_n$

No. of binary search tree that can be created

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

Dynamic Approach

1. weight $= w[i, j]$
2. Expected value $= e[i,j] = \min \{ e[i, r-1] + e[r+1, j] + w[i,j] \}$
3. root $= r[i, j]$ → choose root node where $e[i, j]$ is the minimum

→ if $j = i-1$

then $w[i,j] = q_i - 1$

else

$j \geq 1$

$w[i,j] = w[i, j-1] + p_i + q_i$

p_i → probability of succesful search
q_i → probability of unsuccesful search

e.g. construct OBST and find cost.

i	0	1	2	3	4	5
p_i	-	0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

sol:

i	0	1	2	3	4	5
p_i	-	0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10
	x	0.25	0.15	0.10	0.15	0.30

$n = 6$

step1: construction of weight/probability matrix(w) [6x6]

q_i →	0	1	2	3	4	5
1	0.05	0.30	0.45	0.55	0.70	1.0
2	-	0.10	0.25	0.35	0.50	0.80
3	-	-	0.05	0.15	0.30	0.60
4	-	-	-	0.05	0.20	0.50
5	-	-	-	-	0.05	0.35
6	-	-	-	-	-	0.10

diagnol = left side value wise sum

step2: construction of expectation matrix (E)

q_i →	0	1	2	3	4	5
1	0.05	0.45	0.90	1.25	1.75	2.75
2	-	0.10	0.40	0.70	1.20	2.00
3	-	-	0.05	0.25	0.60	1.30
4	-	-	-	0.05	0.30	0.90
5	-	-	-	-	0.05	0.50
6	-	-	-	-	-	0.10

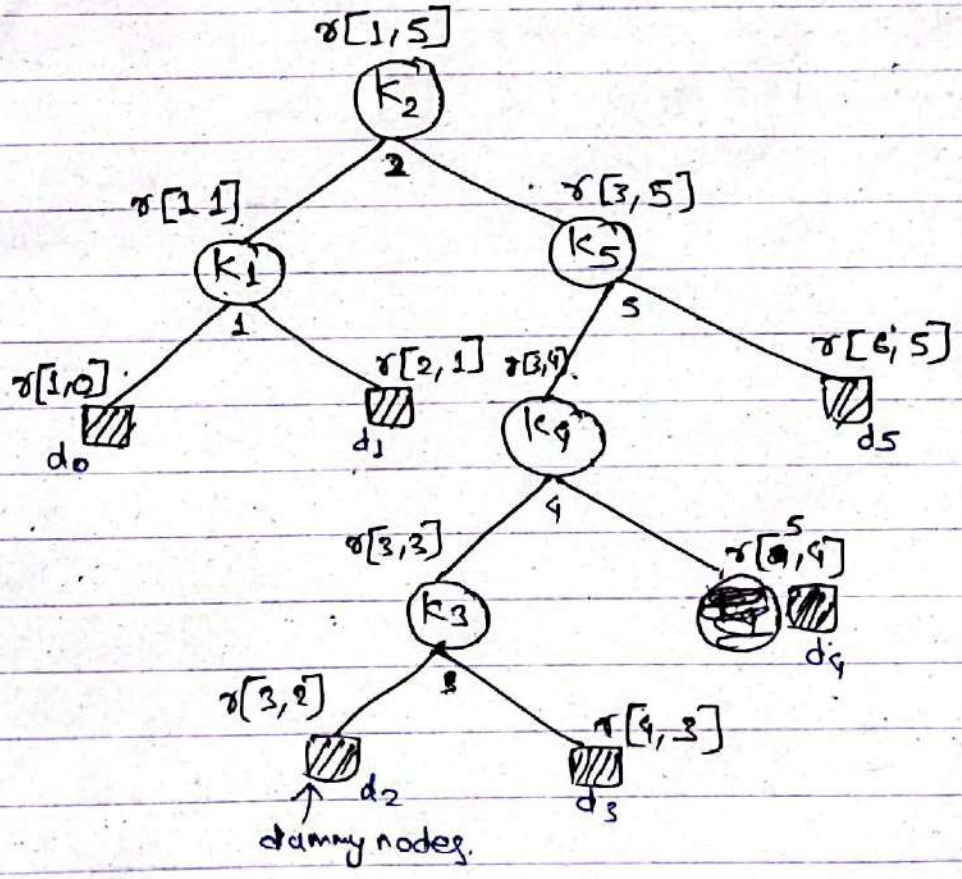
1st row = [row no | weight] matrix

2nd row = min(row no) + weight matrix

step 3: Construction of Root matrix

	1	2	3	4	5
1	1	1	2	2	2
2	-	2	2	2	4
3	-	-	3	4	5
4	-	-	-	4	5
5	-	-	-	-	5

step 4: Construct OBST

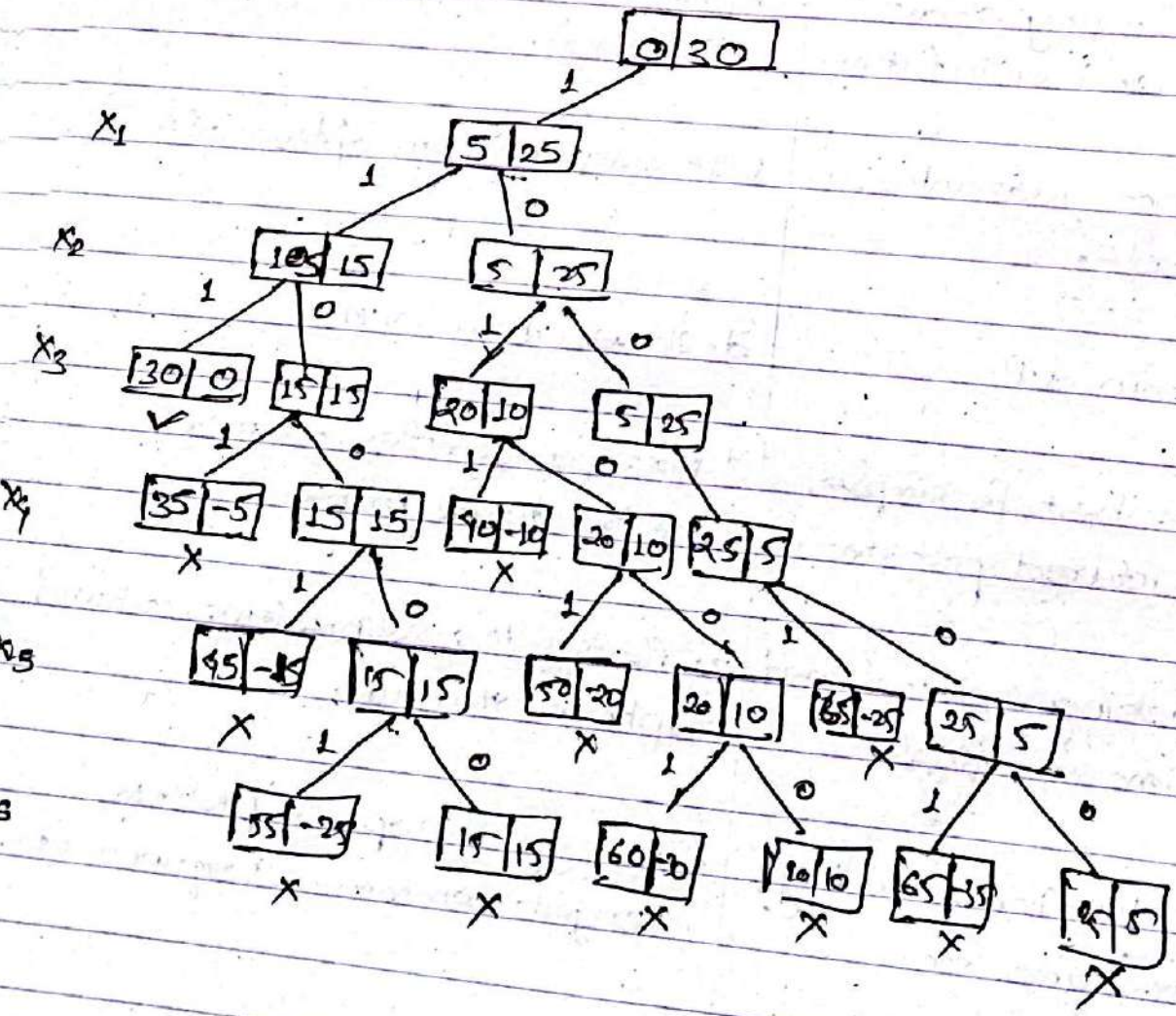


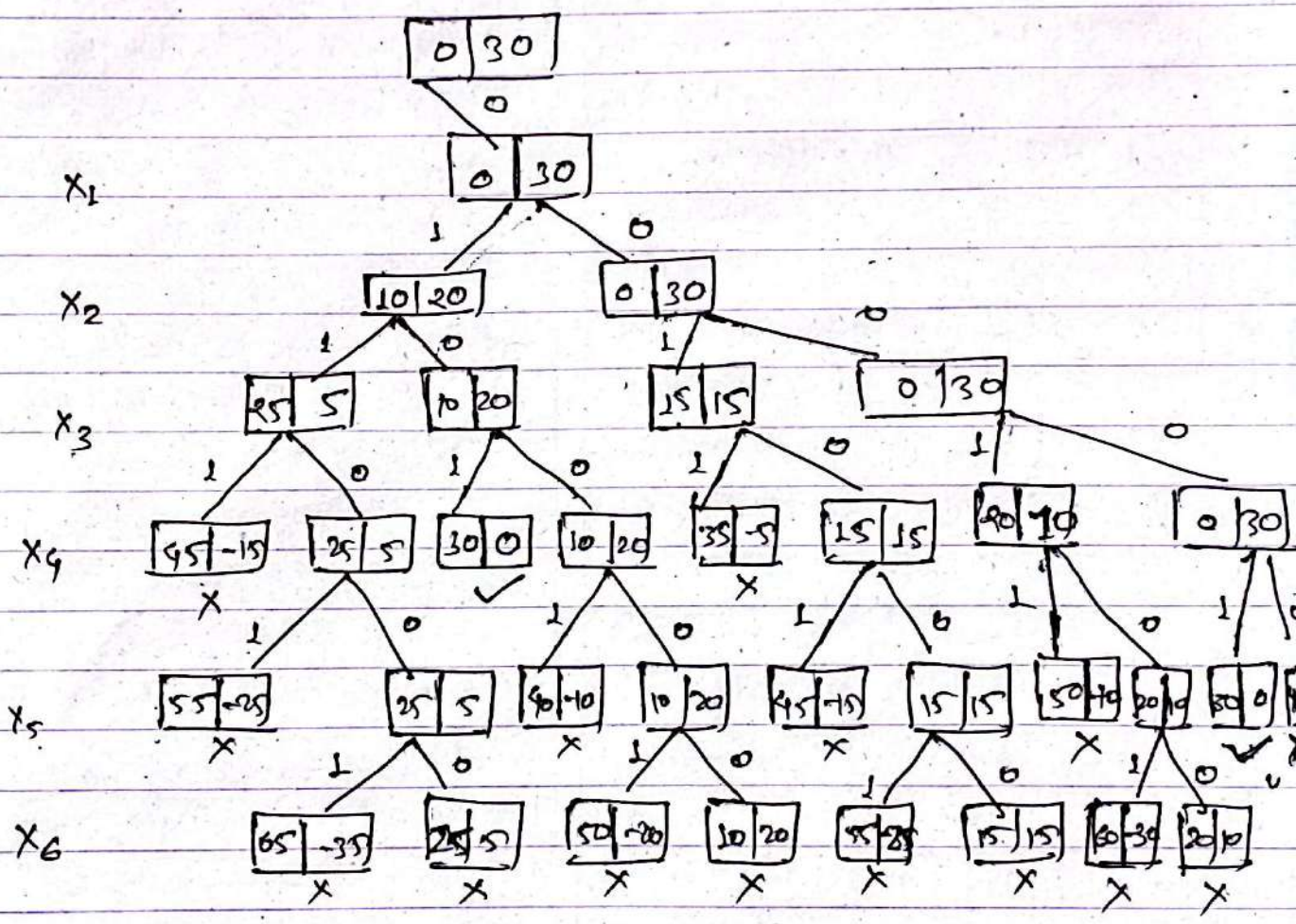
cost of OBST = 2.75

Greedy Algo	Dynamic Programming
a simple & st. forward algo that takes decision on basis of current available info without worrying about effect of decision in future.	a. optimization atg problem based on concept of solving sequences of sub-modules (i.e it. tries to figure out all possible solutions).
b It doesnot guarantee an optimal sol^n.	b It guarantee an optimal sol^n
c Faster than d.P	d. slower than Gr.A
d. They are best for simple, well-structured problems	d Best for complex, recursive, combinatorial problems.
e Backtracking is not allowed & decisions are final	e. Revisits the subproblems to build optimal structure.
f For e.g Huffman coding shannon fano etc.	f For e.g knapsack problem, Longest common subsequence etc.

Sum of subset.

Given a set $S = \{5, 10, 15, 20, 30, 40\}$ and $X = 30$. obtain subset sum using backtracking approach.





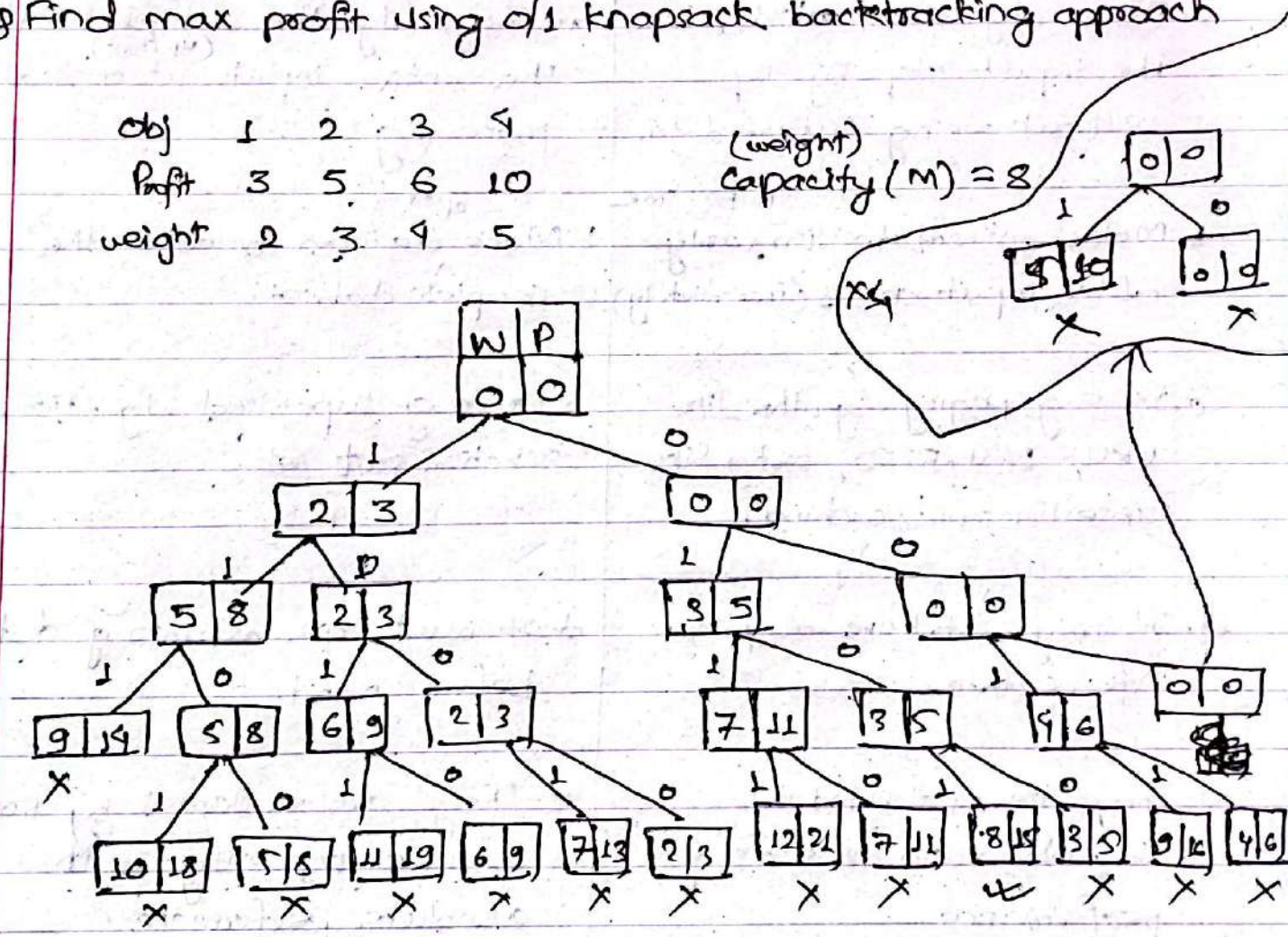
$S = \{111 \text{ or } 0101 \text{ or } 00001\}$ //

0/1 knapsack

Q Find max profit using 0/1 knapsack backtracking approach

Obj	1	2	3	4
Profit	3	5	6	10
weight	2	3	4	5

(weight)
Capacity (M) = 8



sum of subset

~~Knapsack profit~~ = 0101

Max profit = 5+10 = 15

Online Algo	Offline Algo
a. Those algorithm which receive the input step-by-step without seeing future data.	a. Those algorithm which receive (up front) the entire input at once before processing
b. Make optimal decision in real time only after input arrives (immediately)	b. Make optimal decisions based on the complete dataset.
c. For e.g paging algorithm like LRU, LRU, FIFO, sorting like insertion sort / caching.	c. For e.g supervised algorithm, selection sort etc.
d. It might not have all prior info of data.	d. It must have all info of data before hand.
e. They are evaluated by competitive ratio in terms of performance.	e. They are evaluated by time & space complexity with absolute performance.
f. It must adapt to new arriving data.	f. It can optimize with complete data.

Karger's Algorithm

→ A randomized method to find the min^{m} cut in a graph

→ Minimum cut is the smallest set of edges whose removal disconnects the graph from source to sink.

Algorithm

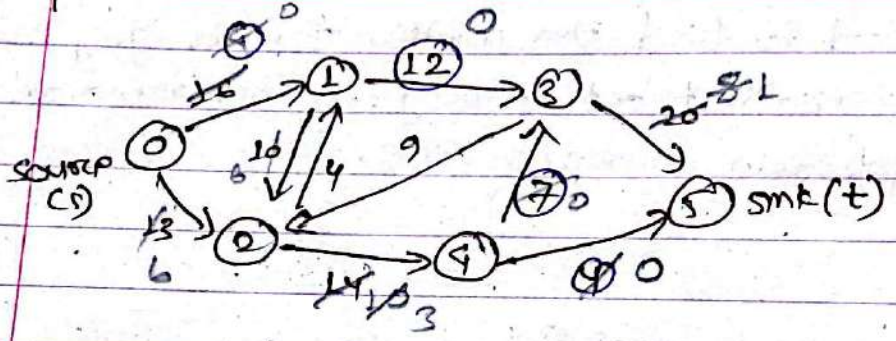
a. while more than 2 vertices remain
b. Pick a random edge.
c. Contract (merge) the connected vertices.
d. Remove any resulting self-loops
e. Remaining edges represent the cut
f. Repeat the process to improve success probability.

- Repeating algorithm increases accuracy.
- Time complexity $O(n^2 \log n)$
- used in graph partitioning, randomized algo, VLSI circuit, clustering etc.

Directed graph

0 = block

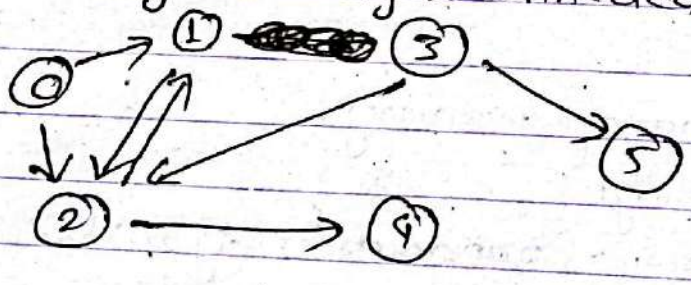
S = 0

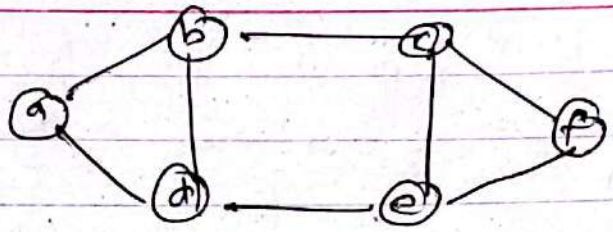


Path	Flow	
0-1-3-5	12	(Resubstract min ⁼ⁿ)
0-1-2-4-5	4	
0-2-4-3-5	7	

Maxflow = 23

Removing the edges connected with flow



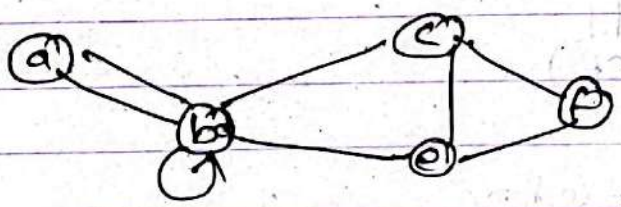


Step 1: Given graph $G = (V, E)$ where $V = \{a, b, c, d, e, f\}$
& $E = \{(a,b), (a,d), (b,c), (b,d), (d,e), (c,e), (c,f), (e,f)\}$

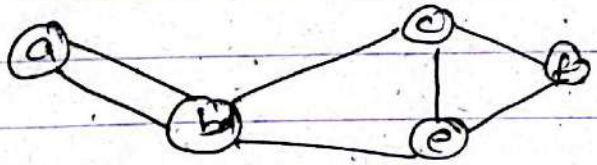
Here, no. of vertices $= 6 > 2$ (true)

let's pick up edges say, (b,d) & merge it

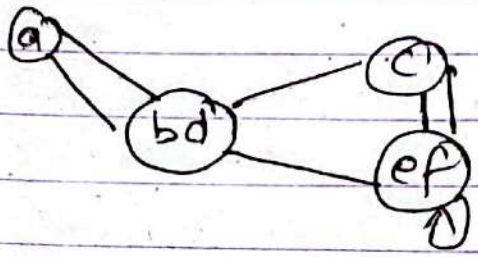
Step 2: all edges incident on b are incident on vertex i.e
$V = \{a, bd, c, e, f\}$



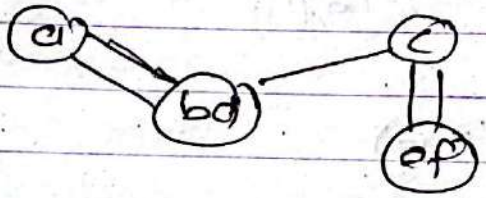
Step 3: Remove self-loop & check vertices.



no of vertices $= 5 > 2$ (true)
let's pick random edge say (e,f)

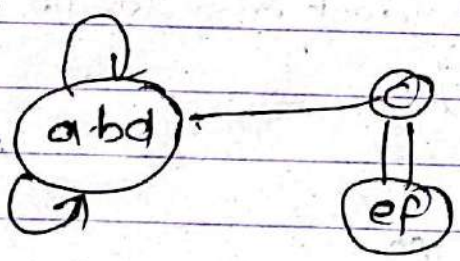


Remove selfloops & check vertices

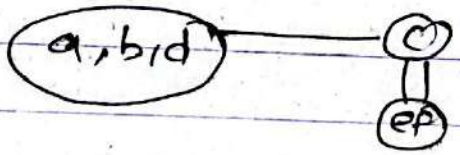


no. of vertices = 4 > 2 (true)

step4: Pick random edge say (a, bd)



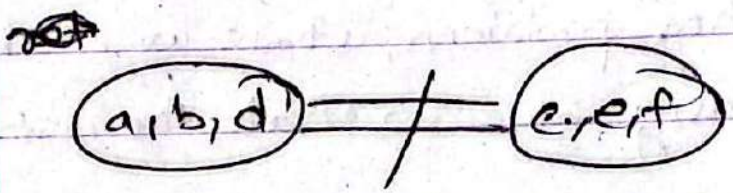
Remove selfloop & check vertices



no. of vertices = 3 > 2 (true

Step5: Pick random edge say (c, e, f)

no. of vertices = 2 > 2 (false


a, b, d
e, e, f

Ski-Rental Problem

⇒ It is an classic online algorithm problem where we need to decide whether to rent or buy skis so that the cost will be minimum.

⇒ Main goal - To minimize the total cost. (without knowing total duration)

⇒ Also known as Buy-Rent Problem.

Let's assume we are taking ski-lessons.
we have 2 choices
1. Rent skis for $1 per day
2. Buy skis for a fixed cost (e.g $10).

⇒ Decision has to be made whether to continue rent or buy at particular instance. and we need to give an algo that makes good decisions.

⇒ An offline strategy already know exactly how many days we'll skill before we make any decisions.,
i.e

$$\text{cost} = \begin{cases} \text{rent for } n \text{ days}, & \text{if } n < \text{total cost of ski} \\ \text{buy}, & n \geq \text{total cost of ski} \end{cases}$$

⇒ Implement online strategy so that it guarantees cost of online algorithm w.r.t skirental problem is always than twice the cost of buying.

let

y = cost of buying

k = no. of days of training

$$\text{min cost} = [k-1] + y$$

Here, $k \approx y$

$$= y - 1 + y$$

$$\text{online cost} = 2y - 1$$

$$\text{offline cost} = y$$

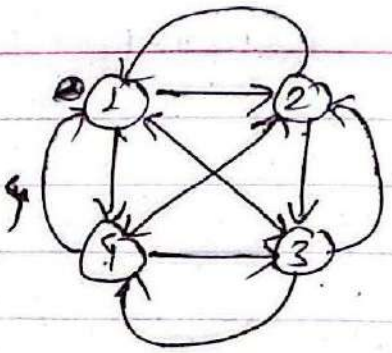
$$\text{competitive ratio} = \frac{\text{online cost}}{\text{offline cost}} = \frac{2y-1}{y} = 2 - \left(\frac{1}{y}\right)$$

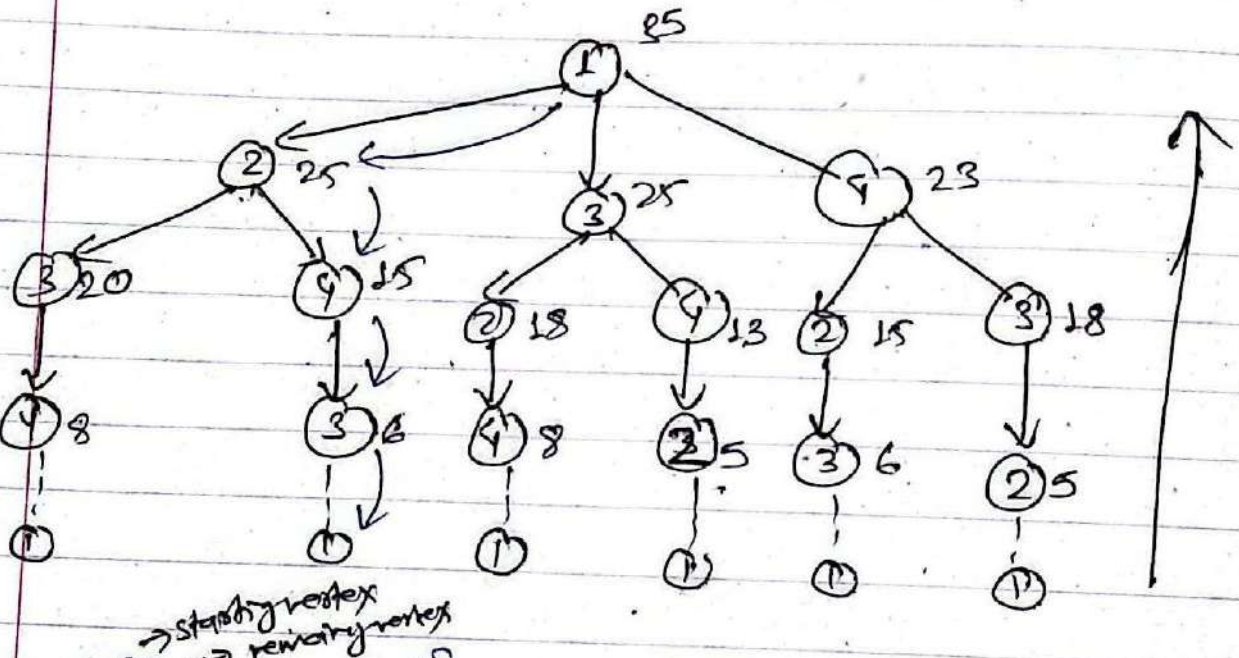
Thus, setting $k = y$ guarantees that we never have to pay more than twice the cost of offline strategy.

Paging and caching

- Paging is a memory management technique used to avoid fragmentation & efficiently use physical memory (RAM)
- Caching is a technique used to speed up data access by storing frequently used data in faster storage (like CPU cache or memory)

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0





$g(i,S) = \min_{j \in S} \{ c_{ij} + g(j, S-\{j\}) \}$

$g(4,\phi) = \min_{j \in \phi} \{ c_{41} + g(\phi, \phi-\{\phi\}) \} = 8$

$g(3,\phi) = \min_{j \in \phi} \{ c_{31} + g(\phi, \phi-\{\phi\}) \} = 6$

$g(4,\phi) = 8$

$g(2,\phi) = \min_{j \in \phi} \{ c_{21} + g(\phi, \phi-\{\phi\}) \} = 5$

$$g(4,\phi) = g(4,1) = 8$$

$$\begin{bmatrix} S-\{j\} \\ = \{4\} - \{\phi\} \\ = 1 \end{bmatrix}$$

$$g(3,\{4\}) = \min_{j \in 4} \{ C_{34} + g(4, 4-\{\phi\}) \}$$

$$= \min_{j \in 4} [12 + 8] = 20$$

$$g(4,\{3\}) = \min_{j \in 3} [C_{43} + g(3, 3-\{\phi\})]$$

$$= \min [9 + 6] = 15$$

$$g(2,\{4\}) = \min_{j \in 4} [C_{24} + g(4, 4-\{\phi\})]$$

$$= \min_{j \in 4} [10 + 8] = 18$$

$$g(4,\{2\}) = \min_{j \in 2} [C_{42} + g(2, 2-\{\phi\})]$$

$$= \min_{j \in 2} [8 + 5] = 13$$

$$g(2,3) = \min_{j \in 3} [C_{23} + g(3, 3-\{\phi\})]$$

$$= \min_{j=3} [9 + 6] = 15$$

$$g(3,2) = \min_{j \in 2} [C_{32} + g(2, 2-\{\phi\})]$$

$$= \min_{j \in 2} [13 + 5] = 18$$

$S - \{3\} = \{3,4\} - \{3\} = \{4\}$

$\{2,4\} - \{2\} = \{4\}$

$g(2,\{3,4\}) = \min_{j \in \{3,4\}} \begin{cases} [C_{23} + g(3,\{4\})] = 9+20 = 29 \\ [C_{24} + g(4,\{3\})] = 10+15 = 25 \end{cases}$

$= \min_{j \in 4} = 25$

$g(3,\{2,4\}) = \min_{j \in 2} \begin{cases} [C_{32} + g(2,\{4\})] = 13+18 = 31 \\ [C_{34} + g(4,\{2\})] = 12+13 = 25 \end{cases}$

$= \min_{j \in 4} = 25$

$g(4,\{2,3\}) = \min_{j \in 2} \begin{cases} [C_{42} + g(2,\{3\})] = 8+15 = 23 \\ [C_{43} + g(3,\{2\})] = 9+18 = 27 \end{cases}$

$= \min_{j \in 3} = 23$

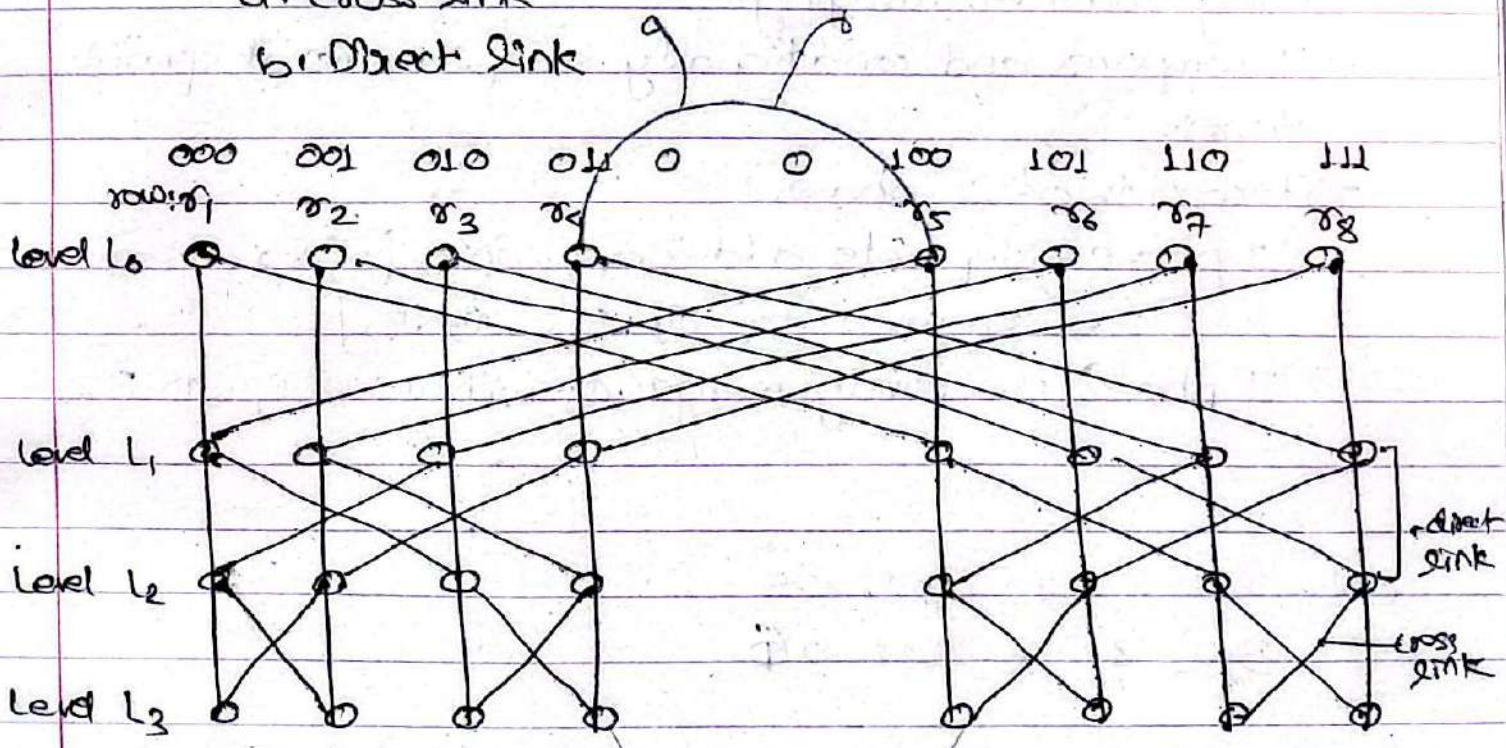
$g(1,\{2,3,4\}) = \min_{j \in \{2,3,4\}} \begin{cases} [C_{12} + g(2,\{3,4\})] = 10+25 = 35 \\ C_{13} + g(3,\{2,4\}) = 15+25 = 40 \\ C_{14} + g(4,\{2,3\}) = 20+23 = 43 \end{cases}$

optimal Route $= 1 \to 2 \to 4 \to 3 \to 1$

Hypercube Algorithms

Butterfly Network

- It is a multi-stage interconnection network (MIN) derived from Hypercube which supports parallel communication and fast data routing.
- It is named for its butterfly like diagram that illustrate its structure
- It is an embedding technique which solves sorting data concentration, prefix computation, selection etc.
- There is a restrictions i.e. ~~each~~ one node in a Butterfly Network consists of one processor.
- Each node is linked with two nodes in the next stage.
- It consists of two links
  a. Cross link
  b. Direct link



1 node = 1 processor

32 nodes = 32 processors

8 column $= 2^3$

$= 3$ dimension

Processor (P) $= (d+1)\,2^d$

$= (3+1)\,2^3$

$= 4 \cdot 2^3$

$= 32$

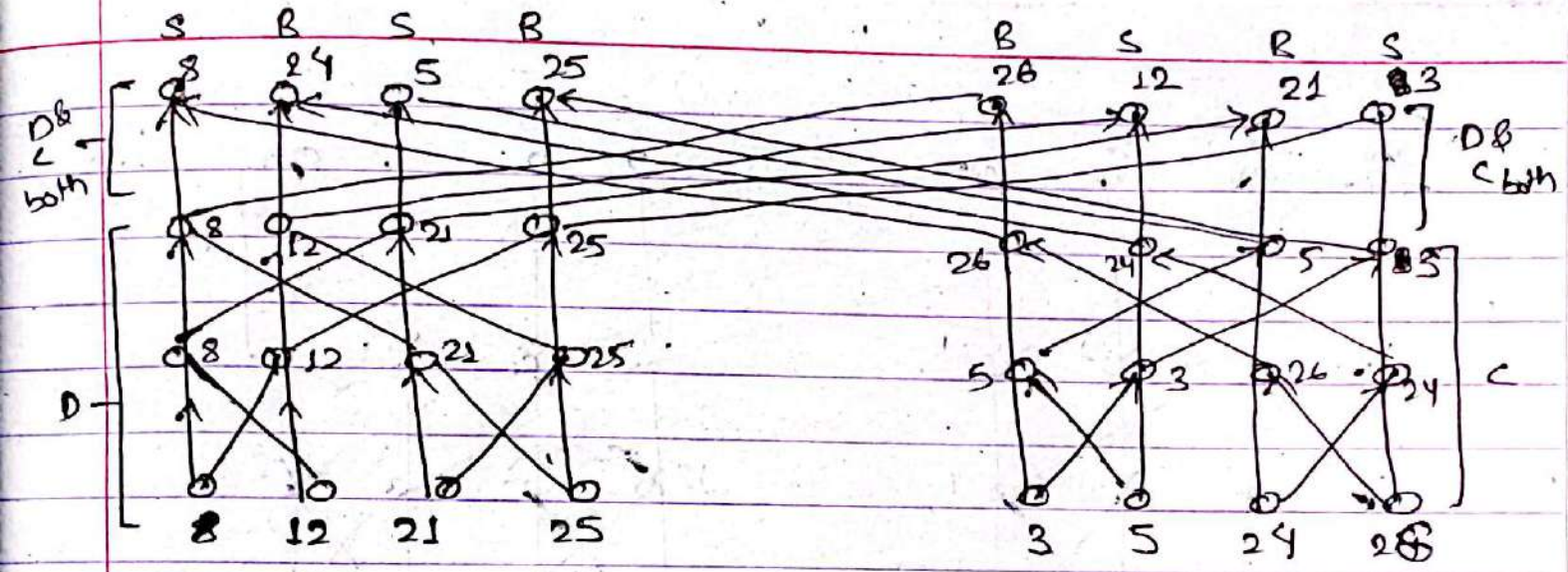
Total links = direct links + cross link
= 24 + 24 = 48

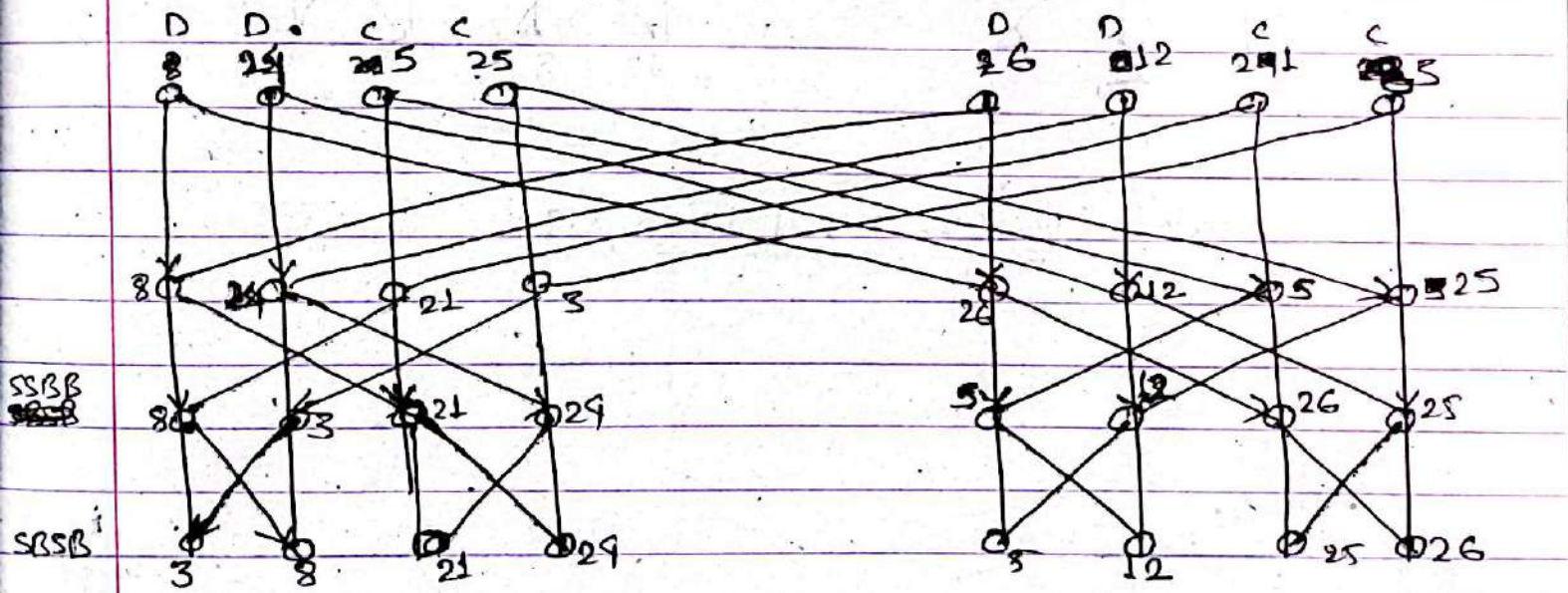
Total links $= d \cdot 2^{d+1}$
$= 3 \cdot 2^{3+1}$
$= 3 \cdot 2^{4}$
$= 48$

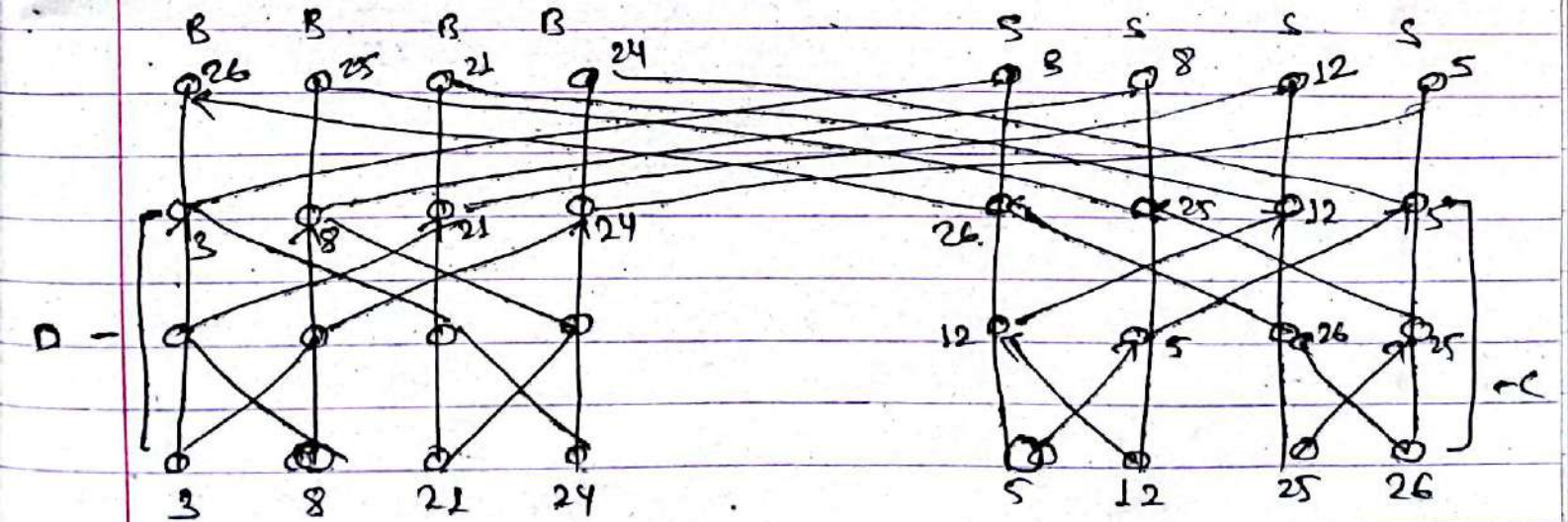
Odd Even Merge Sort (using Butterfly Network)
- A parallel sorting algorithm that merges two sorted halves using odd-even merging.
- It compares and conditionally swaps values at specific stages.
- It consists of 2 phases:
  - I phase: Separate odd & even into $[x_1 \& x_2$ given sequence to $O_1 \cdot E_1, O_2 \cdot E_2]$
  - II phase: Recursively merge O_1 with O_2 & E_1 with E_2

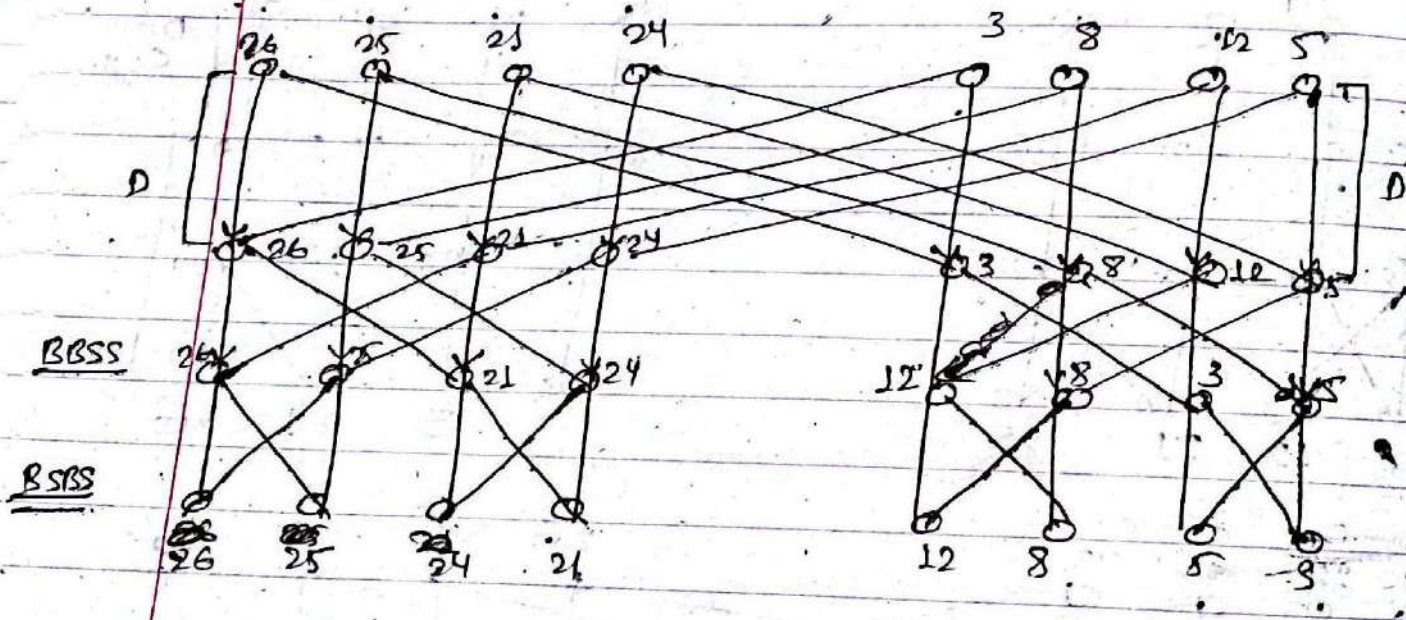
eg: $x_1 = 8 \quad 12 \quad 21 \quad 25$
$x_2 = 3 \quad 5 \quad 24 \quad 26$

↑↓↑↓


S B S B
8 24 5 25
DB both
8 12 21 25
8 12 21 25
D
8 12 21 25
B S B S
26 12 21 3
DB both
26 24 5 5
5 3 24 24
C
3 5 24 26


D D C C
8 24 5 25
8 21 3
SSBB
8 3 21 24
SBSB
3 8 21 24
D D C C
26 12 21 5
26 12 5 25
5 26 25
5 12 25 26


B B B B
26 25 21 24
3 8 21 24
D
3 8 21 24
S S S S
8 12 5
26 25 12 5
12 5 26 25
C
5 12 25 26



Sorted: 26, 25, 24, 21, 12, 8, 5, 3

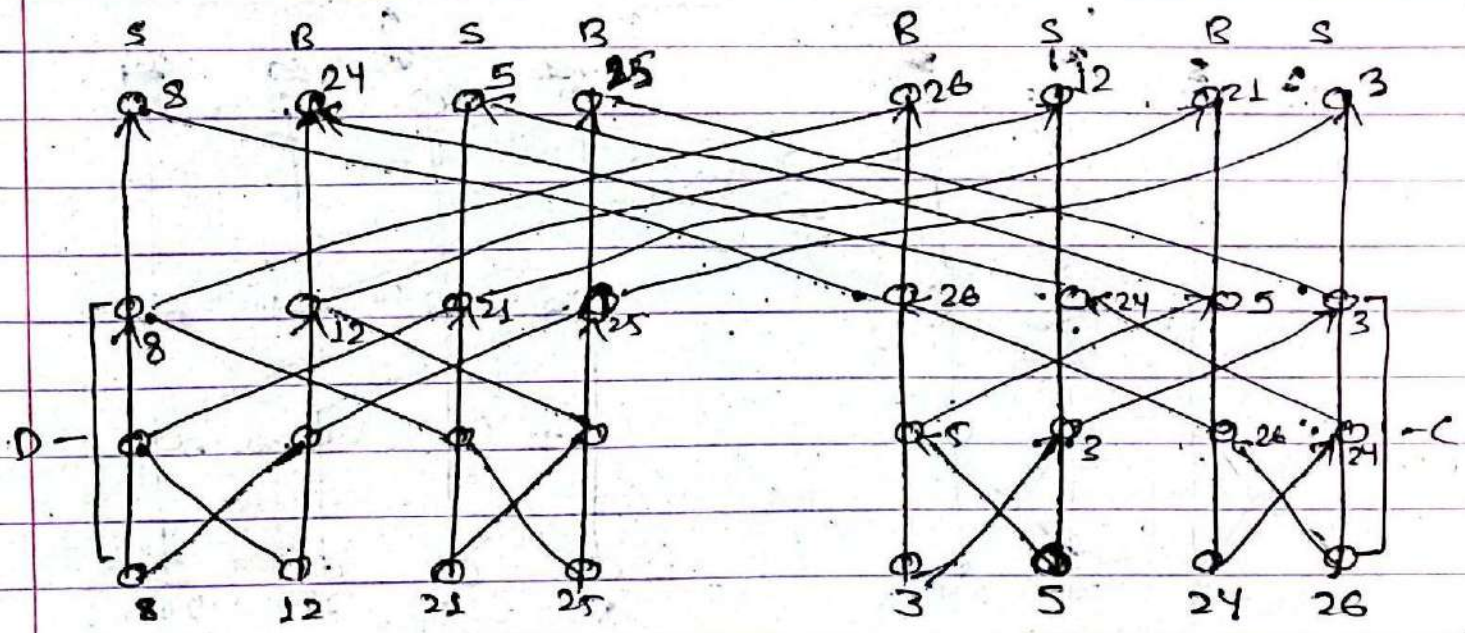
Bitonic Sort

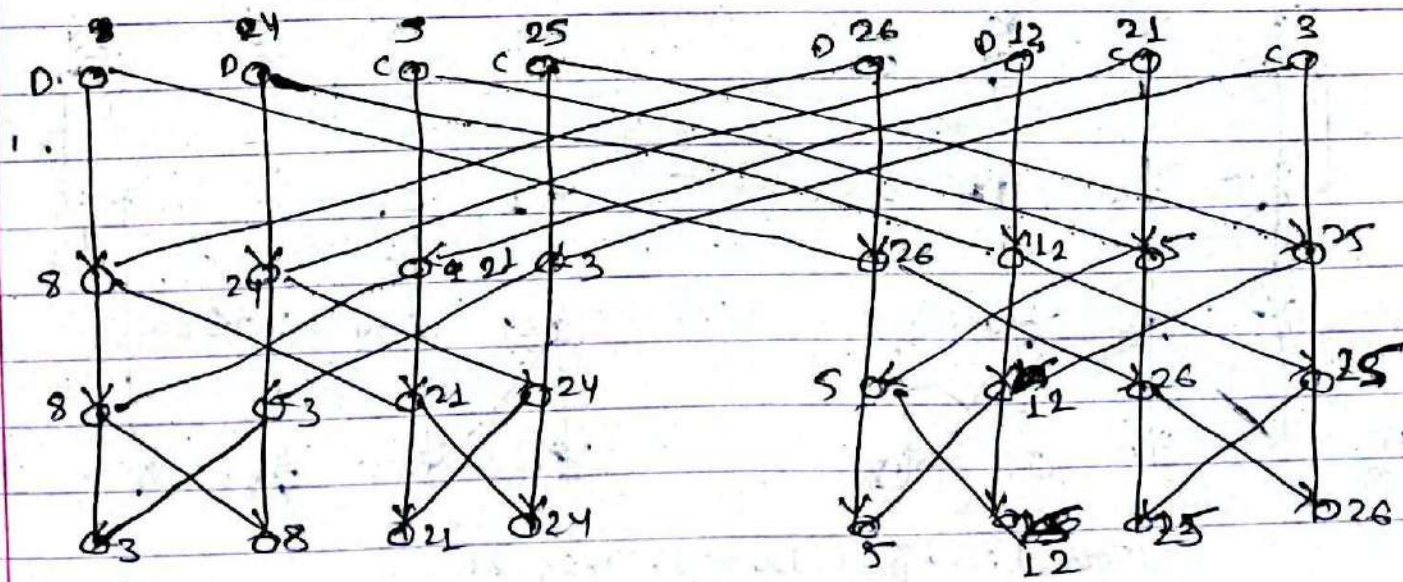
- A parallel sorting ~~algorithm~~ algorithm that implements bitonic sequence and bitonic merge.
- Bitonic sequence is a sequence that increases, then decreases or vice-versa.
- Bitonic merge is a process that turns bitonic sequence into sorted one by recursively comparing and swapping elements.

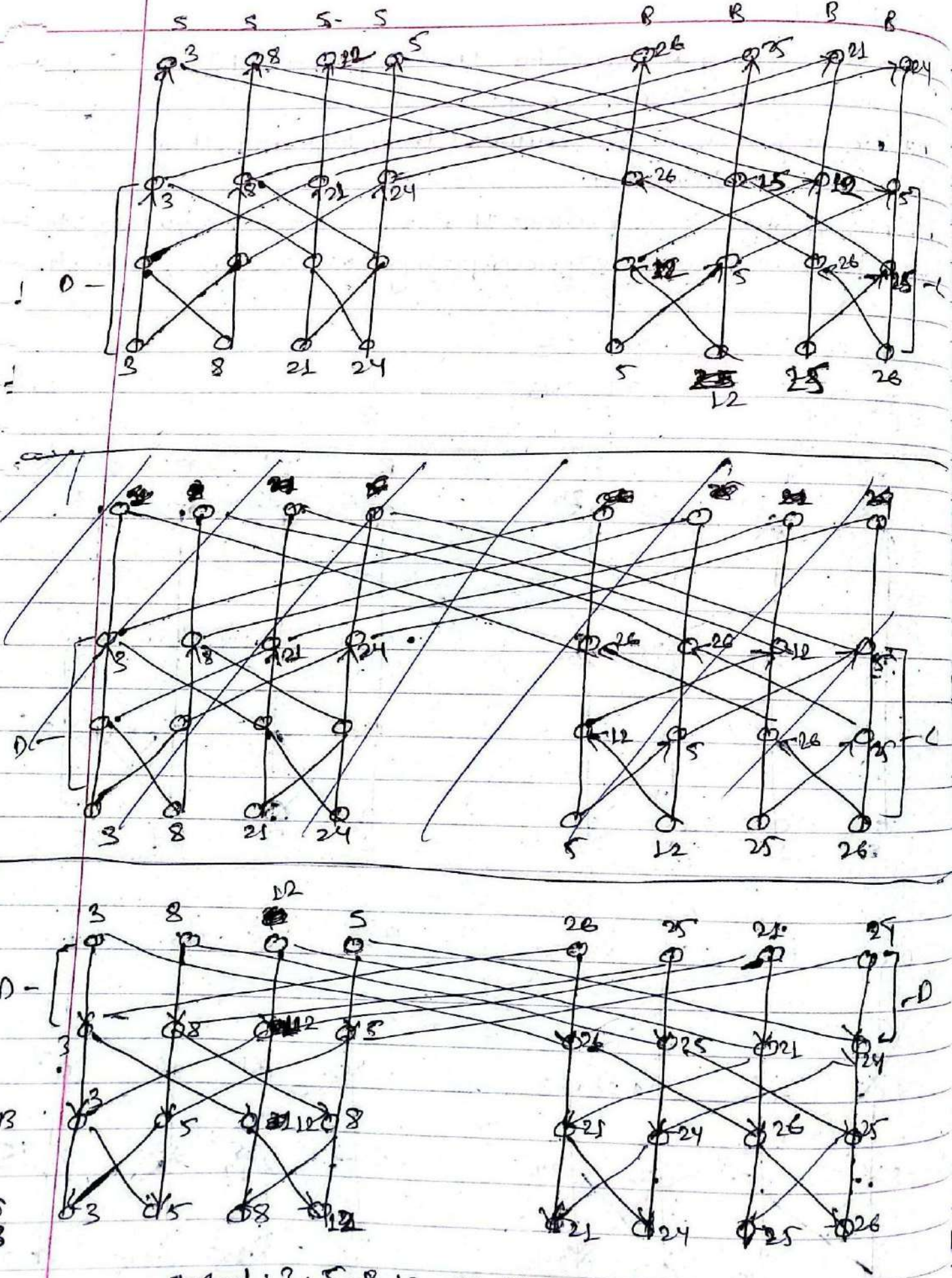
e.g

$x_1 = 8 \quad 12 \quad 21 \quad 25$

$x_2 = 3 \quad 5 \quad 24 \quad 26$






S S S S
B B B B
3 8 12 5
26 25 21 24
3 8 21 24
26 25 12 5
5 12 25 26
A
B
C
D
3 8 21 24
26 25 12 5
5 12 25 26
A
B
C
3 8 12 5
26 25 21 24
3 5 8 12
21 24 25 26
D
SSBB
SBSB
Sorted: 3, 5, 8, 12, 21, 24, 25, 26

Fundamental Algorithms

a. Broadcasting

b. Prefix computation

c. Data Concentration

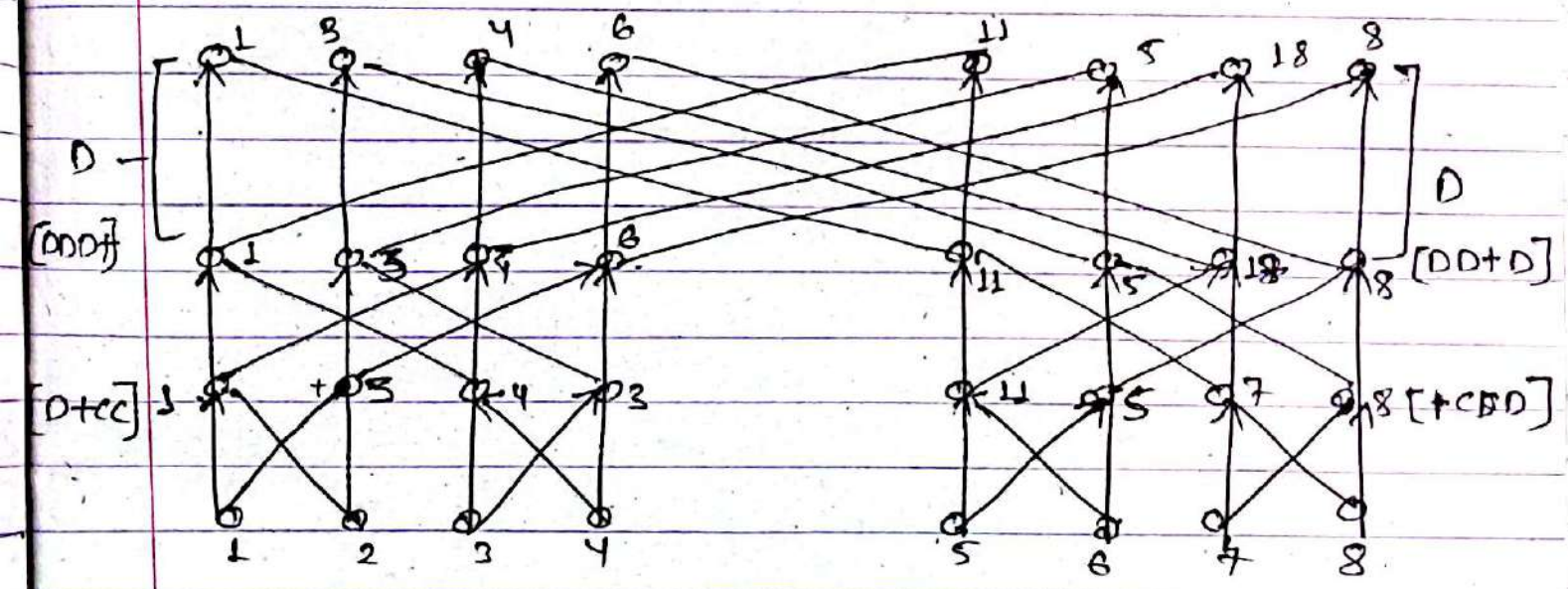
b. Prefix Computation.

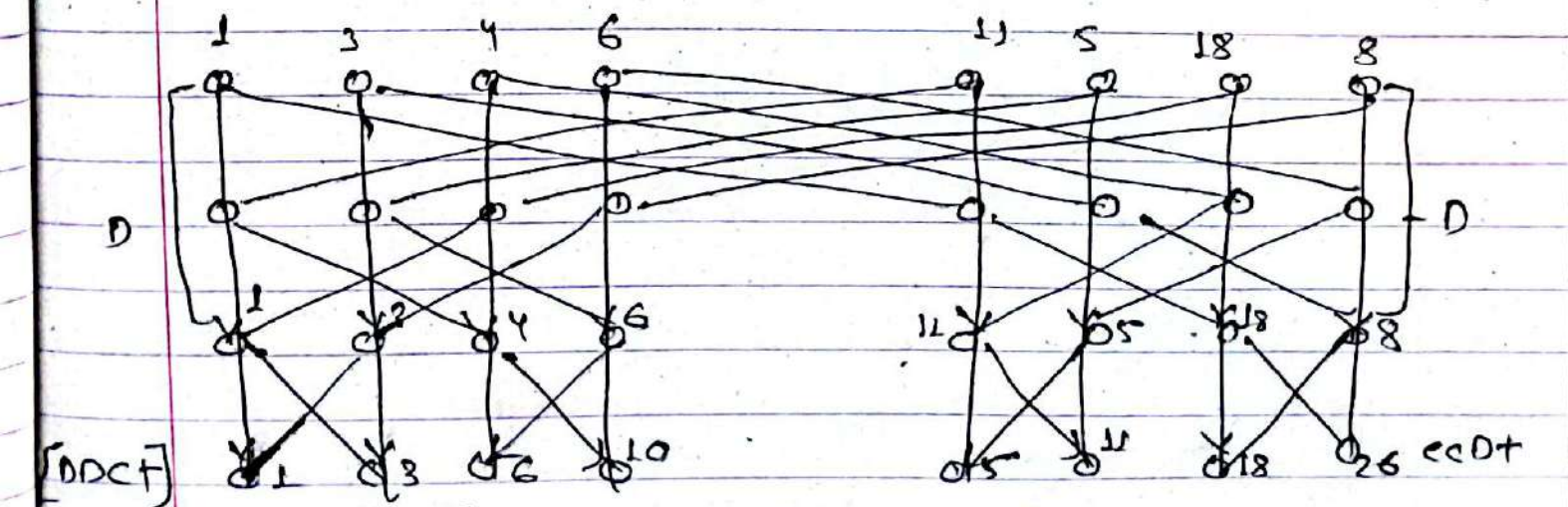
We use prefix computation using Butterfly network

1 + 2 + 3 4 5 6 7 8

⇒ 1 3 6 10 5 11 18 26

sol 2





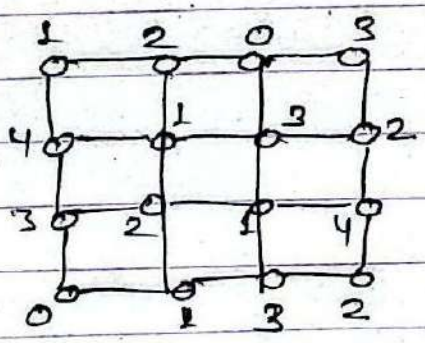
sorted sol 2 : 1 3 6 10 5 11 18 26

Fundamental Algorithms

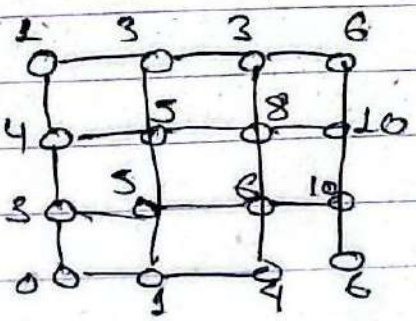
a. Broadcasting
b. Prefix computation
c. Data concentration

b. Prefix computation

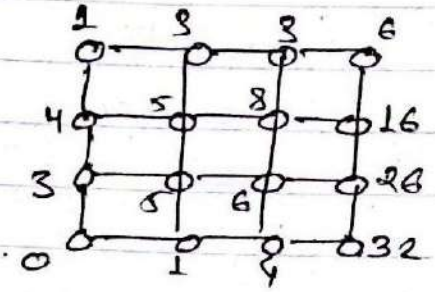
eg:



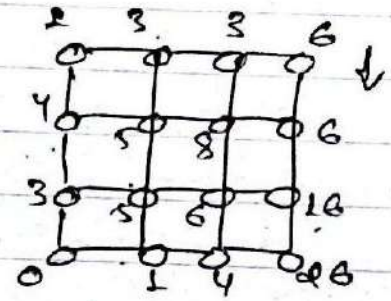
Step1: Prefix computation in each row



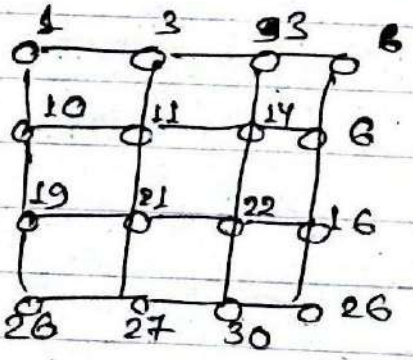
step2: only last column computes its prefix



step3: shift the prefix of last column one down



step 4 Broadcast the last prefix of step 3 to each row.



Here,

prefix calculation $\rightarrow \sqrt{p}$

last col $\hat{=}$ prefix calculation $= \sqrt{p}$

shifting 1

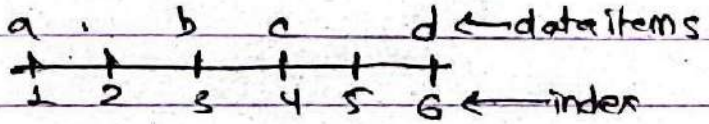
updating 1

Broadcasting $\rightarrow \sqrt{p}$

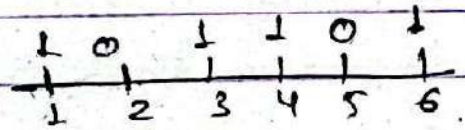
$3\sqrt{p} + 2$

$= O(\sqrt{p})$

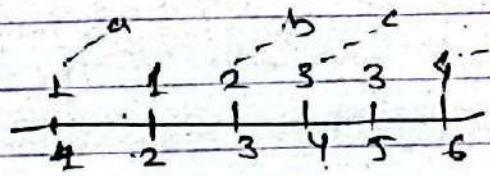
Data concentration

e.g 

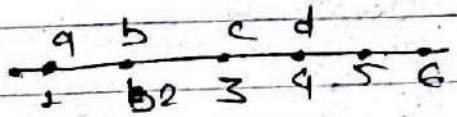
→ data items < processor (satisfies)

→ 

Linear representation
data → 1
No data → 0

→ 

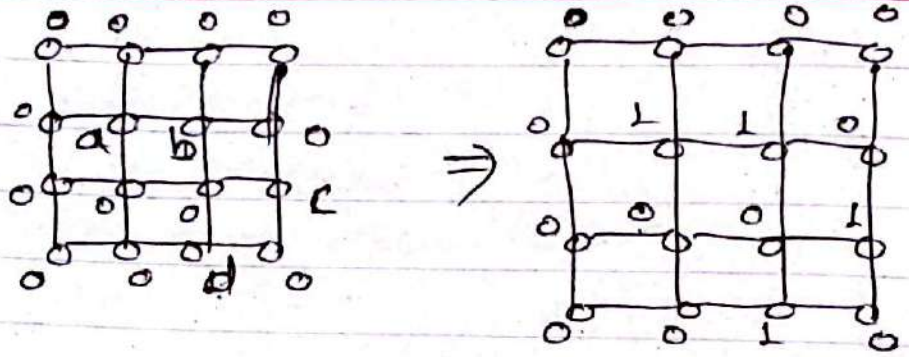
[Prefix]

→ 

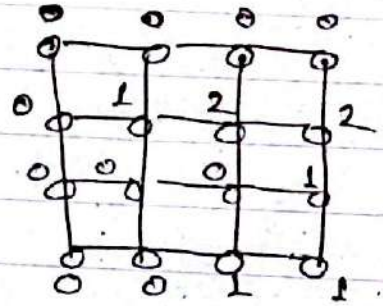
[Routing]

Complexity Analysis
Prefix calculation - P
Routing - $\frac{P}{2P}$

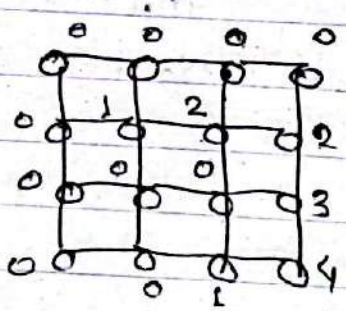
$\therefore O(P)$ //



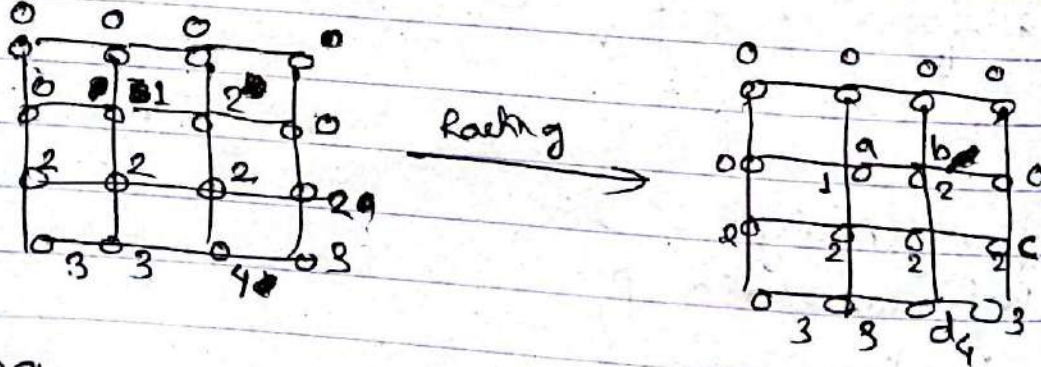
step 1: Prefix computation in each row



Step 2: last col^n prefix computation



step 3: shift the last prefix of last col^n one down

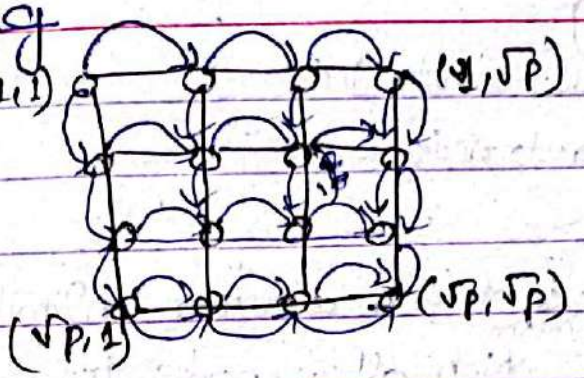


Step 4: Routing

linear takes (16-1) steps ≈ (n-1) steps

1 linear 16

a Broadcasting



- Broadcasting in mesh means sending a message from one source node to all other neighbouring node (i.e right & down neighbours).
- First message lets suppose (i,j) is broadcast along the first row in phase 1
- Once the first row has the message, each node in first row broadcasts down its column in phase 2.
- This algorithm takes $(2\sqrt{p}-1)$ steps where p is no. of processors (nodes).
- If the mesh has p nodes arranged in $\sqrt{p} \times \sqrt{p}$ form
- Then each dimension has $\sqrt{p}$ nodes.

- Broadcasting takes $\sqrt{p}-1$ steps to spread across first row and $\sqrt{p}-1$ to spread across second row
- Total time steps $= 2(\sqrt{p}-1)$

4×4 mesh has 16 nodes. So,

$$\text{time} = 2(\sqrt{p}-1)$$
$$= 2(\sqrt{16}-1)$$
$$= 2(4-1)$$
$$= 6 \text{ steps}$$

6 steps is required in ideal case for broadcast message to reach all nodes for row then column broadcast in mesh

Types of Broadcasting

a. Broadcasting without constraints
b. Broadcasting with constraints

a. Broadcasting without constraints assumes unlimited bandwidth & full-duplex communication links which allow nodes to send message simultaneously to multiple neighbours.
- There will be no collisions, no redundancy, no buffer limitations.
- For e.g Row-col Broadcast in Mesh & Hypercube.
- It provides max^m parallism. which results in fast communication

b. Broadcasting with constraints considers real-world-limits like limited bandwidth, half-duplex lines or nodes which allow nodes to send message only to one neighbour at a time.
- Constraints slow down broadcasting by forcing serialized & scheduled message passing.
- There will be engergy/power constraints, redundancy, limited buffer at nodes.
- For e.g Scheduled broadcast, tree-based broadcast, pipelined broadcast etc.
- It provides partial parallism which results in slow communication.

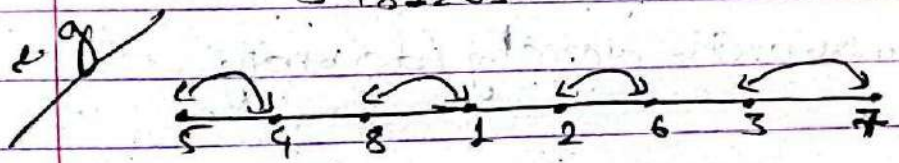
Sorting in Linear Array [Odd Even Transposition]

- It is a parallel sorting algorithm which works through a sequence of compare and swap steps that alternate between odd and even indexed pairs.

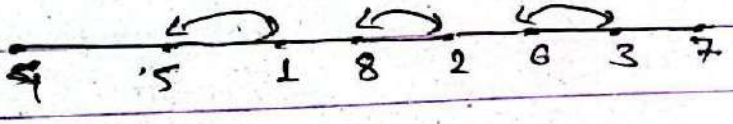
Algorithm

```
for (i=1; i<=p; i++)
  if (i%2==0){ -> even
    compare & exchange keys at process 2i & 2i+1
  else
    compare & exchange keys at process 2i & 2i-1 }
```

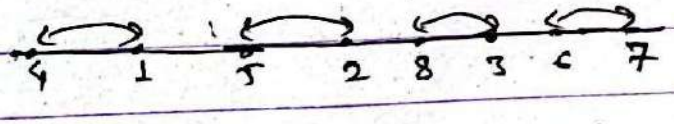
5 4 8 1 2 6 3 7



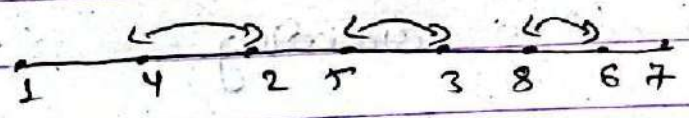
step 1 :



step 2 :



step 3 :



step 4 :

step 5 :

Sorting in Mesh (shear sort)

Algo

for (i=1 ; i≤

e.g.

15	12	8	32
7	18	6	17
2	16	19	25
18	11	5	3

Iteration – I : Perform row wise ascending / descending

8	12	15	32
17	18	7	6
2	16	19	25
18	11	5	3

Iteration – II : Perform col^n wise ascending

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

Iteration III: Perform row wise asc/desc.

2 3 5 11

12 8 7 6

13 15 17 25

32 19 18 16

Iteration IV: Perform col wise asc.

2 3 5 6

12 8 7 11

13 15 17 16

32 19 18 25

Iteration V: Perform row wise asc/dc

2 3 5 6

12 11 8 7

13 15 16 17

32 25 19 18

a. Least Recently used (LRU)

String : 3 2 1 3 4 1 6 2 4 3 4 2 1 4 5 2 1 3 4

string :	3	2	1	3	4	1	6	2	4	3	4	2	1	4	5	2	1	3	4
f_1	3	3	3	3	3	3	6	6	6	3	3	3	1	1	1	2	2	2	4
f_2		2	2	2	4	4	4	2	2	2	2	2	2	2	5	5	5	3	3
f_3			1	1	1	1	1	1	4	4	4	4	4	4	4	4	1	1	1
Action	x	x	x	✓	x	✓	x	x	x	x	✓	✓	x	✓	x	x	x	x	x

page fault = 14
page hit = 5

fault ratio = $\frac{14}{19} \times 100\%$

hit ratio = $\frac{5}{19} \times 100\%$

b. Last In Fast out
- Replaces most recently loaded page.

string: 7012030423032120 1701

String	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
f_1	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
f_2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_3			1	2	2	3	3	4	2	3	3	3	2	1	2	2	1	1	1	1
Action	X	X	X	X	✓	X	✓	X	X	X	✓	X	X	X	X	✓	X	✓	✓	

Page fault = 13 f.ratio = 13/20 × 100%.

hit = 7 Hit ratio = 7/20 × 100%.

c Longest forward distance (LFD)

- Also known as optimal or belady's algorithm
- It uses a look-ahead to decide which page to evict (the one with longest future use distance).

string: 7012030423032120 1701

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
String	7	0	1	2	0	3	0	4	2	3	0	2	2	1	2	0	1	7	0	1
f_1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
f_2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
f_3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
Action	X	X	X	X	✓	X	✓	X	✓	✓	X	✓	✓	X	✓	✓	✓	X	✓	✓

page fault = 9 f.ratio = 9/20 × 100%.

hit = 11 Hit ratio = 11/20 × 100%.

Complexity classes

class P

- Problems that can be solved by deterministic TM in polynomial time.
- They are obviously tractable
- easier to solve & verify
- eg addition, substraction, multiplication.