Amortized cost Analysis:

→ also known as average cost analysis.
→ method of analyzing cost associate with D.S. that average the worst operation over time
→ overall performance of an algo can't be justified by single worst case.
→ Hence amortized cost analysis is needed.
→ It averages out the costly operation in the worst cases

→ There are 3 main types of amortized analysis:
  * Aggregate Method.
  * Accounting Method.
  * Potential Method.

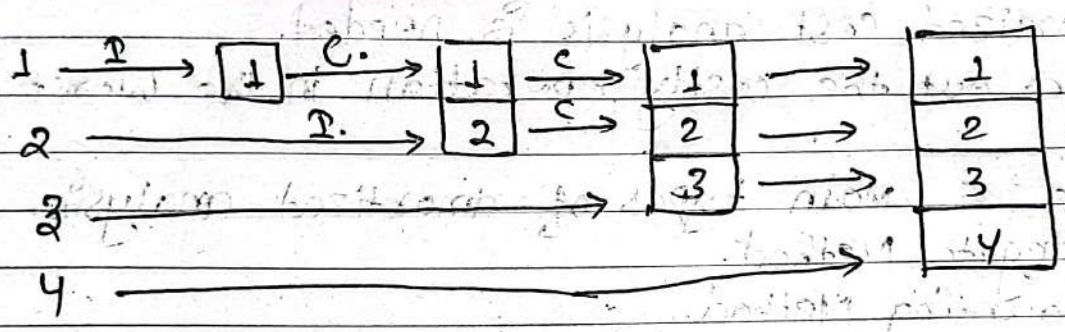
Aggregate Method: (Also called Averaging Method)

→ There are 2 steps.

→ We must show, sequence of 'n' operation takes T(n) time.

→ Then, we show each operation take T(n) on average.

→ Consider hash table:



Index (i)	1	2	3	4	5	6	7	8	9	10
size (i)	1	2	4	4	8	8	8	8	16	16
cost (c_i)	1	2	3	1	5	1	1	1	9	1

c_i = cost involved = copy + insert.

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

$$c_i = n + \sum_{j=0}^{\lceil \log z_i \rceil} 2^j$$

it can be generalized as:

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

Average cost = $O(n)/n = 1$.

With table doubling:

Item	Size	insert cost	Copy cost	Doubling cost	Total cost.
1	1	1	0	0	1
2	2	1	1	1	3
3	4	1	2	4	7
4	4	1	0	0	1
5	8	1	4	8	13
6	8	1	0	0	1
7	8	1	0	0	1
8	8	1	0	0	1
9	16	1	8	16	25

Here C_i = insert + copy + doubling.

$$= n + \sum_{i=0}^{n} 2^i + \sum_{j=0}^{n} 2^j$$

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

$$= 4n - 3 = O(n)$$

$\therefore$ Average cost = $O(n)/n = 1$.

Accounting Method:

→ we should first guess amortized cost ($\hat{c_i}$) such that

$\hat{c_i} - c_i \geq 0$; c_i is the actual cost.

if $\hat{c_i} - c_i > 0$ put the remaining cost to bank.

else borrow needed cost from bank.

Guess $\hat{c_i} = 6$.

index (i)	size(N)	$\hat{c_i}$	c_i (Actual cost)	Bank Balance
1	1	6	1	5 (6-1)
2	2	6	4	7 (5+2)
3	4	6	7	6 (7-1)
4	4	6	1	11 (6+5)
5	8	6	13	4 (11-7)
6	8	6	1	9 (5+4)
7	8	6	1	14 ~~+5 (10+5)~~
8	8	6	1	19
9	16	6	25	0

Table doubling occur when: $(\frac{n}{N})$ load factor ≥ 1.

Table half occur when $(\frac{n}{N} - 1) = 0.25$.

Bank Balance $\leq 4\left(|n - \frac{N}{2}| - 1\right)$.

Case I: cost of resize (table double)

$n = N+1$.

$= 4(|N+1 - \frac{N}{2}| - 1)$

$= 4(\frac{2+N-2}{2})$

$= 2N$

Case II cost of resize (table half)

$n = \frac{N}{4} - 1$

$= 4(\frac{N}{4} - 1 - \frac{N}{2}| - 1)$

$= 4(|\frac{N-4-2N}{4}| - 1)$

$= N$.

Potential Method :

→ we need to define a potential function.

$\Phi(D_i)$

$\Phi(D_0) = 0$ initial state

$\Phi(D_i) > 0$ final state.

Potential Difference $\Delta(\Phi(D_i)) = \Phi(D_i) - \Phi(D_{i-1})$

$\Delta(\Phi(D_i)) > 0$ put in bank.

$\Delta(\Phi(D_i)) < 0$ borrow from bank.

$$\hat{C}_i = C_i + \Delta(D_i)$$
$$= C_i + \Phi(D_i) - \Phi(D_{i-1})$$

Neglecting table doubling;

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

Not neglecting table doubling:

$$C_i = \begin{cases} 3i-2 & \text{if } (i-1) \text{ is exact power of 2.} \\ 1 & \text{otherwise,} \end{cases}$$

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

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

we take table doubling;

case I

$\hat{C}_i = 3i - 2 + \Phi(D_i) - \Phi(D_{i-1})$

$= 3i - 2 + 2i - 2^{\lceil \log i \rceil} - [2(i-1) - 2^{\lceil \log i-1 \rceil}]$

$= 3i - 2 + 2i - 2i + 2 - 2i + 2 + i - 1$

$= 2i + 1$

Calculate

$= 3$

case II

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

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

$=$ calc.

$= 3.$

$=$

Las vegas vs Monte-carlo:

Date
Page

Randomized Algorithm:

- A probabilistic algo is an algo where the result and/or the way the result is obtained depend on chance.

- It makes use of randomizer (random no. generator) to make decision in algo.
- The Output and/or execution time could also differ from run to run for same input.
- Can be categorized into:
  1. Las Vegas algo.
  2. monte Carlo algo.

Las vegas algorithm.

- uses the concept of randomization.
- generates always same output for same input
- execution time is characterized as a random Variable
- eg. randomized quicksort.
- It guarantees the correctness of the soln.
- doesn't guarantee the upper bound.

Monte Carlo algorithm:

- output might be different from run to run for same input data set.
- doesn't guarantee the correctness of the solution
- eg: Primility testing using miller Rabin's algo.
- guarantees the upper bound.
- execution time is fixed.

1.2. Advanced Algo design Technique

Greedy algo:

- Simplest and straight forward method
- take decision on the basis of current available info without worrying about the effect of the decision in future
- It works in phases.

At each phase:

* you take the best you can get right now, without worrying about future consequences.
* you always choose local optimum leading to global optimum.

Greedy

Greedy algo = Cost minimization — Profit maximization.

Huffamman Coding, Channon fano algo, Job scheduling etc. are eg.

Huffman Coding: - David A Huffman.

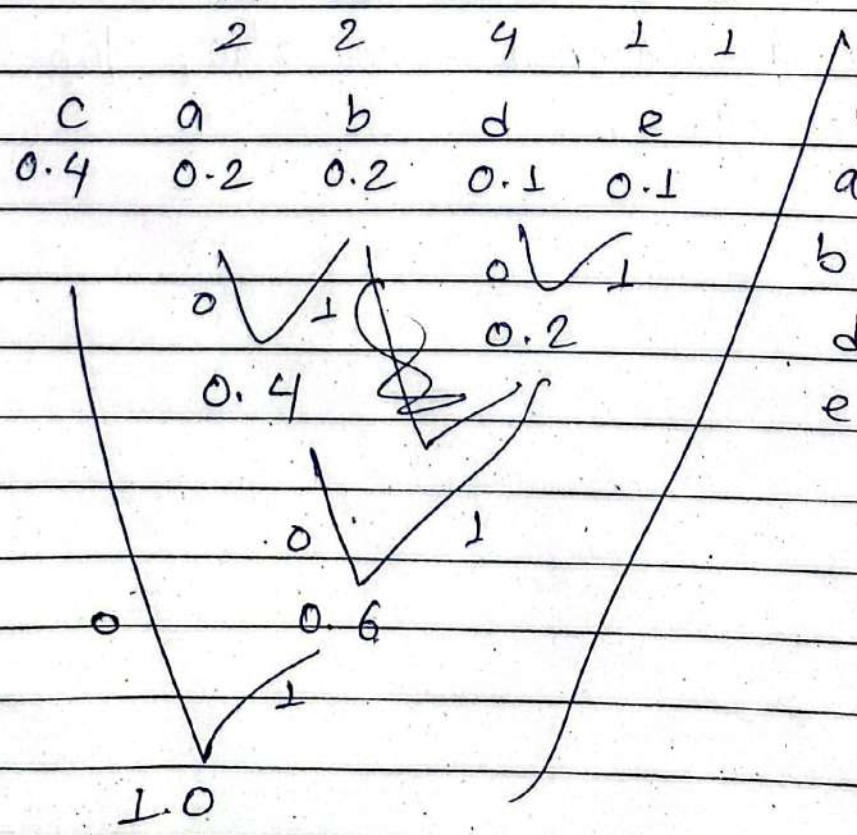
- Popular algo for encoding data.
- greedy approach.
- reduces average access time of codes as much as possible.
- unique code generator algo.
- generates optimal Prefix code.
- lossless data compression algo.
- uses variable length code word to represent character in message.
- shorter codes are assign to more-frequent character. and longer for less frequent.
-

eg:

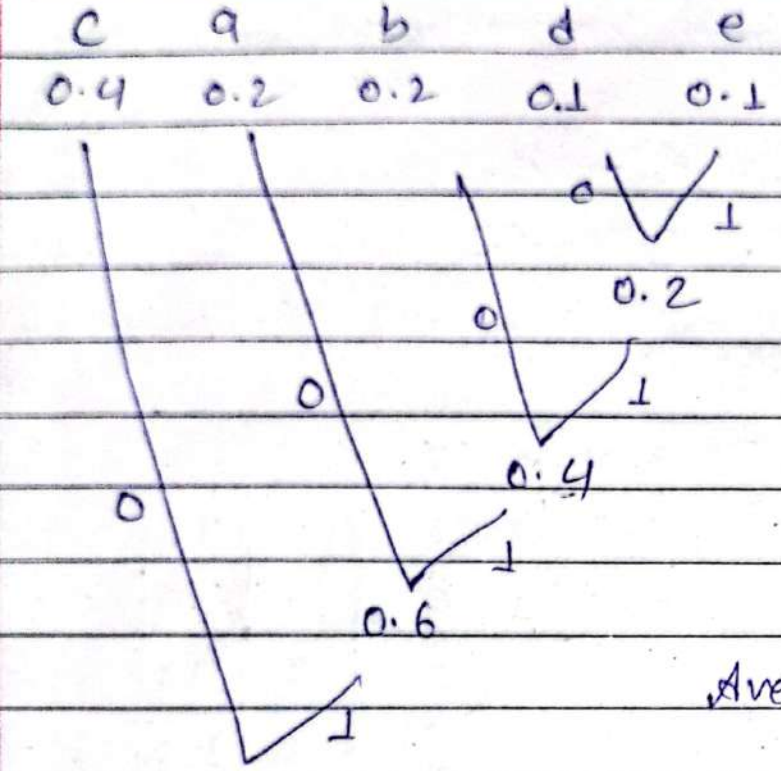
String: ~~aaaa bbb c~~

aa bb cccc d e.

2 2 4 1 1



Huffman code



	Prefix code
c → 0	1
a → 10	2
b → 110	3
d → 1110	4
e → 1111	4

bit length.

Average bit length :

$$\sum_{i=1}^{n} (bl)_i * P_i$$

bl → bit length.

P → probability of occurance

= 1×0.4 + 2×0.2 + 3×0.2 + 4×0.1 + 4×0.1.

= 2.2.

Shannon - fano:

- related to field of data compression
-

a	b	c	d	e
0.2	0.2	0.4	0.1	0.1

c	0.4	0.6
a	0.2	
b	0.2	0.4
d	0.1	
e	0.1	

Stage 2:

						length.
c	0.4	0	0		00	2
a	0.2	0	1		01	2
b	0.2	1	0		10	2
d	0.1	1	1	0	110	3
e	0.1	1	1	1	111.	3.

Average bit length = $\sum_{i=1}^{n} (bl_i) * P_i$.

$= 2 \times 0.4 + 2 \times 0.2 + 2 \times 0.2 + 3 \times 0.1 + 3 \times 0.1.$

$= 2.2.$

Which coding technique is optimal & worst case.

Job Scheduling : Job Sequencing with deadline

- it has unique processor machine. (only one machine)
- each job J_i require one unit time to complete
- each job J_i has deadline d_i and profit P_i
- Profit P_i is added in (Paid obs) overall profit iff J_i meets deadline
- arrange set of jobs in descending order of profit.
- A machine can provide service to single job at a time

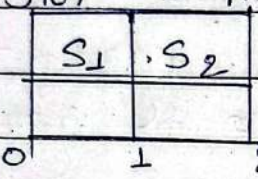
Algorithm:

1. Arrange all job in descending order with respect to P_i
2. Allocate time slot w.r.t maximum deadline
3. K_i = min { max {deadline}, d [J_i] }
   (slot_position)
4. Allocate time slot for J_i on the basis of K_i.
5. If the slot is already occupied search for position before it.
6. If all the ~~slot~~ slots are pre-occupied drop off the Job J_i.
7. n = maximum deadline & m = total jobs
   total time = $n \times m$.

Example-1.

Job J_i	J_1	J_2	J_3	J_4
Profit P_i	100	10	15	17
deadline d_i	2	1	2	1

Time_Slot = max deadline = 2.

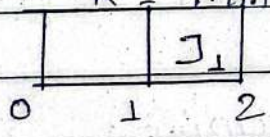


* Arrange the job in descending order:

J_i	J_1	J_4	J_3	J_2
P_i	100	17	15	10
d_i	2	1	2	1

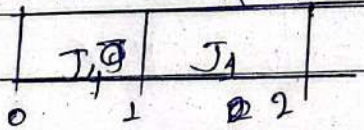
* Iteration: 1 ($i=1$) for J_1.

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



* $i=2$ (J_4)

$k = \min(\text{max\_deadline}, d_i) = \min(2, 1) = 1.$



Set = $\{J_1, J_4\}$

Sequence = $\{J_4, J_1\}$ ⇒ max profit = 17 + 100

= 117. #

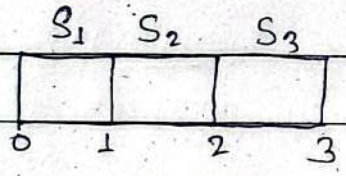
Example-2

J_i	J_1	J_2	J_3	J_4	J_5
P_i	30	20	40	15	25
d_i	3	3	2	1	1

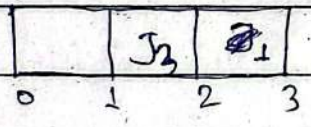
No of timeslot = max_deadline = 3

Jobs in Ascending order =

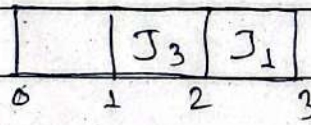
J_3	J_1	J_5	J_2	J_4
40	30	25	20	15.
2	3	1	3	1.



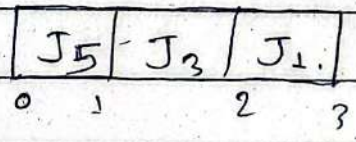
Iteration-1: Slot position k_i = min (max_deadline, d_i)
= min (3, 2) = 2



Iteration 2: Slot position k_i = min (max_deadline, d_i)
= min (3, 3) = 3



Iteration-3: Slot position k_i = min (max_deadline, d_i)
= min (3, 1) = 1.



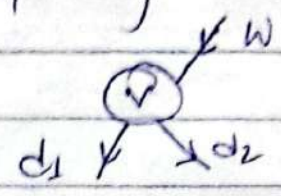
Set = { J_1, J_3, J_5 }.

Sequence = { J_5, J_3, J_1 }.

max. profit = 40 + 30 + 25 = 95 //

Tree Vertex Splitting:

- used in directed graph.
- vertex Splitting is carried out w.r.t δ (threshold)

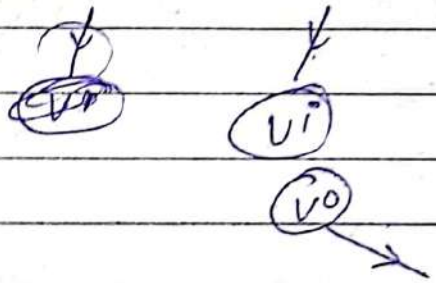


$w \rightarrow$ weight.

$d_1, d_2 \ldots d_n \rightarrow$ delay.

$$d[v] = \max \begin{cases} d_1 + w \\ d_2 + w_2 \\ \vdots \\ d_n + w \end{cases}$$

if $d[v] > \delta \rightarrow$ split the vertex



i → inbound.

o → outbound.

- TVS can be defined as

let $T = (V, E, W)$ be a weighted directed tree

V → set of vertices.

E → set of edges.

W(i,j) → weight of edge

Algorithm:

```
Algorithm TVS(T, δ){
    if(T ≠ 0) then {
        d[T] = 0;
        for each child v to T do {
        TVS(v, δ);
        d[T] = max{d[T], d[v] + w[T,v]};
    if((T is not root) and (d[T] + W(parent(t),T) → δ
then { write[T]; d[T] = 0; }}}
```

Date
Page

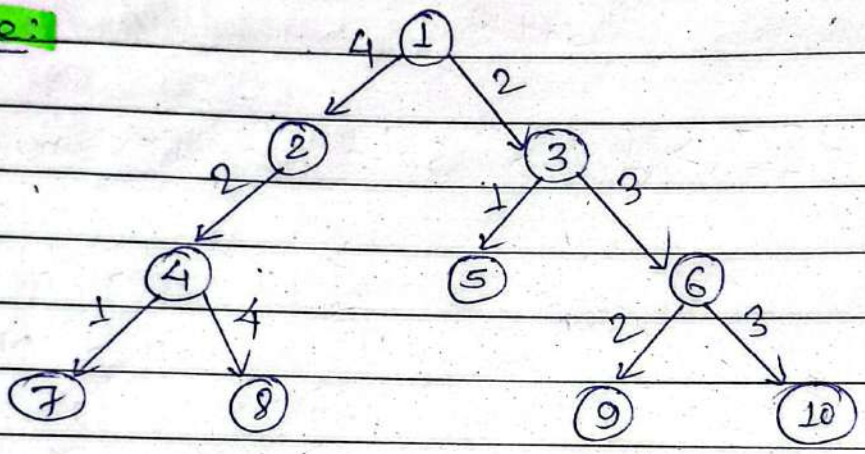
Analysis:

Visit each vertex V;

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

$\therefore O(n) = n$ #.

Example:



when $d[v] > \delta$ place the booster i.e. split the vertex.

Soln:

All leaf nodes delay = 0 and leaf nodes are 7, 8, 9, 10, 5

$\delta = 5$.

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

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

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

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

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

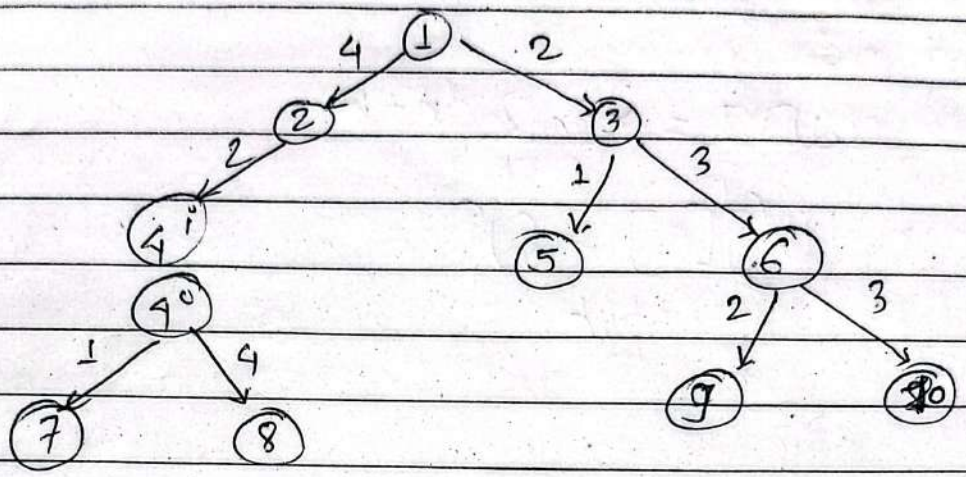
$\therefore d[7], d[8], d[5], d[9], d[10] < \delta$.

no booster required.

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

$= \max\{1+2, 4+2\} = \max\{3, 6\}$

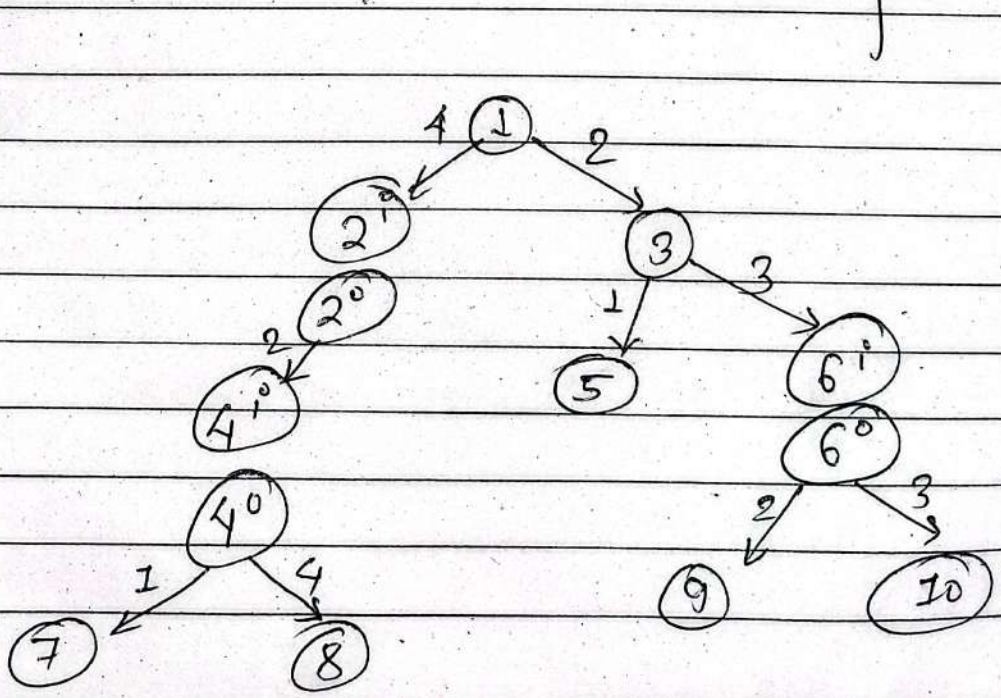
$= 6 > \delta$. ~~no~~ Splitting



$d[5] = \max\{2+4 = \max\{6 = 6 > \delta$. booster.

$d[6] = \max\{3+2, 3+3 = \max\{5, 6 = 6 > \delta$ booster

$d[3] = \max\{\cancel{2+3}\ 3+2,\ 1+2 = \max\{5, 3 = 5 > \delta$ false (no boost



Dynamic Programming:

- technique in computer programming used to solve optimized problem by breaking down a large problem into smaller problems - and solving each sub-problem.

- optimal soln can be found by combining the optimal soln of each subproblem. - principle of optimal substructure

- efficient for cases with lots of overlapping sub-problem
- Solves problem by recombining soln of sub-problems.
- Sub problem may share sub-subproblems.

eg: fibonacci No. 1,1,2,3,5,8,13,21,....

soln:
```
def fib(n)
    if n==0 or n==1
        return n.
    else:
        return fib(n-1) + fib(n-2)
```

Step 1: Construct the structure
Step 2: define equation to calculate values within structure
Step 3: Calculate values using bottom-up approach.
Step 4: Re-construct the structure by using the calculated value to get optimal structure

Optimal BST: (OBST)

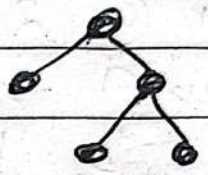
→ a BST- whose average search time is least.

→ for n. number of nodes

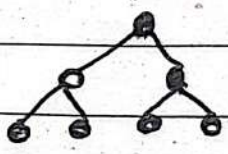
No. of BST = $\frac{^{2n}C_n}{n+1}$.

→ 4 types

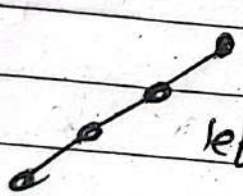
* Incomplete BST.
* Complete BST.
* left skewed.
* Right skewed.



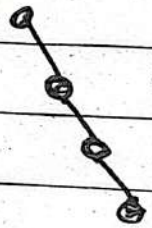
→ InComple...



complete.



left-skewed.



right skewed.

→ There are 2 probability associated with BST.

* Successful search = P_i
* unsuccessful search = q_i

i	0	1	2	3	4	5
P_i		0.15	0.1	0.05	0.10	0.20
q_i	0.05	0.1	0.05	0.05	0.05	0.10

we need to find weight table, cost table and root table.

	k_0	k_1	k_2	k_3	k_4	k_5
P_i	–	0.15	0.1	0.05	0.1	0.2
q_i	0.05	0.1	0.05	0.05	0.05	0.1
	x	0.25 ①	0.15 ②	0.10 ③	0.15 ④	0.3 ⑤

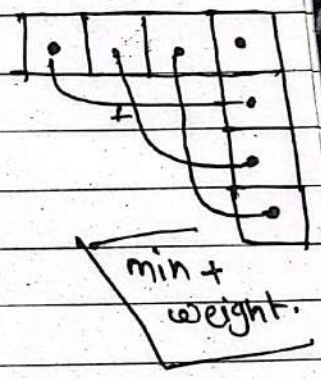
i ↓ j →

i \ j	0	1	2	3	4	5
1	0.05	0.3	0.45	0.55	0.7	1.0
2	–	0.1	0.25	0.35	0.50	0.8
3	–	–	0.05	0.15	0.30	0.6
4	–	–	–	0.05	0.2	0.5
5	–	–	–	–	0.05	0.35
6	–	–	–	–	–	0.1

weight table.

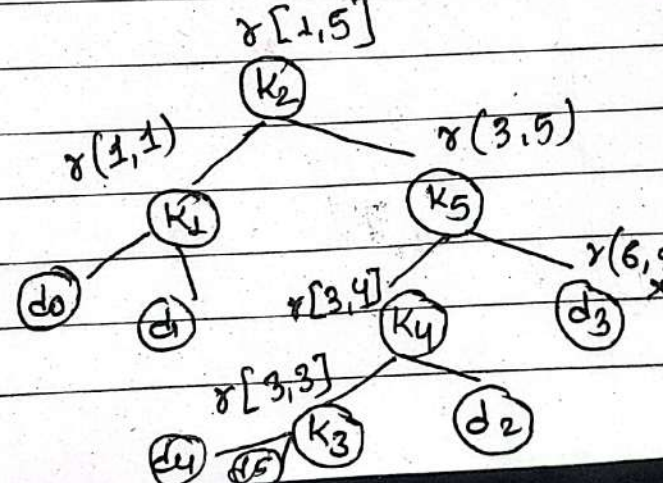
i ↓ j →

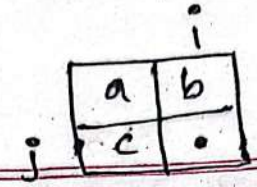
i \ j	0	1	2	3	4	5.
1	0.05	0.45	0.9	1.25	1.75	2.75
2	–	0.10	0.4	0.7	1.2	2.0
3	–	–	0.05	0.25	0.6	1.3
4	–	–	–	0.05	0.3	0.9
5	–	–	–	–	0.05	0.5
6	–	–	–	–	–	0.1



cost of tree = 2.75

root \	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





Data Page

String Editing:

if i=j ;
a
else
min {a, b, c} + 1.

a b c d e f.

a k e d g.

		a	b	c	d	e	f
	g	1	2	3	4	5	6
a	k	1 → 0	1	2	3	4	5
k	c	2	1	1 → 2	3	4	5
c	d	3	2	2 → 1	2	3	4
d	g	4	3	3 → 2	1	2	3
g		5	4	4	3	2	2	3

Date
Page

Knapsack Problem With Backtracking.

- It is a classic optimization problem that asks for the maximum value that can be achieved within a given weight capacity in a knapsack.
- The objective is to maximize the total value of the selected items while not exceeding the capacity of the knapsack.

Algorithm:

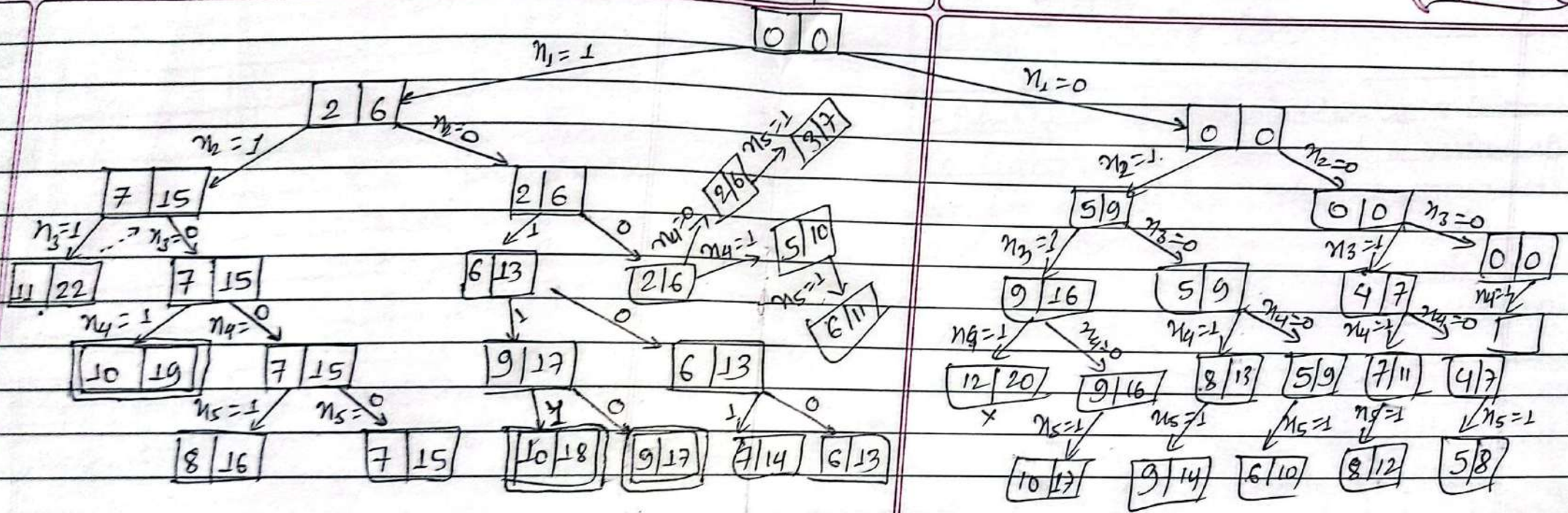
1. Start with an empty knapsack and the total capacity it can hold.
2. Consider each item and make decision: either to include or not.
3. Recurse for the next time with the updated weight and value if the item is included.
4. Backtrack: to explore the option of not including the item after exploring including case.
5. Compare the result of including and not including the current item to select the best solution for the current state.
6. Repeate the process for all item.
7. Return the maximum value obtained.

Example:-

Item	Weight	(Profit) value.
1	2	6
2	5	9
3	4	7
4	3	4
5	1	1.

Time complexity of knapsack problem solved using backtracking is $O(2^n)$
where n is number of items.

Analysis of Time Complexity:
The exponential time complexity $(O(2^n))$ arise from the fact that for each item, there are two choices either to include it or not leading to a decision tree of depth n with 2^n possible combination.



$x_1, x_2, x_4 \Rightarrow$ weight $= 2+5+3 = 10$
$\Rightarrow$ profit $= 6+9+4 = 19$

Identifying Repeated Elements.

- Identifying repeated elements in a collection is a common task ~~in com~~ used in various application like data analysis, database management, and algorithm design.
- The goal is to find all elements that appear more than once in the collection.

Algorithm:

1. Initialize an empty has table
2. Iterate through each element
3. for each element, check if it is already in the hash table:
   - If not, add it to the hash table with count 1.
   - If it is, increment its count in the has table.
4. After the iteration, iterate through the hash table and collect all elements with count greater than 1.
5. Return the list of ~~repeated~~ elements.

Assume Data:

Array of integers:

[4, 2, 5, 8, 4, 5, 8, 9];

Trace the output:

- Create hash table
- Insert 4 with count 1.
- Insert 2 with count 1.
- Insert 5 with count 1.
- Insert 8 with count 1.
- Increase count of 4 by 1.
- Increase count of 5 by 1.
- Increase count of 8 by 1.
- ~~Incre~~ Insert 9 with count 1.

	count.
4	1+1
2	1
5	1+1
8	1+1
9	1.

element with count > 1 is [4, 5, 8]

Time Complexity: O(n) → n is the number of elements in the collection.

Analysis of Time Complexity:

Every element is accessed and processed ~~at least once~~ exactly once.

#

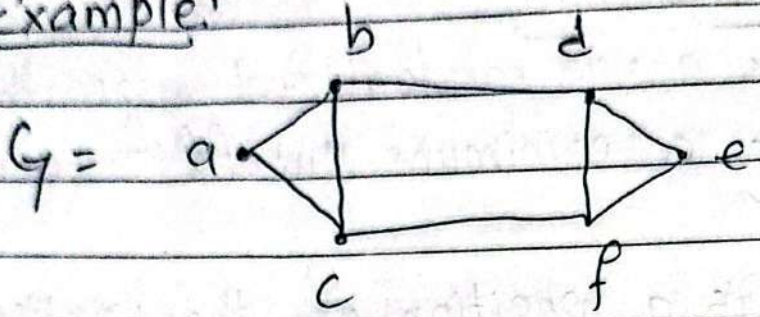
Karger's Algorithm:

- It is a randomized algorithm to compute a minimum cut of a connected graph.
- Cut is a position of the vertices of a graph into two disjoint subsets that are connected by at least one edge.
- The goal is to find the cut that minimize the no. of crossing edges.

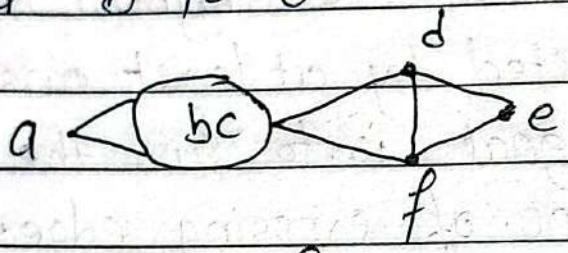
Algorithm:

1. While there are more than 2 vertices in the graph:
   - pick remaining edge (u,v) at random.
   - Merge the two vertices into single vertex, combining the ed. adjacent edges.
2. After reducing the graph to two vertices the cut represented by the edges between these two vertices is one possible minimum cut of the original graph.

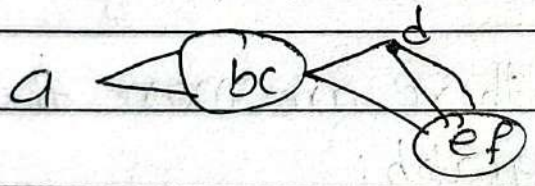
Example:



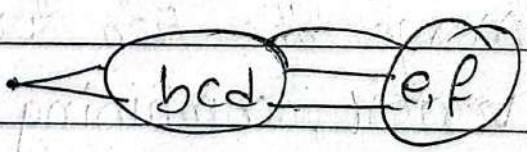
1. Cut b to c.



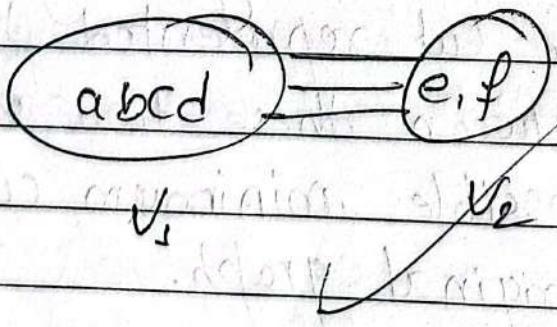
2. Cut d to f.



3. Cut bc to d.



4. Cut a to bcd



Date
Page

Time Complexity:

$O(n^2m)$ where n = no. of vertices.
m = no. of edges.

- In each iteration of the contraction step an edge is selected randomly, and the merging of vertices and edges is performed which takes
$O(n+m)$
and the process is repeated $O(n)$ times.

Overall

Unit - 2.

Date
Page

* Explain Reduction with example
* Difference betⁿ P, NP, NP-hard and NP-Complete with example.
* How can you prove that a problem is complete.
* Prove Satisfiability of Boolean problem (SAT)
* Cook's theorem.
* Maximum CLIQUE Problem.

Complexity Theory:

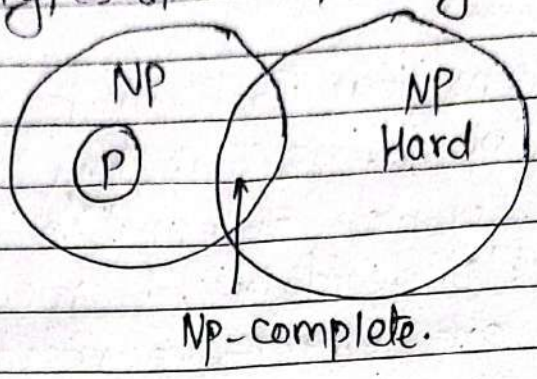
→ A mathematical Problem is computable if it can be solved in principle by a computing device.
→ Some common alternative phrases for computable are solvable, decidable, & recursive.
→ There are extensive research and classification of which problem is computable and which is not.
→ It concerned with the resources such as time & space, needed to solve computational problem.
→ It is the appropriate setting for study of such problem

Complexity Classes:

→ It helps computer scientist groups problem based on how much time and space required to solve the problem and verify solution.
→ A big-O notation is necessary to understand the complexity classes.
→ A complexity class is the set of all the computational problem which can be solved using a certain amount of a certain computational resources.
→ Time complexity requires how much no. of steps to completion of a problem and space complexity requires how much memory are useful in organizing similar type of problem.

→ There are following types of complexity classes.

- P- class.
- NP class.
- Np- hard.
- NP- complete.



* P- class:

(yes/no question)

→ P class contains decision problem that are solved by deterministic turing machine in polynomial time.
→ An algorithm takes polynomial time to solve this type of problem.
→ The Big-O notation for P- class problem is $O(n^k)$.
→ This class contains many natural problems:

- Calculating the greatest common divisor.
- finding a maximum matchings.
- Decision version of linear programming.
- Problem are easy and quick to solve in polynomial time.

* Np- class:

→ Problem that are verifyable in polynomial time
→ it is easy to check the correctness of a claimed answer with the aid off of a little extra information.
→ we are not finding a way to solution instead looking for way to verify the correctness of solution.

→ Every problem of Np-class can be solved in exponential time using exhaustive search.
→ Np - stands for Non-deterministic polynomial time.
→ Collection of decision problem that can be solved by a non-deterministic machine in polynomial time.
→ Solution to Np-problem are hard to find but easy to verify.
→ Np-problem can be verified by turing machine in polynomial time.

eg:
Boolean Satisfiability problem (SAT)
Hamiltonian Path problem
Graph colorings.

* Np-hard:

→ It is a class of problems such that every problem in Np reduce to Np-hard.
→ If a problem is hard as the hardest problem in Np-class then we will say that this problem is Np-hard.
→ If solution are given for Np-hard problem then it takes a long time to check whether it is right or not.
→ Np-hard problems are slow to verify, slow to solve and can be reduced to any other Np-problem.

eg:
- Halting Problem.
- Qualified Boolean formula.
- No Hamiltonian Cycle.

NP-Complete:

→ A problem is Np-Complete if it is both NP and Np-hard.

→ Np-complete problem are the hard problem in NP.

→ Any Np-problem can be transformed or reduced into Np-complete problem in polynomial time.

eg:
- Decision version of 1/0 knapsack.
- Hamiltonian cycle.
- Satisfiability.
- Verter cover

→ Np-complete problems are are also quick to verify slow to solve and can be reduced to any other NP problem.

→ A problem that is NP and NP-hard is Np-Complete

Decision Problem:

* Reduction:

→ Reduction is a transformation of one problem into another one.
→ Reduction of problem Q_1 to Q_2 is a mapping of every instance of Q_1 as q_1 to an instance of Q_2 as q_2 such that q_1 is yes iff q_2 is yes. (Decision Problem)
→ if Q_1 reduces to Q_2 in polynomial time and Q_2 reduces to Q_3 in polynomial time then Q_1 reduces to Q_3 in polynomial time.

* Polynomial Time Reduction:

→ It is a mapping reduction that can be computed in polynomial time.
→ Suppose that A and B are two language. function f is a polynomial time reduction from A to B if both the following are true
  * f is computable in polynomial time
  * for every x, $x \in A \Leftrightarrow f(x) \in B$
  ∴ $A \leq_p B$ if there exist a polynomial time reduction from A to B.

Cook's Theorem:

It states that the boolean satisfiability problem is NP-complete. that is, any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of deterministic whether a boolean formula is satisfiable.

SAT $\in$ P iff P = NP.

Proof: There are two parts to prove that boolean satisfiability Problem (SAT) is NP-complete.

1. One is to show that SAT is NP-~~complete~~ problem.
→ SAT is NP because any assignment of boolean value to boolean variable that is claimed to satisfy the given expression can be verified in polynomial time by a deterministic turing machine

2. Another is to show that every NP-~~complete~~ problem can be reduced to an instance of a SAT problem.
→ suppose that a given problem in NP can be solved by the NDTM $M = \{Q, \Sigma, S, f, \delta\}$ where

$Q \rightarrow$ set of state.
$\Sigma \rightarrow$ Alphabet.
$S \in Q \rightarrow$ set of initial state.
$f \subseteq Q \rightarrow$ set of accepting state.
$\delta \rightarrow$ transition relation.

$\delta \subseteq ((Q|f) * \Sigma) \times (Q \times \Sigma \times \{-1, +1\})$

M accepts or rejects an instance of the problem in time $p(n)$ p: polynomial with size of instance 'n'.
⇒ for each input, I, we specify a Boolean expression which is satisfiable iff the machine M accept I.

* SAT.

→ also called boolean satisfiability problem.
→ A propositional logic formula φ is called satisfiable if there is some assignment to its variable that makes it ~~executable~~ evaluate to TRUE.
→ $p \wedge q$ is satisfiable i.e. for $p \Rightarrow$ & $q \Rightarrow \perp$ $p \wedge q$ is true
→ $p \wedge \neg p$ is not satisfiable cause no value of p could this function true
→ SAT is NP-Complete.

Prove that SAT ∈ NP class.

→ given an assignment for $x_1, x_2, \ldots x_n$. you can check if $\phi(x_1, x_2, \ldots x_n) = 1$ in polynomial time by evaluating a formula on a given assignment.
Hence
SAT ∈ NP.

Prove that SAT ∈ NP-hard.

Vertex Cover:

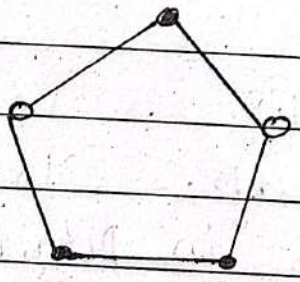
→ In mathematical discipline or graph theory "A vertex cover of graph is a subset of vertices which cover every edge." one of

→ An edge is covered iff its endpoint is choosen.

→ A vertex cover of a Graph G is a set of vertices incident to every edge of G.

Vertex Cover Problem: What is the max. size Vertex cover in G?

Given graph $G = (V, E)$, find smallest $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ or both.

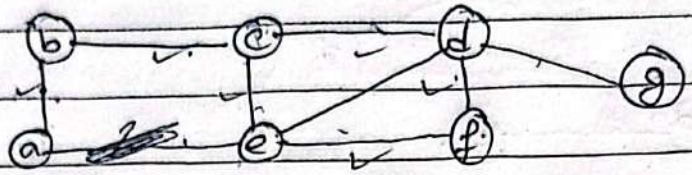


This problem can be solved by

① Greedy approach.

② Approx optimal soln.

Example:



Soln:

$C = \phi$

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

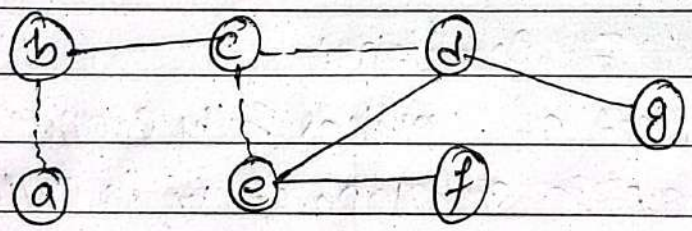
Initially let select (b,c) edge, where

$u = b$

$v = c$

$C = \phi \cup \{b,c\} = \{b,c\}$.

Now, from E' remove every edge incident $E' = \{(d,e), (e,f), (d,f), (d,g)\}$



Again, lets select (e,f) vertex where

$u = e$

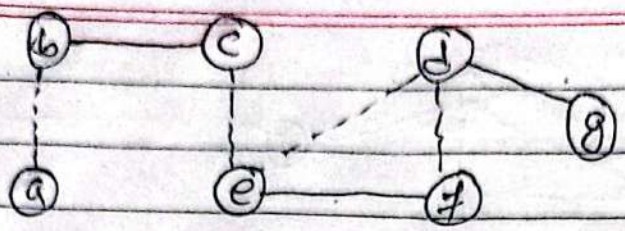
$v = f$

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

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

Now, remove E' every edge incident

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



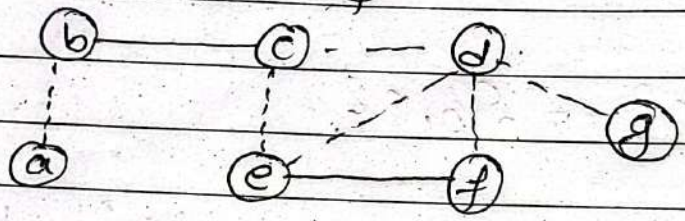
Again let select (d,g) vertex where,

$u = d$

$v = g$.

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

$E' = \phi$

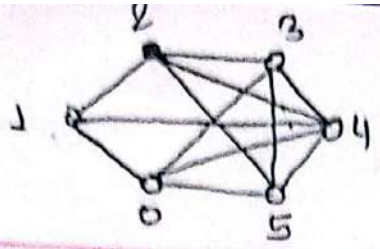


Maximum CLIQUE Problem:

→ In a graph G a subset of vertices fully connected to each other i.e. complete subgraph of G is called cliq

→ CLIQUE problem: How large is the max. size clique a graph?

→ given a graph of vertices some of which have edges between them, the max clique is the largest su of vertices in which each point is directly connected to every other vertex in the subse



→ Given graph with vertices 0, 1, 2, 3, 4, 5. Here, the max. clique problem is the largest subset of graph which is complete.

→ And the max. clique size is 4, and the max. clique contains the nodes : 2, 3, 4, 5.

Travelling Saleman Problem: (TSP):

→ It consist of a salesman and a set of cities. The salesman has to visit each cities starting from a certain one & return to the same city. The challenge is that the travelling salesman want to minimize the length of the trip.

→ It find shortest possible route.

→ Given a weighted graph G, find the min. cost path from V_1 to V_n which visits each vertex exactly once.

→ No known polynomial time algo for solving this problem.

Unit - 3 : Online and PRAM algorithm.

- How to compute rank in linear Array. - 74. imp quest.
- Explain work done and its efficiency with suitable example for PRAM. When do you confirm that algorithm is optimal.
- Work optimal PRAM algorithm to solve prefix computation. (of D. CS. N)
- where do you think online algo will be implemented. Explain with eg. - AC imp ques. - 34
- Compare and contrast strategies implemented in Online algo. Exp.
- Explain speed up, asymptotic speed up, total work done and efficiency of PRAM algo.
- State and give algo to solve Ski-rental problem -
- Adv. & Disadv. of online algo.
- Online vs offline algo.
- Explain PRAM algo to Solve two sorted list.
- explain online algo for page replacement.

AC-notes.pdf → 36, 37, 39.
Online Algo.
Ski-rental — a & c.
Prefix-computation.

algo & comp.
- Ski-rental - 52.

Work optimal PRAM algo. to solve Prefix Computation.

Algorithm:

Step 1: Processor i ($i = 1 \ldots n/\log n$) in parallel computes the prefix of assigned elements.
It takes $O(\log n)$ time.

Step 2: A total of $\frac{n}{\log n}$ processor collectively perform:
- recursively compute the prefix of $n/\log n$ elements.

Step 3: Each processor update the prefix it computed previously ~~as~~.

~~Step 4~~:
optimality of the algorithm:

→ The prefix computation is solved by linear algo in $O(n)$ time.
→ parallel algorithm uses $n/\log n$ processor and workdone is $O(\log n)$ then.

$$\text{efficiency} = \frac{S(n)}{p \cdot T(n,p)} = \frac{O(n)}{n/\log n * \log n} = \frac{O(n)}{O(n)} = 1$$

Hence above algorithm is work optimal.

Example:

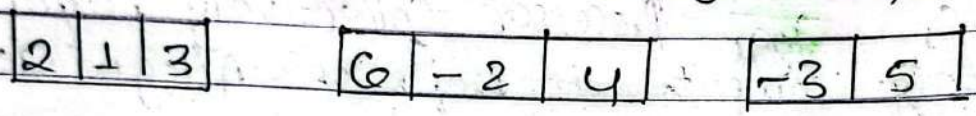
2, 1, 3, 6, -2, 4, -3, 5.

Since, we have 8 elements. (n) = 8.

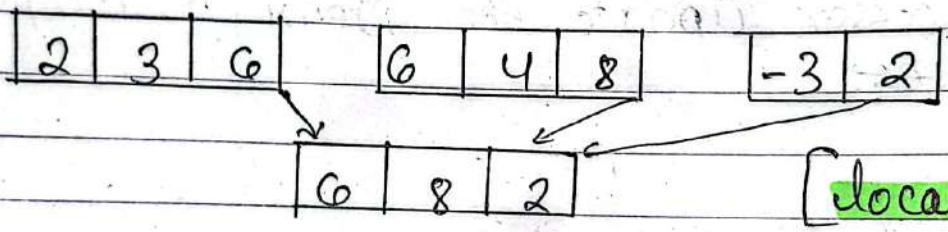
no. of processor = n/logn = 8/3

= 3 processor.

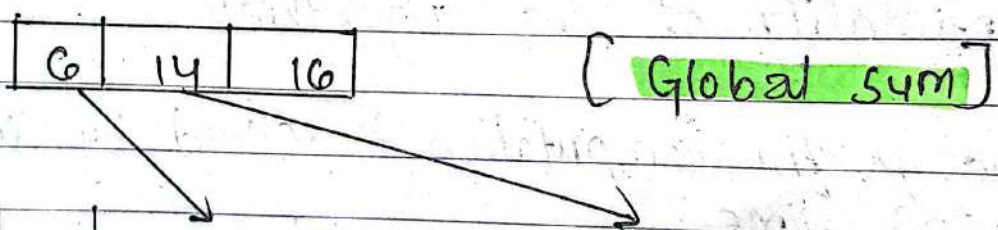
each processor has n/logn input.



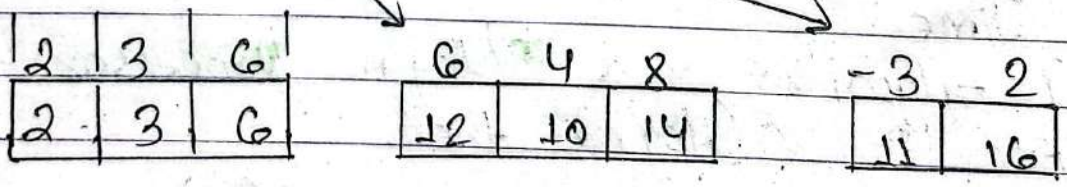
Step 1:



Step 2:



Step 3:



Hence the output is

2, 3, 6, 12, 10, 14, 11, 16.

Explain Work done and its efficiency with a suitable example for PRAM.
When do you confirm that the algorithm is Work optimal

Solution:→

for any PRAM algorithm:

a) Work done:

If p- processor parallel algorithm for a problem runs in $T(n,p)$, the total work done by this algorithm is $p * T(n,p)$.

b) efficiency:

If $S(n)$ be the time taken by sequential algo to solve the problem then the efficiency is computed as:

$$\frac{S(n)}{p * T(n,p)}$$

p = no. of processor.
$T(n,p)$ = run-time of algorithm

* Any algorithm is said to work optimal if the algo gains linear speedup or if efficiency $= 1$.

To show all these lets take prefix Computation in parallel environment:

Step1: Processor i ($i = 1 - n/\log n$) compute the prefix of ($\log n$) sized element. in $O(\log n)$ time

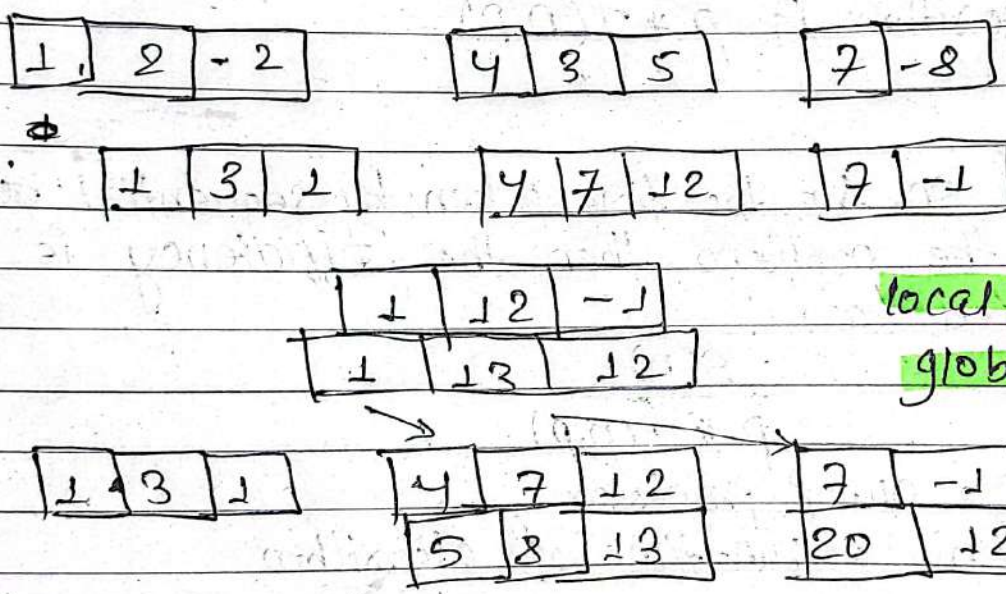
Step 2: $(n/\log n)$ processor computes prefixes of $(n/\log n)$ elements in $O(\log n)$ time.

Step 3: Each processor updates the prefix computed in step 1 with the prefix computed in step 2.

Example: 1, 2, -2, 4, 3, 5, 7, -8

$n = 8$

$p = n/\log n = \frac{8}{3} = 3$



Since, we have $n/\log n$ processor and it takes $O(\log n)$ time,

Work done is: $p * T(n,p) = n/\log n * \log n = O(n)$

Prefix computation takes $O(n)$ time in sequential algo.

hence,

efficiency $E = \frac{S(n)}{p \times T(n,p)} = \frac{O(n)}{\frac{n}{\log n} \times \log n} = \frac{O(n)}{n} = O(1)$

Since the efficiency is linear i.e. O(1) we can say that the algorithm work optimal.

How to Compute Rank in Linear Array.

→ The list ranking problem is of identifying the distance for each object in a linked list from the end of the list.

Given a list L with n objects, we compute the dista from the end of the list to each n in L.

We get input as a list each of which hold some data and a pointer to its neighbour in the list. Then we have to compute rank for each node.

Rank can be calculated with the help of parallel algorithm using the concept of pointer jumping.

'Process':

1. Initially each node points to immediate neighbour.
2. Then the neighbour of each node is modified to the neighbour of it's neighbour.
3. step 2 is repeated until/unless all the nodes poin to ground/end of the list.

Also we need to calculate distance.

1. initially set the distance of each node as 1 except for the last node which is 0.
2. Then for each step distance of i^{th} item is calculated as

$$d[i] = d[i] + d[next[i]].$$

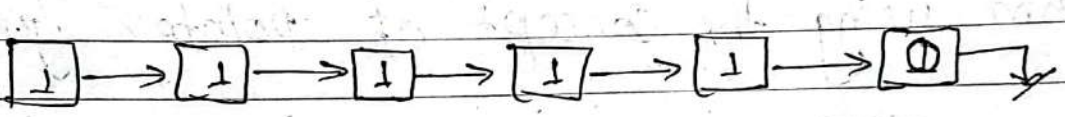
Algorithm:

```
for (int n=1 ; n ≤ ⌈log n⌉ ; n++)
    if neighbour of i ≠ 0
    {
        d_i = d_i + d_next(d_i)
        ~~neighbou~~ next_i = next_i + next(next_i)
    }
```

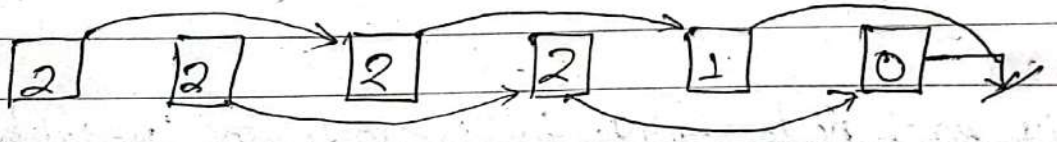
Example: given the linked list

A[2] → A[6] → A[5] → A[4] → A[1] → A[3]

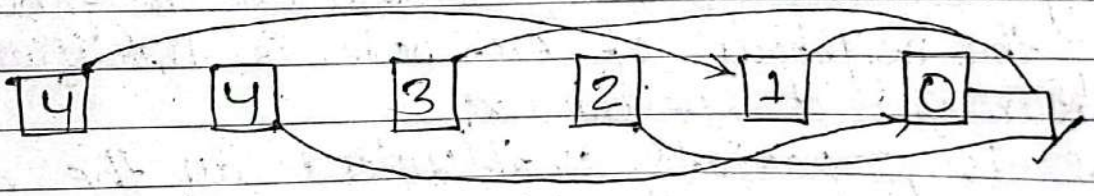
Initially:



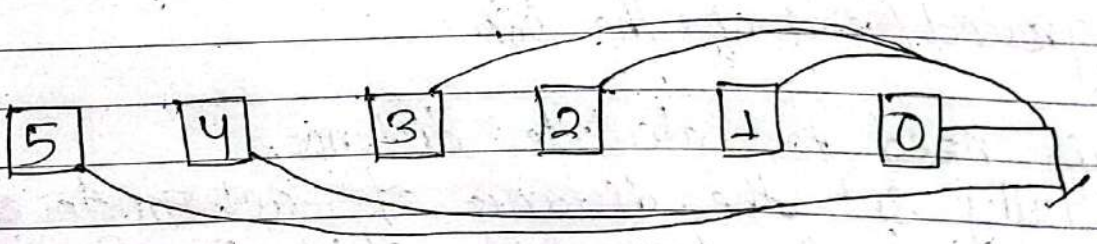
Pass 1:



Pass 2:



Pass 3:



after log n iteration we have

$\log n = 6$

$n = 3$

respective rank and,

node	Rank
a[2]	5
a[6]	4
a[5]	3
a[4]	2
a[1]	1
a[3]	0

Analysis:

→ Total running time : $O(\log n)$

Total work $n * O(\log n) = O(n \log n)$

Sequential algorithm : $O(n)$

Speed up $= O(n) / O(\log n) = O(n / \log n)$

Efficiency $= O(n) / O(n \log n) = 1 / \log n$.

It is not work optimal.

Online Algorithm:

→ An online Algorithm is one that can process its input piece-by-piece in a serial fashion i.e. in the order that the input is fed to the algorithm, without having the entire input available from the start.

→ An offline algorithm is given the whole problem data from the beginning and is required to output an answer which solve the problem at hand.

eg→ Selection sort requires that the entire list be give before it can start while insertion sort doesn't.

→ As it 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 algorithm has focused on the quality of decision making that is possible in this setting.

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

→ An online algorithm is c-competitive if there is a constant b for all sequence of operation.

$A(s) \leq c \cdot opt(s) + b.$

$A(s)$ → cost of A on the sequence s.

$opt(s)$ → optimal offline cost for same sequence

Comparative ratio is a worst case bound.

Characteristic of online algorithm:

* Sequential Processing.
* Real-Time decision making.
* Adaptability. → adapt to new arriving data.
* Application → RT. system, network routing, Stock trading.

Characteristic of offline algorithm.

* Complete data access.
* Global optimization
* Non-sequential Processing.
* Application → Batch processing, Sorting.

Advantages of Online Algorithm:

* Real Time Decision making
* Adaptability:
* low initial data requirement:
* Incremental Processing:
* Flexibility.

Disadvantages of Online Algorithm:

* Suboptimal Decision:
* Competitive ratio
* Complexity in handling uncertainty.
* Possibility of Greedy decision.
* Error propagation.

Attributes	Online Algorithm.	Offline Algorithm.
Input access	Sequential, Piece-by-piece.	Entire input available at once
Decision making.	Real-Time based on current and past input.	Based on entire dataset & globally optimal.
flexibility.	Must adapt to new arriving data.	Can optimize with complete data.
Performance evaluation.	competitive ratio against optimal algorithm.	Absolute performance.
Processing.	Incremental as data arrives	Batch processing, all at once
Typical Application	Real Time system, Online Caching Network routing, stock trading.	Batch Processing, Sorting, optimization problem, etc.
Example	LRU Paging algorithm.	Belady's optimal paging algo.

Where do you think online algorithm can be implemented

→ Online algorithm can be implemented for problems where difficulty is not necessarily 'computationally hard' but rather the algorithm doesnot have all the information it needs to solve the problem.

→ Online algorithm need to make decision without full knowledge of the input. They have full knowledge of the past but no knowledge of the future.

→ Online algorithm are designed for settings where inputs/ data is arriving over time and we need to make decisions on the fly without knowing what will happen in the future.

→ Page replacement algorithm is an example of online algorithms. Since one cannot predict the pages the system is going to demand in advance, the algorithm has no knowledge about the future.

Problem Statement

→ There are two memory M_1 & M_2. → fast memory consist k-pages. (cache) → slow memory consist n pages ($k < n$)
→ pages are accessible through M_1 only.
→ page accesing cost for M_1 is 0.
→ if page not found should bring from M_2 to M_1 spending 1. cost.
→ This is called page fault.

Problem statement :

→ When M_1 is full and ~~we need~~ page fault occur, we must evict some pages from M_1.

→ how to choose which page to evict.

Paging : Optimal offline Algorithm

Longest forward Distance (LFD) :

When page fault occur replace the page in M_1 that will be requested farthest out in the future.

Assume $M_2 = \{a, b, c, d, e\}$.

$n = 5$.

$k = 3$.

T =	a	b	c	d	a	b	e	d	e	b	c	c	a	d
	a	a	a	a	a	a	e	e	e	e	c	c	c	c
		b	b	b	b	b	b	b	b	b	b	b	a	a
			c	d	d	d	d	d	d	d	d	d	d	d
				*		*				*		*		

4 cache Missed.

LFD is optimal but it is not online.

Online Paging Algorithms:

* **FIFO** : first In first Out. → evict the page that was entered first in the cache.

$M_2 = \{a, b, c, d, e\}$ $n = 5$ $k = 3$.

$\sigma =$	a	b	c	d	a	b	e	d	e	b	c	c	a	d
	a	a	a	d	d	d	e	e	e	e	e	e	a	a
	b	b	b	a	a	a	d	d	d	d	d	d	d	d
	c	c	c	b	b	b	b	b	c	c	c	c	c	c
		*	*	*	*	*			*		*			

7 cache Miss.

* **LIFO** : last in first out; evict the page that was entered last to the cache.

$M_2 = \{a, b, c, d, e\}$ $n = 5$ $k = 3$

$\sigma =$	a	b	c	d	a	b	e	d	e	b	c	c	a	d
	a	a	a	a	a	a	a	a	a	a	a	a	a	a
	b	b	b	b	b	b	b	b	b	b	b	b	b	b
	c	d	d	d	e	d	e	e	c	c	c	d	d	d
		*			*	*	*		*			*		

6 cache Misses.

* LRU: least Recently used : evict the page with earliest last reference

$M_2 = \{a, b, c, d, e\}$. $n = 5$, $K = 3$.

T =	a	b	c	d	a	b	d	e	d	e	b	c
	a	d	d	d	d	d	d	d	d	d	d	c
	b	b	a	a	a	e	e	e	e	e	e	e
	c	c	c	b	b	b	b	b	b	b	b	b
		*	*	*		*						*

5 cache miss.

Ski-rental Problem:

→ Assume that you are taking ski-lessons.
→ It is also called Buy-Rent problem.
→ The training period is unknown.
→ Decision has to be made whether to continue Rent or Buy at a particular instance.
→ After each lesson you decide whether to continue ski or to stop totally.

→ you have the choic of either renting for 1$ a time or buy for y$.

→ If you knew in advance how many times you would ski in your life time the choice become simple. If you'll ski more than y-times then buy before you start otherwise rent always.

→ In parctice you don't know how many times you'll ski.

→ An online ~~star~~ strategy is will be number k-such that after renting k-1 times you will buy skis.

→ Setting k=y guarantees that you'll never pay more than twice the cost of the offline strategy

(P.T.O)

Theorem:

→ Setting $k = y$ guarantees that you never pay more than twice the cost of the offline strategy.

Proof:

When you buy skis in your k^{th} visit, though if you quit right after;

$t \geq y$.

∴ total payment is $k - 1 + y = 2y - 1$

offline cost is $\min(t, y) = y$.

The ratio is $2y - 1/y = 2 - (1/y)$

We say that this approach is $2 - (1/y)$ competitive.

Perform list ranking on the following using the concept of list → pointer jumping.

4	5	0	2	3	1	
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	
4	5	0	2	3	1	neighbour
1	1	0	1	1	1	rank.

It-1.

2	3	0	5	2	4	neighbour.
2	2	0	2	1	2	rank

It-2

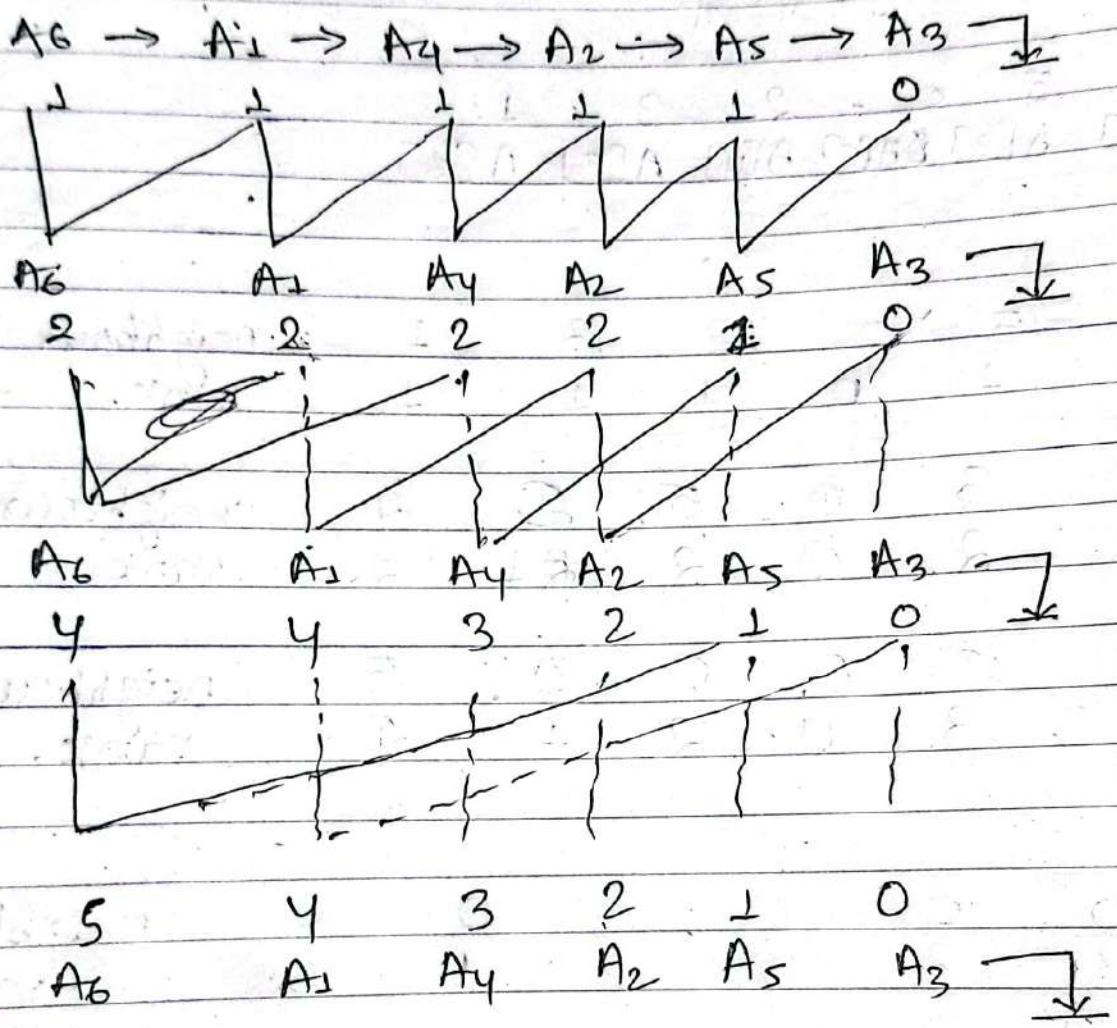
0	0	0	0	0	5	neighbour
4	3	0	2	1	4	rank.

It-3

0	0	0	0	0	0	neighbour
4	3	0	2	1	5	rank.

rough.

It-1.



(i) initialize rank = 1 if dnext.
0 if ! dnext.

(ii) update.
rank = rank i° + rank(d_i)
dnext.d = dnext.d dnext.j

(iii) repeate step 2
until neigh 0.

Unit-3

Online algo:

→ one that can process input piece-by-piece. in serial fashion.
→ they have no idea about complete data set.
→ e.g. insertion sort.
→ forced to make decision that may become non-optimal in future.
→ study is focused on quality of decision making based on this strategy.
→ competitive analysis is performed to check performance
→ The analysis can be performed with comparison to offline algorithm. for same problem instance.
→ online algo is c-competitive if there's a constant b- for all sequence of operation.
→ The comparative ratio is worst case bound.

Characteristic of online Algo:

→ Sequential processing
→ Real Time decision making
→ Adaptability.
→

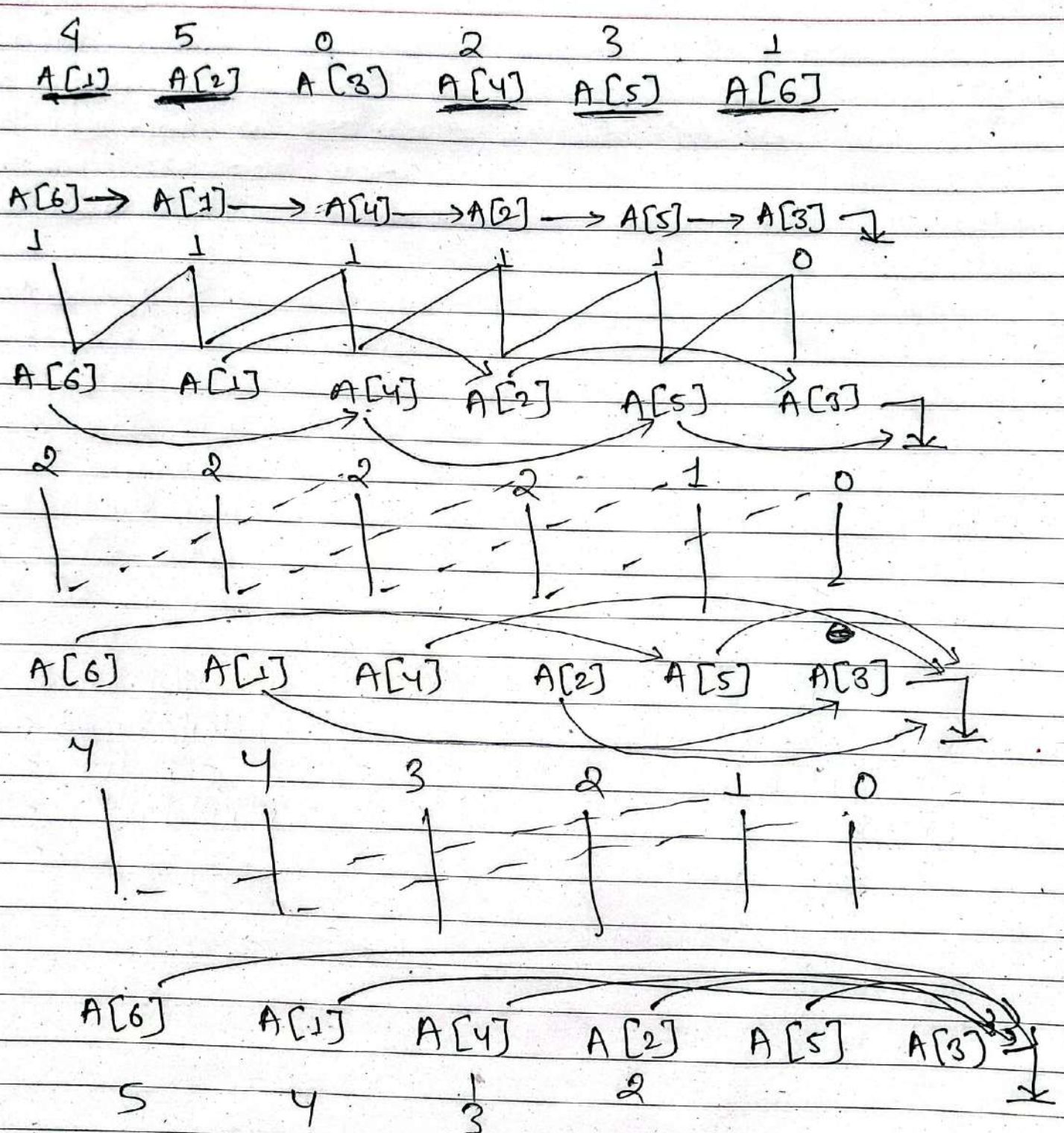
offline Algo
→ complete data access.
→ Global optimization.
→ Non-seq. processing.

4	5	0	2	3	1	neighbour.
1	1	0	1	1	1	rank.
2	3	0	5	0	4	neighbour.
2	2	0	2	1	2	rank.
3	0	0	0	0	5	neighbour.
4	2	0	3	1	4	rank.
0	0	0	0	0	0	neighbour.
4	2	0	3	1	5	rank.

So the final rank is

A[3] = 0
A[5] = 1
A[2] = 2
A[4] = 3
A[1] = 4
A[6] = 5

Sol.


4 5 0 2 3 1
A[1] A[2] A[3] A[4] A[5] A[6]
A[6] → A[1] → A[4] → A[2] → A[5] → A[3]
1 1 1 1 1 0
A[6] A[1] A[4] A[2] A[5] A[3]
2 2 2 2 1 0
A[6] A[1] A[4] A[2] A[5] A[3]
4 4 3 2 1 0
A[6] A[1] A[4] A[2] A[5] A[3]
5 4 3 2

a) Work done:

→ If a P processor parallel algorithm for a problem runs in $T(n,p)$ the total work-done by the algorithm is

$$P * T(n,p).$$

b) Efficiency:

→ If $S(n)$ be the time taken by sequencial algorithm to solve the problem then the eff. is computed as:

$$\frac{S(n)}{P * T(n,p)}$$

⇒ An algo (parallel) is said to be work optimal if the algorithm gains linear speed up

i.e $$\frac{S(n)}{T(n,p)} = \Theta(1).$$

c) Speedup:

let sequential run time = $S(n)$

parallel runtime = ~~$P*t$~~ $T(n,p)$.

Speed up = $$\frac{S(n)}{T(n,p)}.$$

d)

Unit-4

(AC-handwritten Note-1)
Page-76.

Date
Page

* Mesh sorting. Sheer sorting. (Numerical)
* Mesh algo for max. selection with n^2 process.
* Mesh algo for max. selection with n processor. will it work optimal? When it will be optimal.
* Prefix computation on Mesh.
* Broadcasting problem on Mesh. I
* Packet routing on Mesh. Demonstrate broadcasting on Mesh

* Explain embedding of networks.
* Explain process of embedding binary tree on hypercube and calculate Expansion, Dilation and Congestion.
* ~~Odd-even merge~~ Data concentration on hypercube
* Explain hypercube. features of hypercube
* algo to solve prefix computation on hypercube.
* algo to solve broadcasting problem of hypercube.
*

* Process of Creating butterfly network.
* Prefix computation in a butterfly network
* Odd-even merge sort in butterfly net. - (time complexity)
*

Butterfly network →

Mesh Algorithm

→ A mesh is a axb grid where there is a process at each grid point. The edges correspond to communication links and are bidirectional.

→ Each processor in a mesh can be labelled with a tuple (i,j) where $1 \le i \le a$ and $1 \le j \le b$.

→ Every processor of the Mesh is a RAM with same local Memory. Hence each processor can perform any of the basic operation like add, subtract, multiply, compare, etc.

→ The computation is assumed to be synchronous.

→ A closely related model for Mesh is linear array consist of p processor named (1,2,---p)

1. process i is connected to i-1 & i+1 for $2 \le i \le p-1$.
2. Processor 1 is connected to 2 and p is connected to p-1.

→ Processor 1 and p are called boundary processor.

→ Processor i-1 and i+1 are called left and right neighbour of i.

→ We consider square Mesh for a=b as shown in below fig

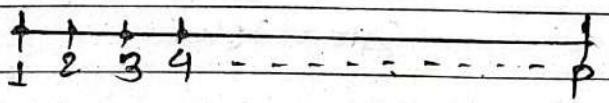


fig: linear array

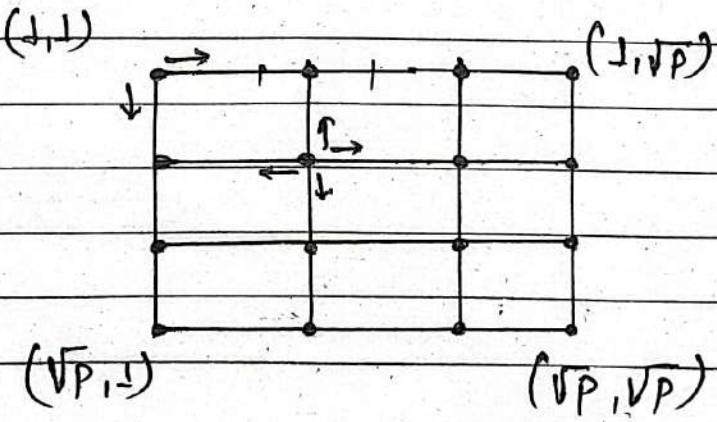


fig: Mesh.

Broadcasting In Mesh:

→ In a case of a $\sqrt{p} \cdot \sqrt{p}$ Mesh broadcasting can be done in two phases:

1. If (i,j) is the processor of Message (M) origin the.
   1. M could be broadcast to all the ~~proje~~ processor in row i.
   2. broadcasting of M is done in each column.

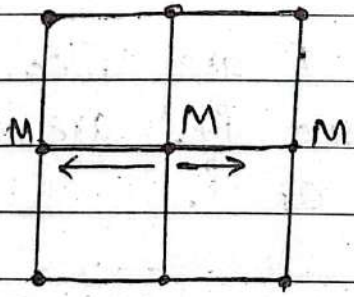
   It takes

   $2(p-1)$ steps $= O(p)$.

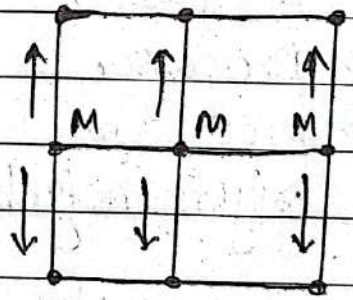
Example consider a 3×3 mesh with Message M originating at (2,2) then broadcast is done at

1. broadcast to node of row 2.
2. broadcast to node at each column.

(i) Message originates.



(ii) Message broadcast at same row.



(iii) Message broadcast at each column.

Packet Routing in Mesh:

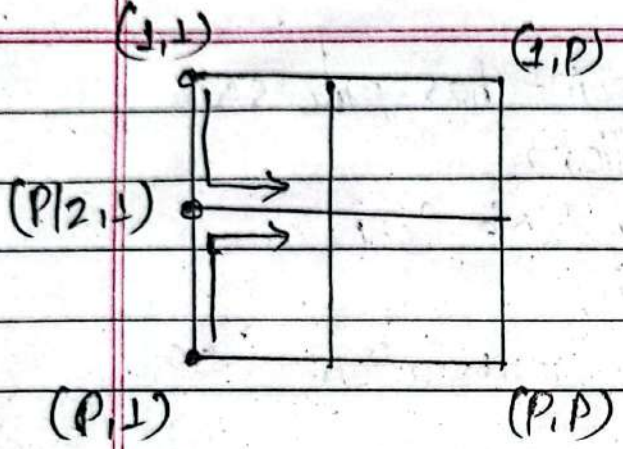
→ mesh is mxn grid, → each grid point has processor.
→ communication link are bi-directional.
→ each process can be labelled by tuple (i,j),
→ each process has local memory.
→ mesh has global clock for synchronization.

→ Packet routing is a primitive inter-process communication operation.
→ each packet has its origin an destination.
→ In a Mesh of pxp, q can be an arbitrary packet with (i,j) as origin and (u,v) as destination.
→ Then packet can travel in two steps:
  1. Travel along j (column) to row u.
  2. Travel along row u to destination (u,v)

→ If a packet with origin (1,1) has destination (P,P) then:
- Step 1: is done in (P-1) step and
  Step 2 also take (P-1) step
Hence the lower bound of the worst case routing time of any algorithm is:

$$(P-1) + (P-1) = 2P-2 = 2(P-1).$$

However, if all the packets originated at col 1 are destined for row P/2 then processor (P/2, 1) gets two packet at every time step and it can sent only one, the other has to wait.
∴ Hence queue size is needed as large as P/2.


(1,1)
(1,P)
(P/2,1)
(P,1)
(P,P)

Prefix Computation on Mesh

→ Consider a $\sqrt{p} * \sqrt{p}$ Mesh.

→ prefix computation can be done in following steps:

Step 1: Prefix computation row-wise.

Row i (for $i = 1, \ldots \sqrt{p}$) compute prefix of p elements.

Step 2: prefix computation column wise. last column only.

only $\sqrt{p}$ column computes the prefix of sum computed on step 1 and store on local Memory.

Step 3: Shift 1 down to column wise. last column only.

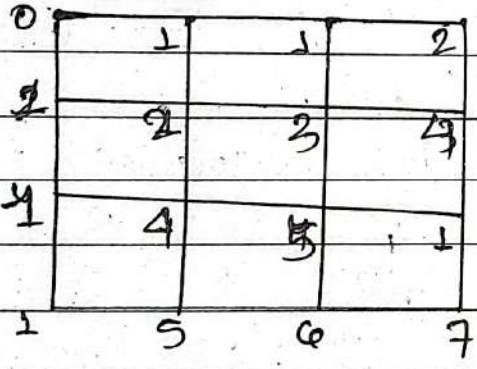
Shifting :→ $(i, \sqrt{p}) \longrightarrow (i+1, \sqrt{p})$

Step 4: Broadcast row wise to each processor.

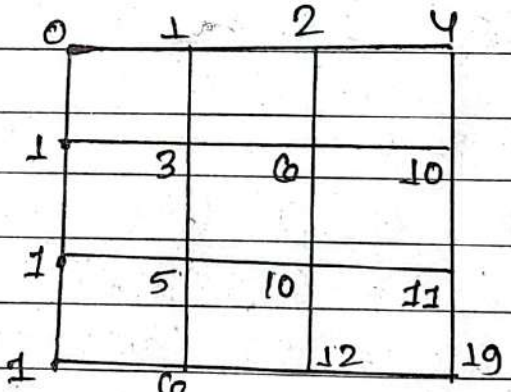
Broadcast

{0, 1, 1, 2, 1, 2, 3, 4, 1, 4, 5, 1, 1, 5, 6, 7}.

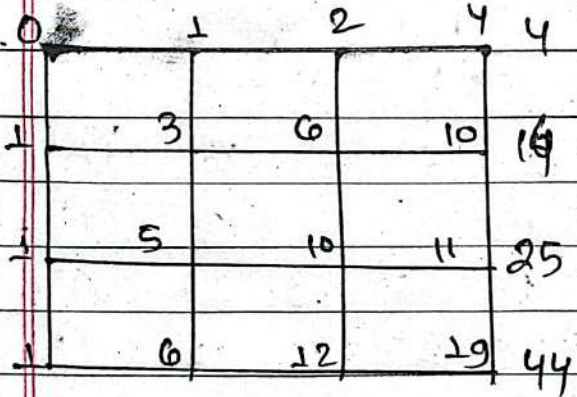
Ste Mesh:



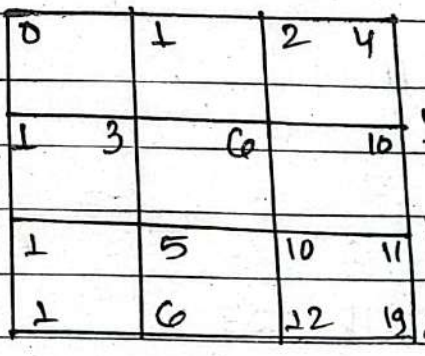
Step 1:



Step 2:



Step 3:



Step: 4

Hence: {0, 1, 2, 4, 5, 7, 10, 14, 15, 19, 24, 25, 26, 31, 37, 44} #

Sheer Sort (Mesh Sorting)

Algorithm:

```
for (i=1; i≤logp+1 ; i++)
{
    if i is even
    Sort the column (top to bottom)
    else
    Sort the row (alternative in order) snake.
```

Example:

i=1.

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

8	12	15	32	(Asc)
17	13	7	6	(Des)
2	16	19	25	(Asc)
18	11	5	3	(Desc)

↓

i=2 (colwise / Asc.)

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

i=3

2	3	5	11	(Asc)
12	8	7	6	(Desc)
13	15	17	25	(Asc) ⇐
32	19	18	16	(Desc)

↓

i=4 (Col / Asc)

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

⇒

i=5

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

#

Complexity: $O(\sqrt{p} \log p)$.

Mesh Algorithm for max. selection with n^2 Processors.

In a mesh network with n^2 process arranged in n×n grid, the problem of max. selection involves finding max value among all the processor.

1. initialize:
   each Processor holds unique value
2. Row wise max selection:
   In each row, perform a parallel comparion to find max. value,
   1. compare (i,j) with $(i, j+1)$
   2. Move larger value towards $(i, n-1)$
   3. repeate the process log n times.
   4. max value will be at end of row.
3. Column wise Max selection:
   1. Max value of each row is at last col
   2. perform similar parallel comparison col-wise.
4. After completing row wise and col wise Max. selection the global max will be at $(n-1, n-1)$.

eg: 3×3 Mesh.

1	3	5
7	2	9
4	6	8

⇒

1	3	5
2	7	9
4	8	8

⇒

1	3	5
2	6	8
4	7	9

$O(\log n)$

Mesh algo for max selection with 'n' processor.

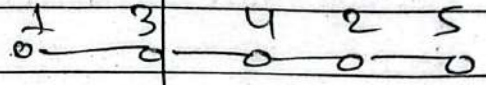
→ arrange a linear array (1-D) Mesh.
→ Compare element pairwise in series of step.

1. Initialize:
   → each processor holds a unique value.
2. Pairwise compare and exchange.
   → perform a series of comparison and exchange to propage the max value to the end of the list.
3. repeate step 2 ~~until~~ (k-1) time until max value reach end of the Array

$P_0 : 3$
$P_1 : 1$
$P_2 : 4$
$P_3 : 2$
$P_4 : 5$

$P_0 > P_1$ exchange.
$P_1 < P_2$ no ex.
$P_2 > P_3$ exch.
$P_3 < P_4$ no ex.

$P_0 > P_1$
exchange.



$P_1 < P_2$
no exchange

$P_2 > P_3$
exchange

$P_3 < P_4$
No

$O(n)$.

Embedding of network:

→ Mapping of one network to another is called embedding for example network such as ring, mesh and binary tree can be shown to be subgraph of hypercube.

→ let $G(V_1, E_1)$ and $H(V_2, E_2)$ are any two connected networks, an embedding of G to H is mapping of V_1 to V_2.

→ Embedding are important.
to simulate one network on another, using an embedding of G to H, an algorithm designed for G can be simulated on H.

- following terminologies are used in embedding:

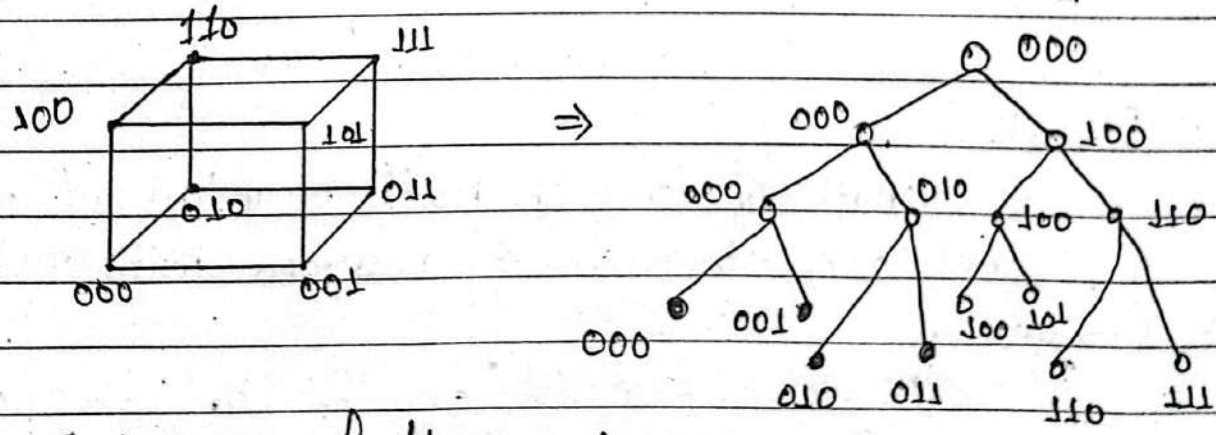
1. Expansion:
   It is obtained as V_2/V_1.
2. Dilation:
   length of longest path any link of one network structure is mapped with another.
3. Congestion:
   Number of path on the H_i corresponding to the links of G that it is on.
   (max. no. of times an individual path of S_2 is traversed for mapping S_1).

Example: Embedding of binary tree: Prefix Computation

We can embed a binary-tree to hypercube in many ways.

→ A p-leaf binary tree, whose $p = 2^d$ can be embedded into H_d.
→ A full binary tree with d-levels has $(2^{d+1} - 1)$ processor however d-dimension hypercube has 2^d processor
→ Hence, more than one processor of binary tree has to be mapped to a single processor of hypercube.

→ If tree leaves are 0, 1, 2, ---- P-1 then leaf is mapped to i^{th} processor of H_d.
→ Each processor of T is mapped to the same processor of H_d as its leafmost descendant leaf.



Embedding of Hypercube.

1. Expansion: $\frac{15}{8}$ → node on binary tree
→ node on hypercube

2. Dilation: 5

3. Congestion: 4

Date
Page

Hypercube:

Data Concentration on hypercube:

⇒ A hypercube is a d-dimensional structure, numbered using d-bits, Hence there are 2^d processor in d-dimensional hypercube, represented as H_d.

→ A hypercube has several variants like butterfly, shuffle-exchange network and cube connected cycle

→ Algorithm for hypercube can be adapted for butterfly network and vice-versa. Hence algorithm for butterfly network can be by applied to hypercube as well.

Data Concentration on hypercube:

→ on a hypercube H_d assume there are $k<p$ data item distributed arbitrarily with atmost one datum per processor. Then the problem of data concentration is to move the data into processor $0, 1, \ldots k-1$, of H_d

→ for this, we map the given hypercube to a butterfly network and then apply the concentration algorithm for butterfly network.

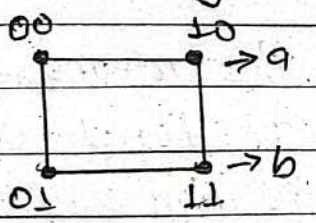


fig: a 2-D hypercube.

Consider a 2-d hypercube that has two data items at 10 & 11 processor be a & b respectively.

Then we map the 2-D hypercube to 2-d butterfly net.

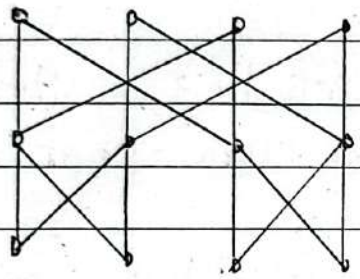


fig: mapping 2-D hypercube to 2-d butterfly network.

Step 1: The first step in data concentration is to compute prefix sum and to do so, we map the butterfly network B_2 to binary tree of depth 2.

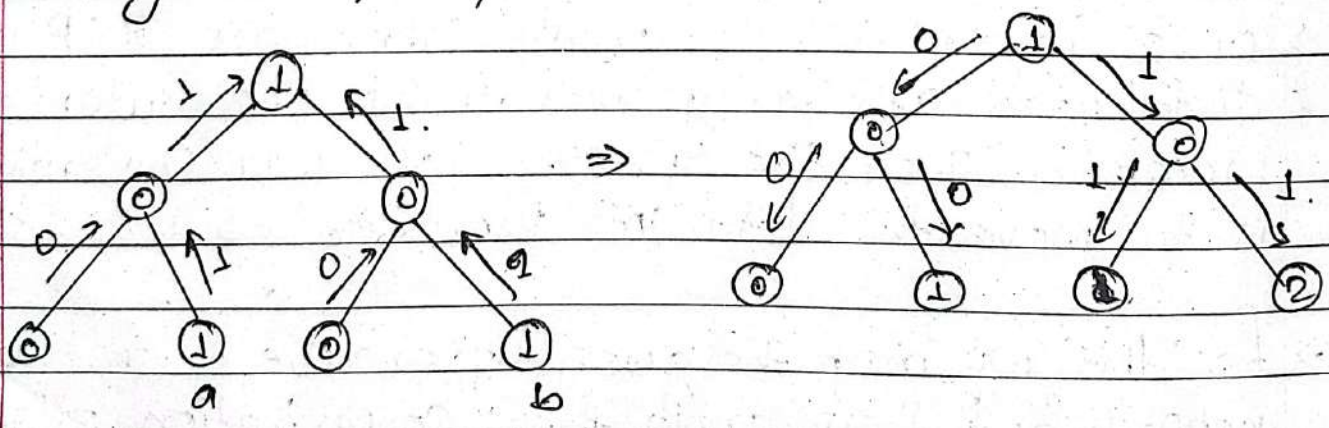
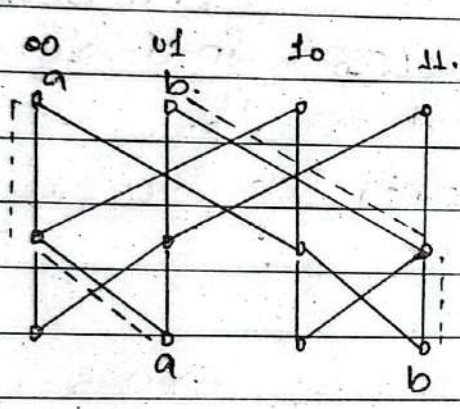


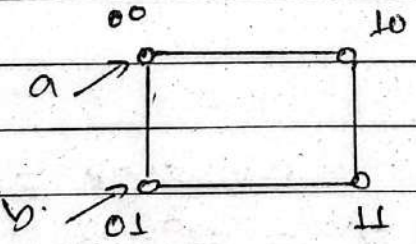
fig: Prefix Computation on binary tree

The prefix sum is $\{0, 1, 1, 2\}$. This means that the element a must be moved to position 1 and the element b must be moved to position 2.

In second phase, using the ~~graph~~ greedy approach the packets are routed to their destination. The path followed is shown dashed line.



finally we map the butterfly network to hypercube.



Hypercube.
features of hypercube.

→ A hypercube is a d-dimensional structure, numbered using d-bits. Hence there are 2^d processor in d-dimensional hypercube. Which is represented as H_d.
→ a hypercube may have variants like butterfly network, shuffle-exchange network, etc.

→ we define hypercube as following terms:
1. Dimension.
2. Coding of Processor.
3. Hamming Distance
4. Diameter.

Example: for d = 3 i.e. dimension = 3
no. of vertex = $2^d = 2^3 = 8$
no. of Processor = 8.

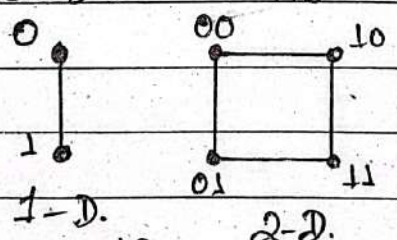
→ Coding of Processor = 000, 001, 010, 011, 110, 100, 101, 111,
→ Coding should be carried out in such a way that the difference of code of two adjacent Processor should be 1, which is called hamming distance.
→ The diameter of d-dimensional hypercube is d.
→ The bisection width of d-dimensional hypercube is $2^{(d-1)}$.
→ Hypercube is highly scalable architecture. Two d-dimensional hypercube can be easily combined to form a d+1- dimensional hypercube.

Characteristic of hypercube:

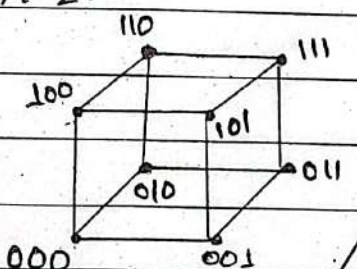
→ each processor code should alter by 1 bit only.
→ Degree of H_d = No. of interconnected processor of reference processor.
→ Processor coding is calculated by humming distance with no. of position change.
→ Sequential :
at one unit time any process can only communicate to one of it's neighbour.
→ Parallel :
at one unit time a processor can communicate with all of its neighbour.

features of hypercube:

→ The diameter of hypercube is $\log P$.
→ A hypercube can be generated by combining different hypercubes: $H_d = H_{d_1} \times H_{d_2}$.

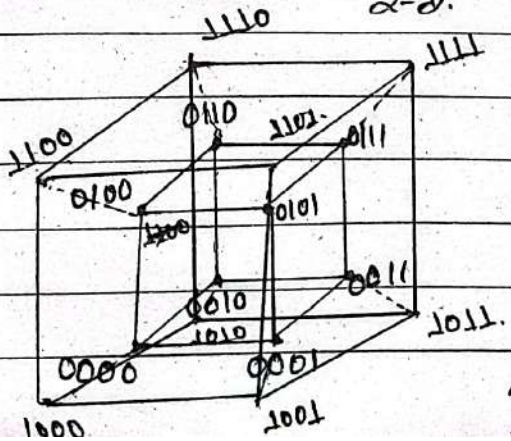


1-D. 2-D.



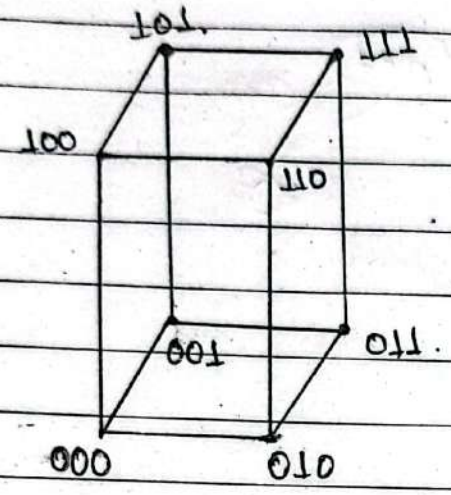
3-D.

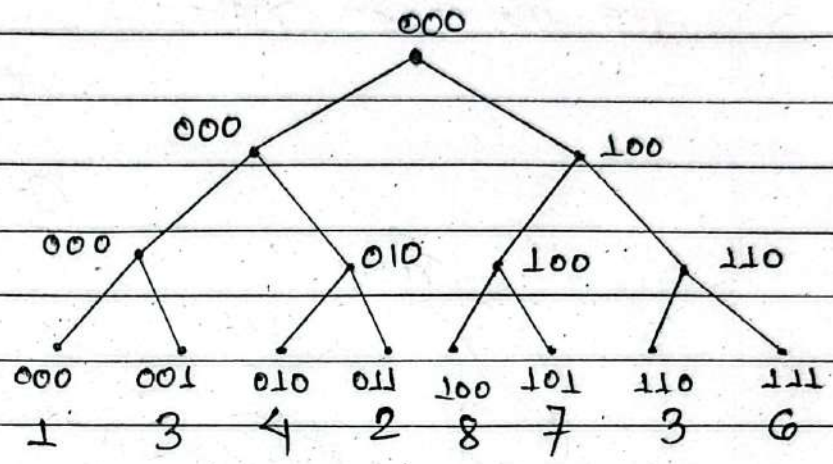
0	000	1	000
0	001	1	001
0	010	1	010
0	011	1	011
0	100	1	100
0	101	1	101
0	110	1	110
0	111	1	111



4-D.

Prefix Computation on hypercube:

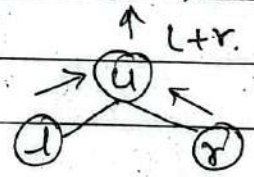




Phase I :

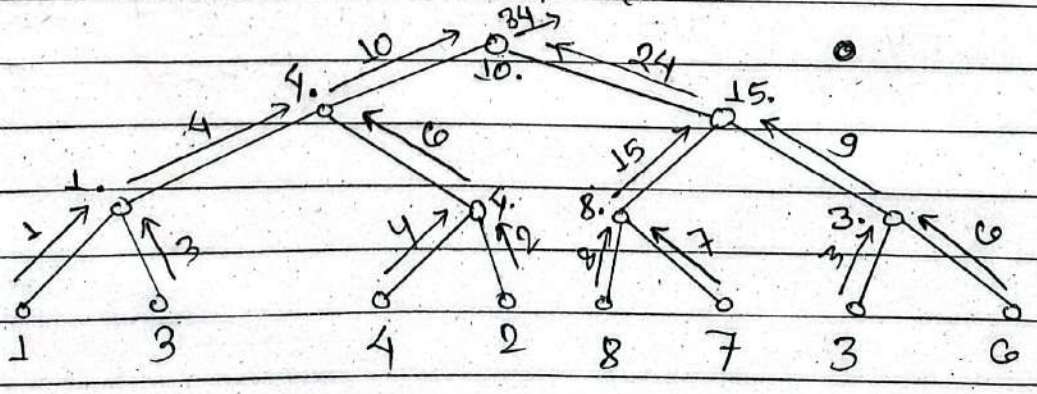
1: Place the values of the H_d to binary tree by mapping its pattern.

2. The leaf node will pass the value to the upper (u) (l,r)



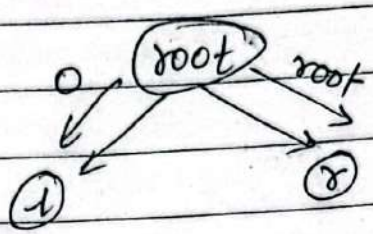
3. 'u' node will transfer the 'l+r' value to ~~its~~ the upper node. and store the received ~~value~~ left value.

4. The process will continue recursively until it reaches the root node.

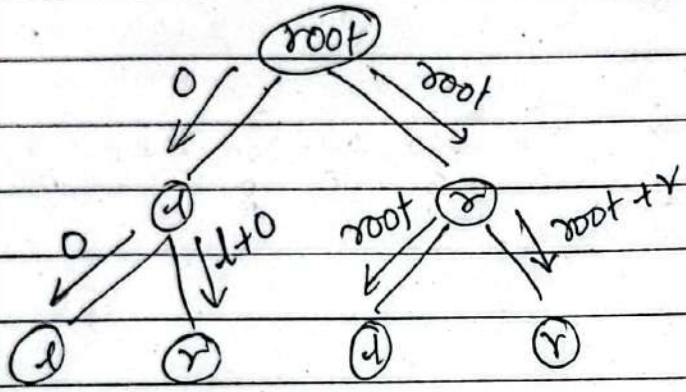


Phase II:

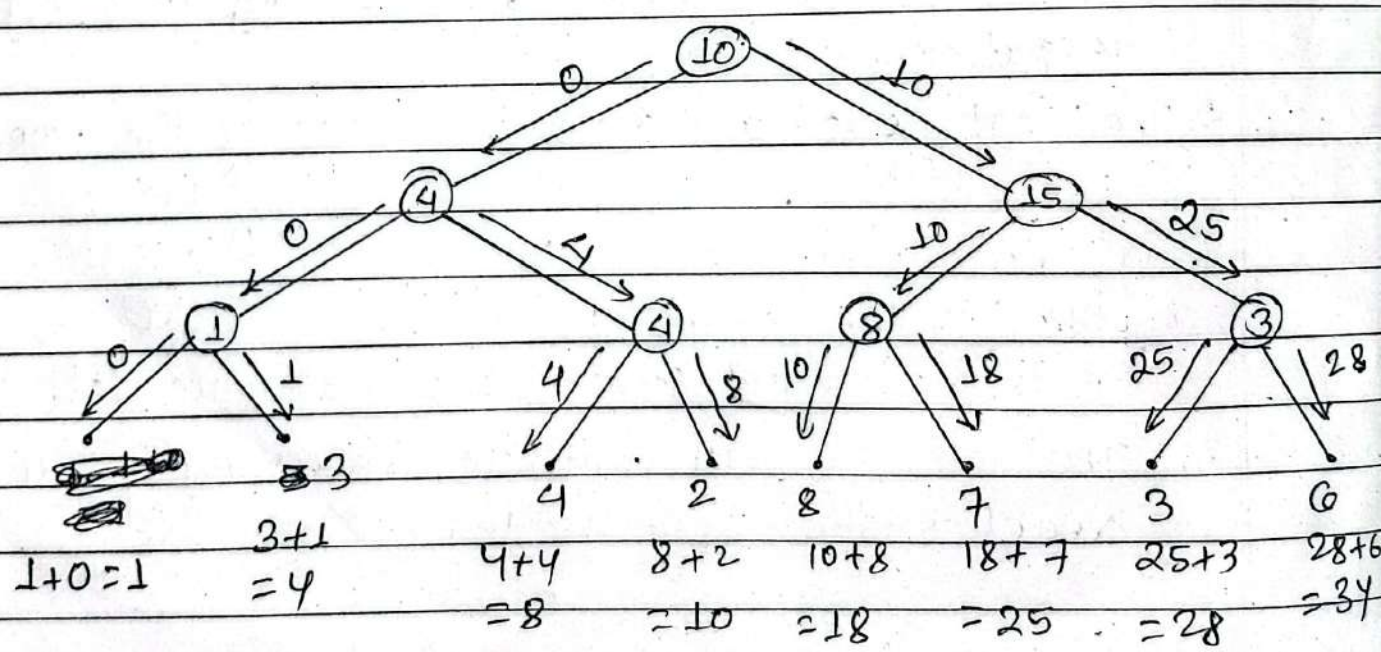
1. Starting from the root node if transfer 0 to left and it's value to right.



Step1



#



Learn - 95.

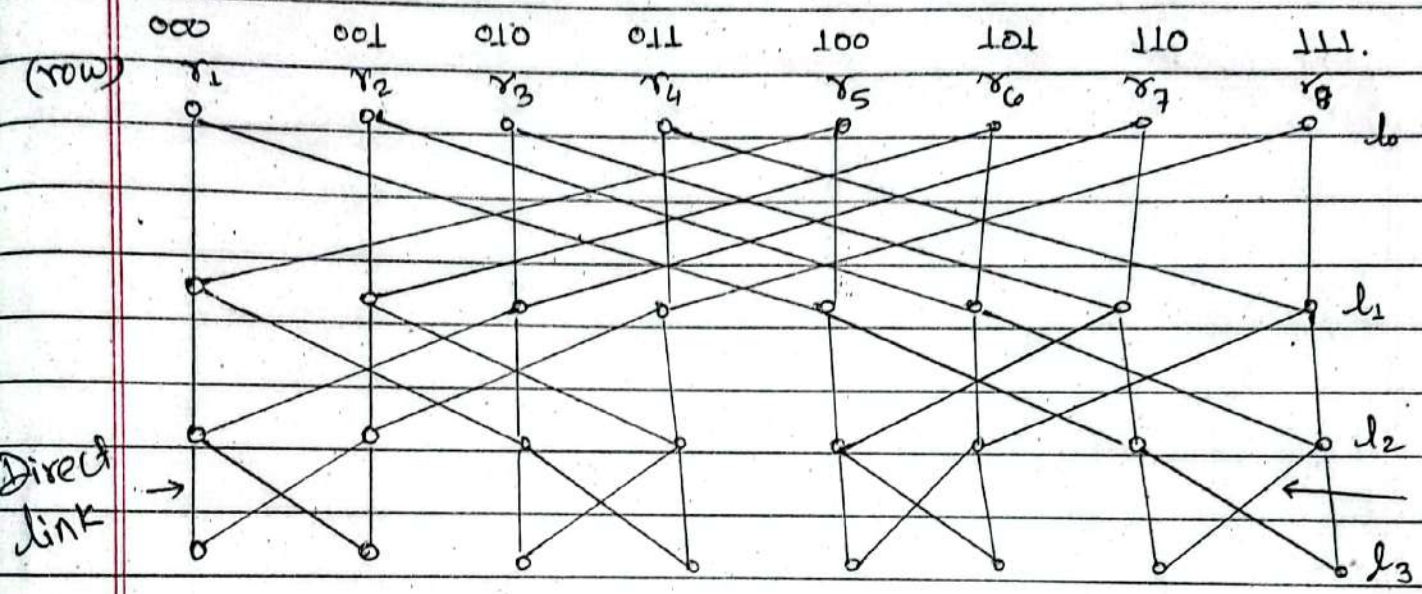
Date
Page

Broadcasting Problem of hypercube. -

→ The broadcasting in an interconnected network is to send a copy of message that originates from a particular processor to a subset of other processor.
→ It is a widely used technique for designing algorithms.
→ To perform broadcasting in H_d, we employ the binary tree embedding
→ Assume a message M is to broadcasted from root of the tree.
→ The root makes two copies. & send one to left child and other to right.
→ The child also do the same after receiving the message.
→ This process continues until all the processor has the message M.

Butterfly network: (B_d)

→ B_d is also one of the embedding technique to solve problems of Sorting data connection, prefin, selection, etc. In a massive parallel connected Network.



3d- butterfly network

1) Vertical line.
2) Cross link
3) Processor (p) = 32 → $(d+1)2^d = 4 \cdot 2^3 = 32$.

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

$3 \times 2^{3+1} = 3 \times 2^4 = 3 \times 16 = 48$

→ A Processor u is connected to v and w;

Where;

$v = \langle r, l+1 \rangle$ → direct connection.

$w = \langle r^{l+1}, l+1 \rangle$ → Cross connection.

$u = \langle r, l \rangle$

$0 \leq r \leq 2^d - 1.$

$0 \leq l \leq d.$

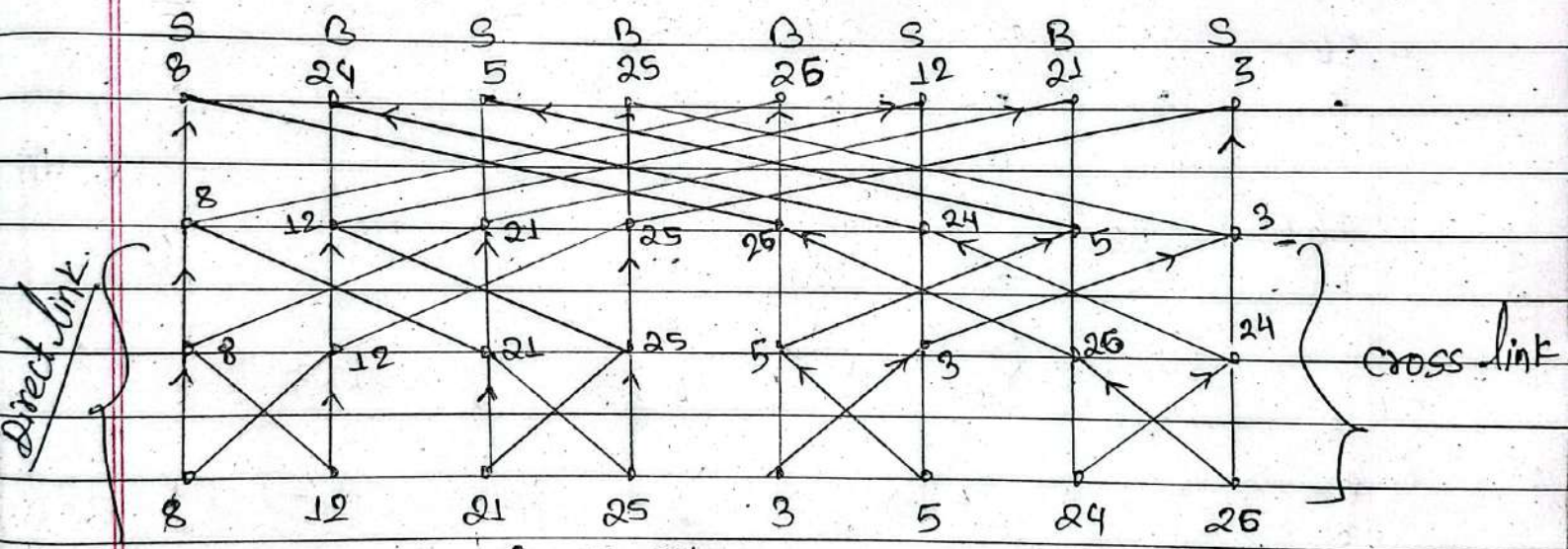
Odd-even merge Sort (Butterfly Network)

→ It consist of two phases.

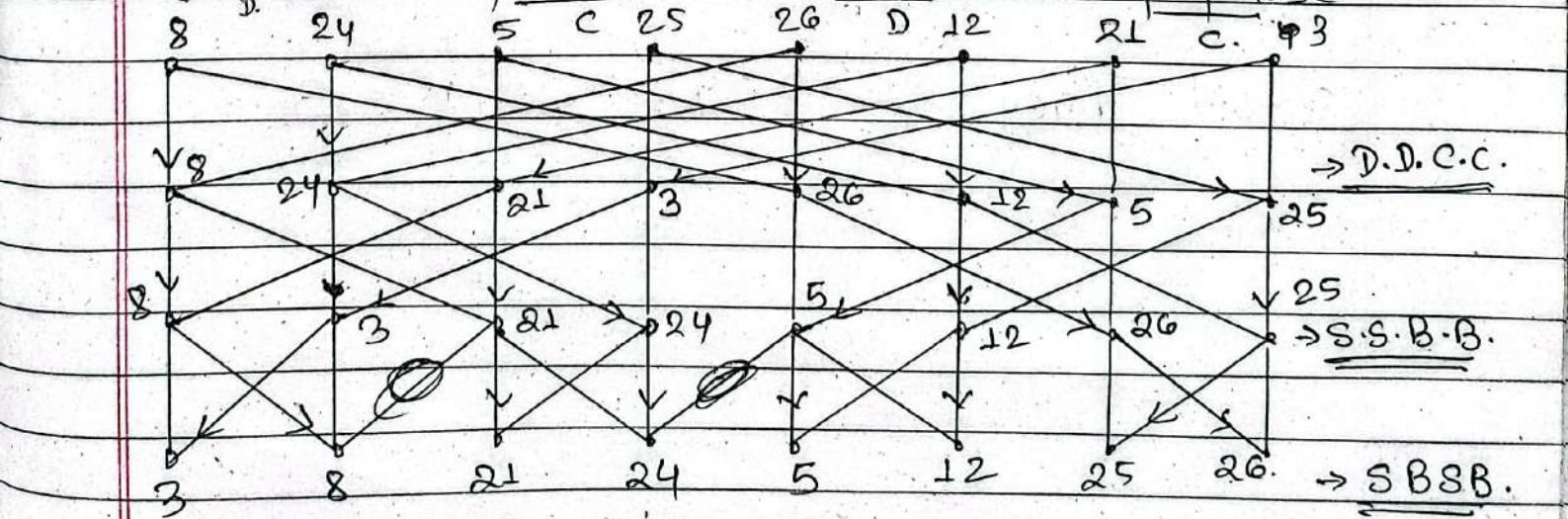
Phase I: separate odd and even into
[x_1 & x_2 given Sequence to O_1, E_1, O_2, E_2]

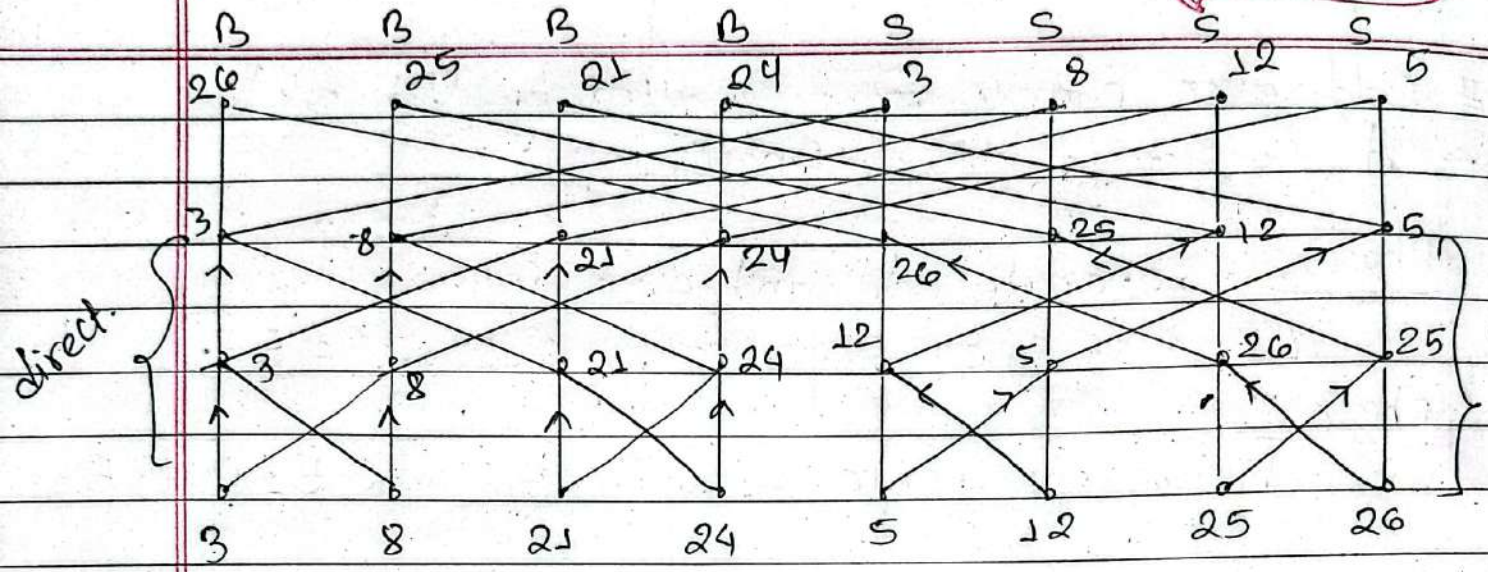
Phase II:
Recursively merge O_1 with O_2 and E_1 with E_2

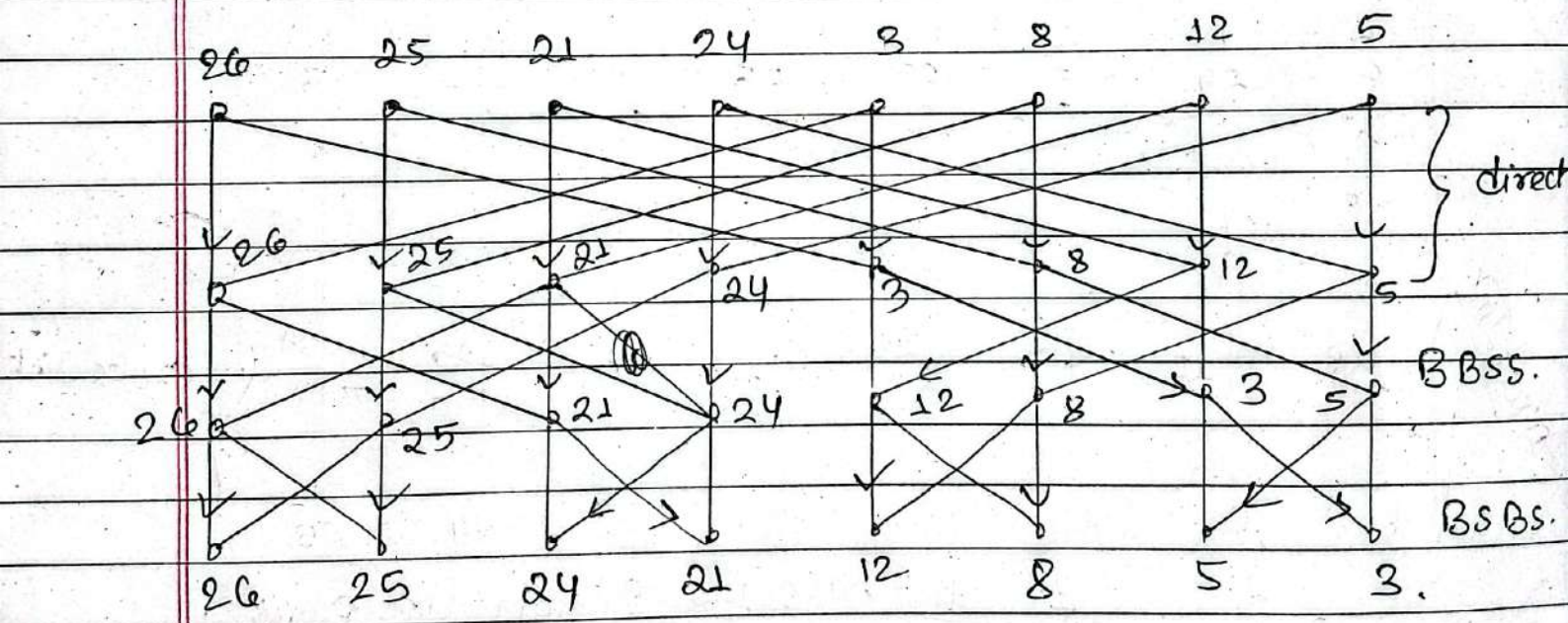
Question = 8, 12, 21, 25, 3, 5, 24, 26.



Phase :→ Buttom up phase.







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

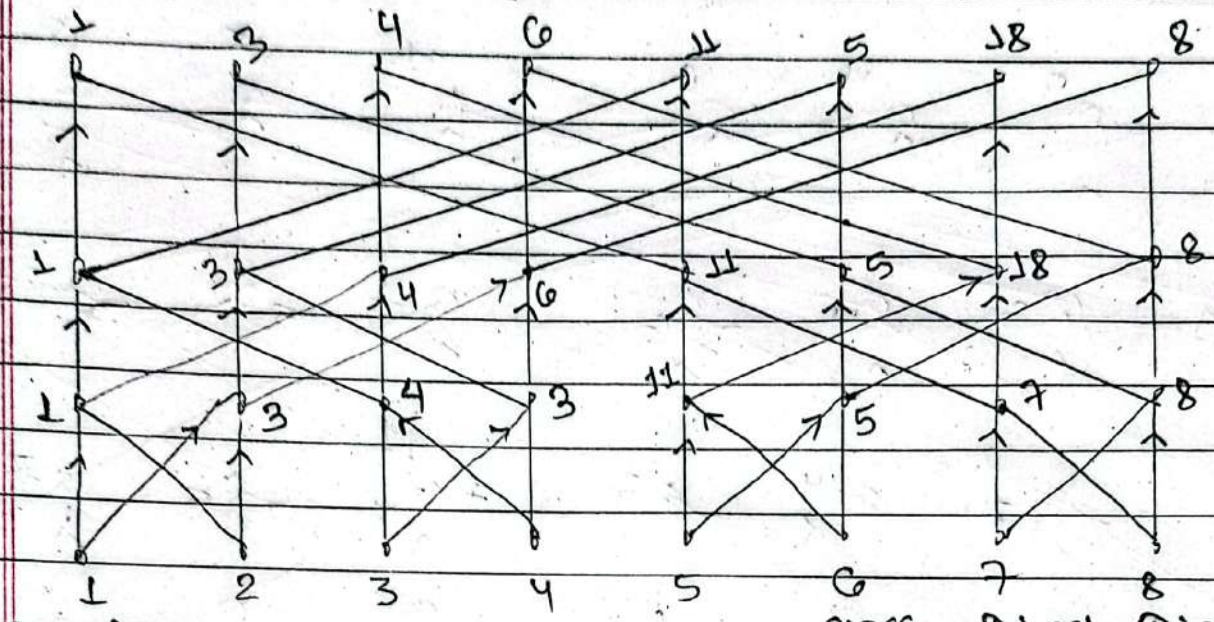
Prefix Computation

Phase I

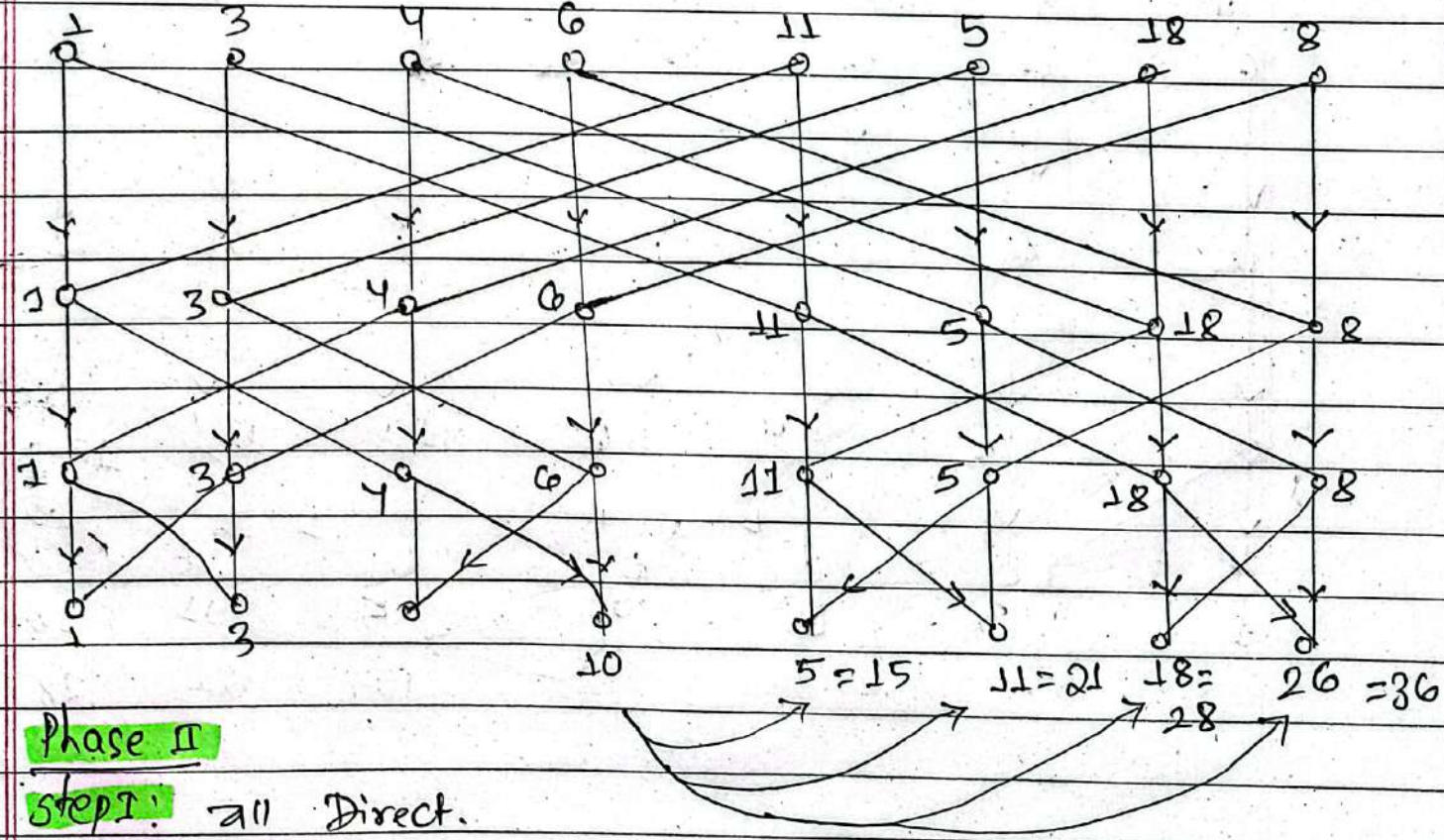
step1: Direct, sum, cross, cross, sum, cross, Direct, Direct.

step2: D, D, D, sum, D, D, sum, D.

step3: all Direct.



Direct ~~Cross~~ (Sum) Cross Cross Sum Cross Direct Direct.



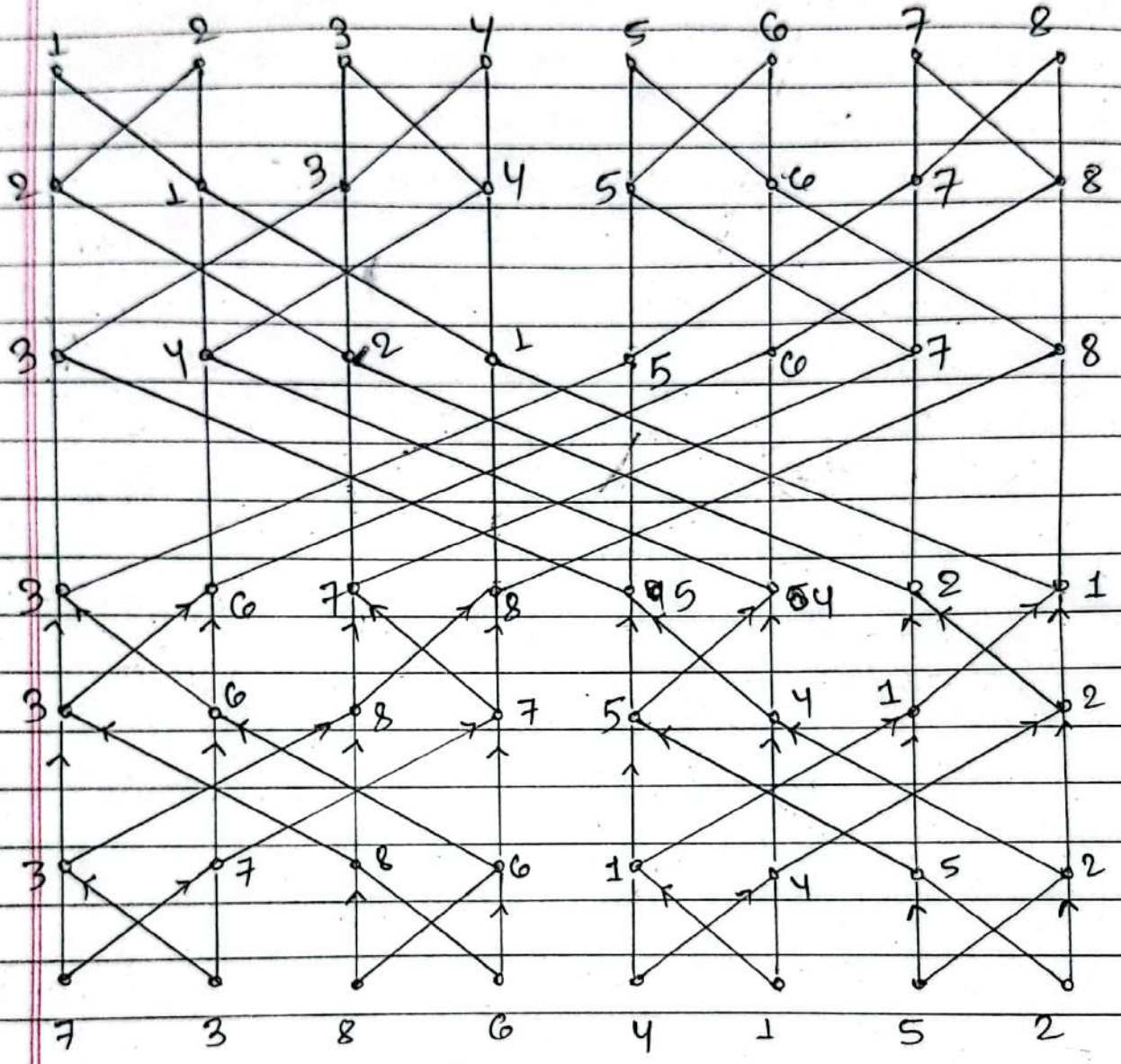
Phase II

step I: all Direct.

step II: all Direct.

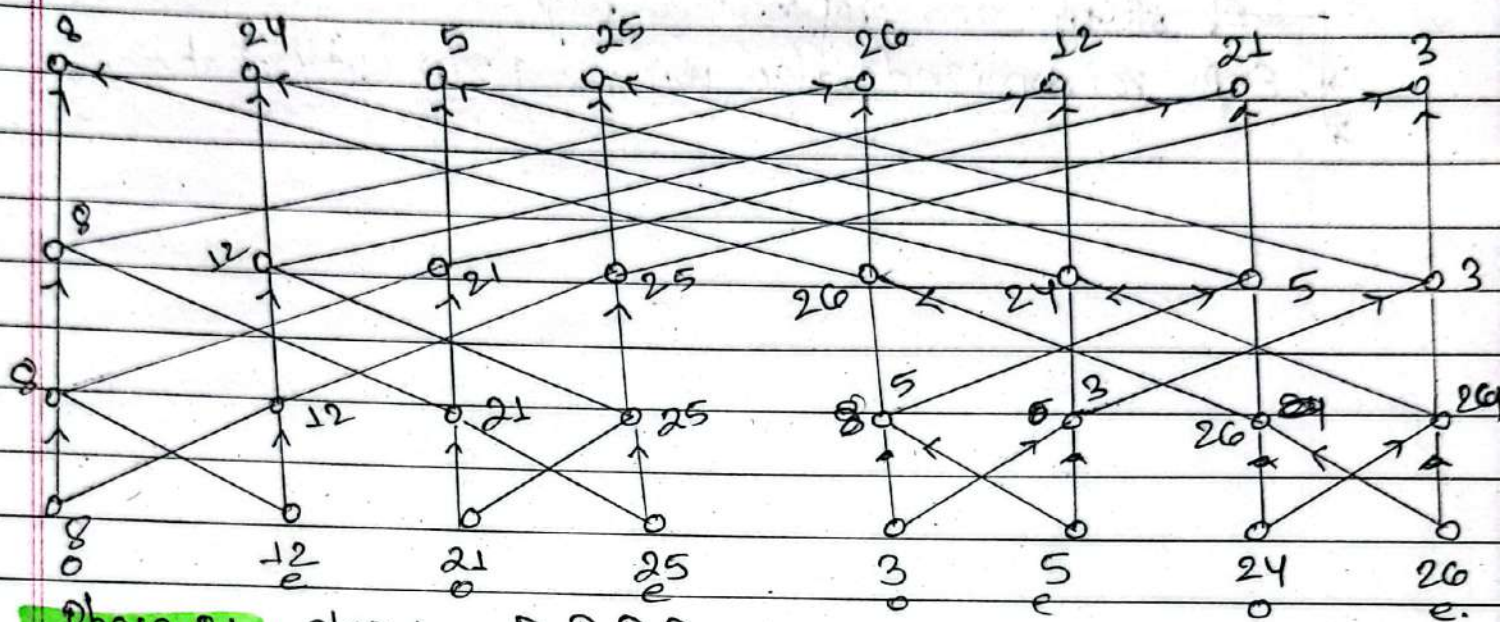
step III: D, D, cross, sum, cross, cross, Direct, sum.

Selection Sort:



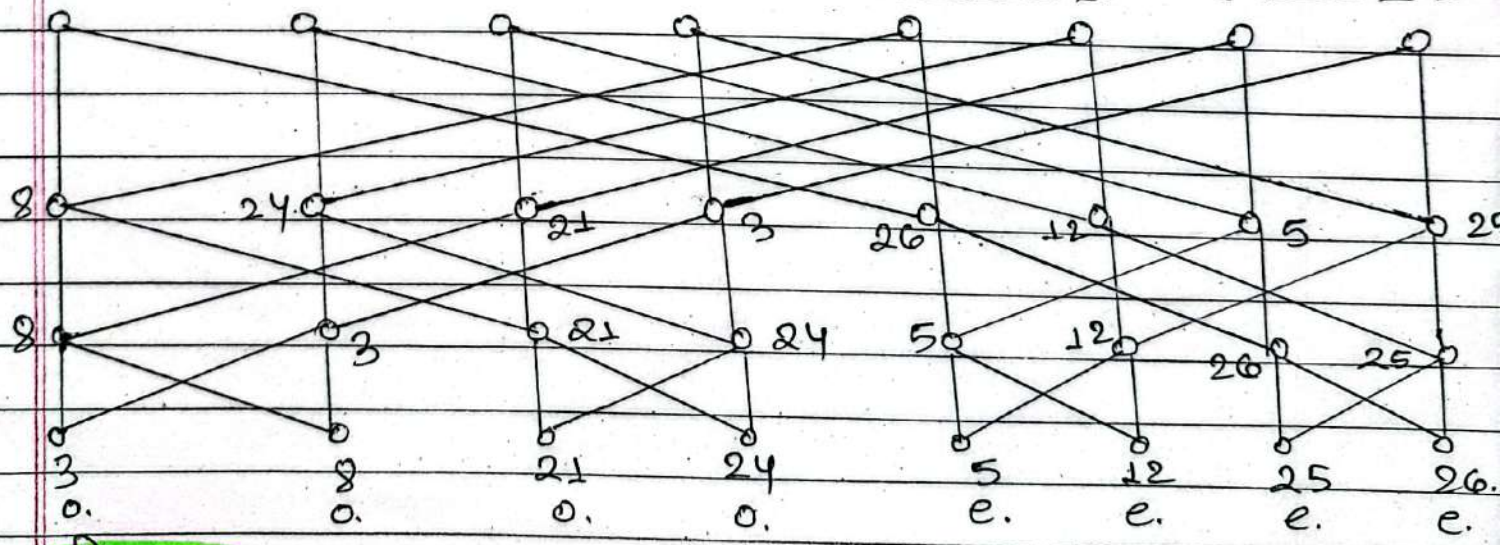
Step 1:	Cross, Cross, Direct, Direct.	Cross, Cross, Direct, Direct.
Step 2:	Small Small Big Big	Big Big small Small.
Step 3:	small Big small Big	Big small Big small.
Step 4:	Small Small Small Small	Big Big Big Big.
Step 5:	Small Small Big Big	small small Big Big.
Step 6:	small Big small Big	small Big small Big.

Odd Even Merge.



Phase I: Step 1: D.D.D.D. C.C.C.C.
Step 2: D.D.D.D. C.C.C.C.
Step 3: S.B.S.B. B.S.B.S.

[8 D 24] [5 C 25] [26 D 12] [21 C 3]



Phase II: Step 1: D.D. C.C. D.D. C.C.
Step 2: S.S. B.B. S.S. B.B.
Step 3: S. B S B. S. B. S. B.

Unity:

Butterfly net:

→ Embedding tech.
→ Solve problem of sorting data, prefin computation, selection, etc.
→ It uses Massive Parallel connected networks.

→ A process u is connected to v & w
where $v = \langle r, l+1 \rangle$ direct connection.
$w = \langle r^{l+1}, l+1 \rangle$ cross connection.
$u = \langle r, l \rangle$
$0 \leq r \leq 2^d - 1$.
$0 \leq l \leq d$.

→ **Butterfly net has:**
→ vertical line that shows direct connection between the processor of a same r.
→ cross link that shows cross connection between the process of different r.
→ Processor (p)
$(p) = (d+1) \cdot 2^d =$
total links $= d \cdot 2^{d+1}$ }.

Odd-even Merge Sort in butterfly net:
#. oddeven merge sort has 4 butterfly net.
Phase I II, III & IV.

Prefin Computation using butterfly net:
It has two phase
phase I & II

Selection sort in butterfly network.
It has merged two B.N. into one #

Embedding of network:

→ maping of one network to another.

→ example hypercube can be shown as binary tree, mesh etc. which considered as subgraph.

→

for example $G(V_1, E_1)$ & $H(V_2, E_2)$ are any two connected networks.

embedding means mapping of V_1 to V_2.

→ using embedding of G to H.

we can simulate algo designed to G on H also.

→ Terms used in embedding:

1. Expansion:

→ It is obtained as V_2/V_1.

2. Dilation:

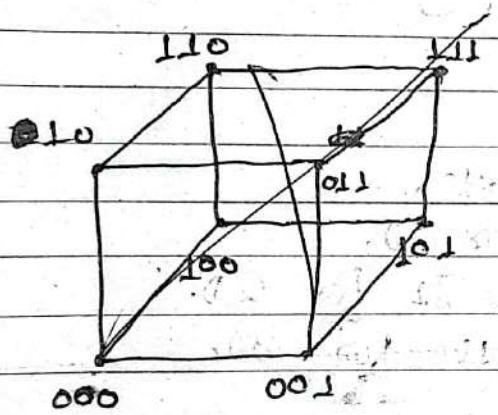
→ length of longest path any link of one net. structure is mapped with another.

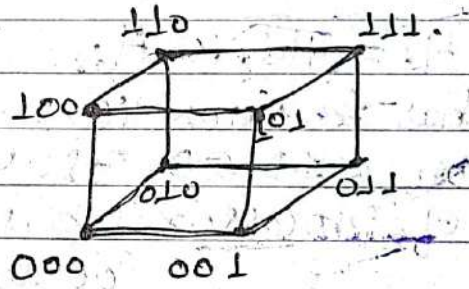
3. Congestion:

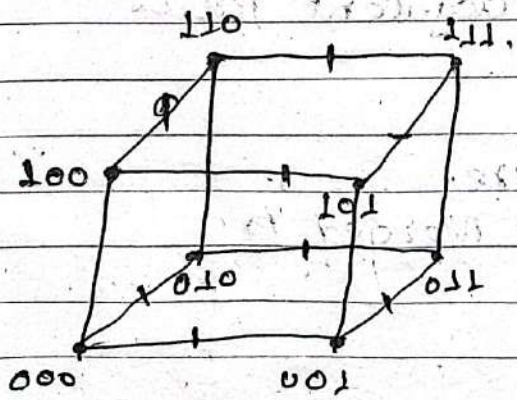
→ max. no. of times a path on structure 2 is traversed while mapping 81.

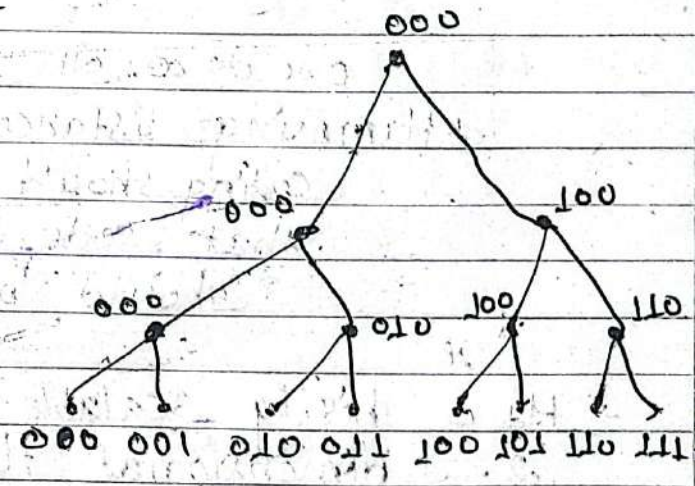
Embedding of binary tree.

1. p-leaf binary tree can be embed into H_d hypercube.
2. full binary tree with d-leaves has $2^{d+1}-1$ processor
2. d-min dimension. H_d has 2^d process.
3. more than one processor of B.T. has to be mapped with single processor of hypercube.
5. each processor of T is mapped with same processor of H_d at its leafmost descendent.









$\Rightarrow$

Expansion = $\frac{15}{8}$ → nodes of T.
→ nodes of H_d

Dilation = 6

Congestion = 7

→ features of hypercube.

→ d-dimensional structure. numbered using d bits.
→ 2^d processor in d-dimensional Hd.
→ represented as Hd.
→ Hd may have variants like butterfly net or shuffle-exchange net.
→ diameter of Hd is log p. /and/or d.
→ Hd can be generated by combining differ. Hypercubes.

$$Hd = Hd_1 \times Hd_2.$$

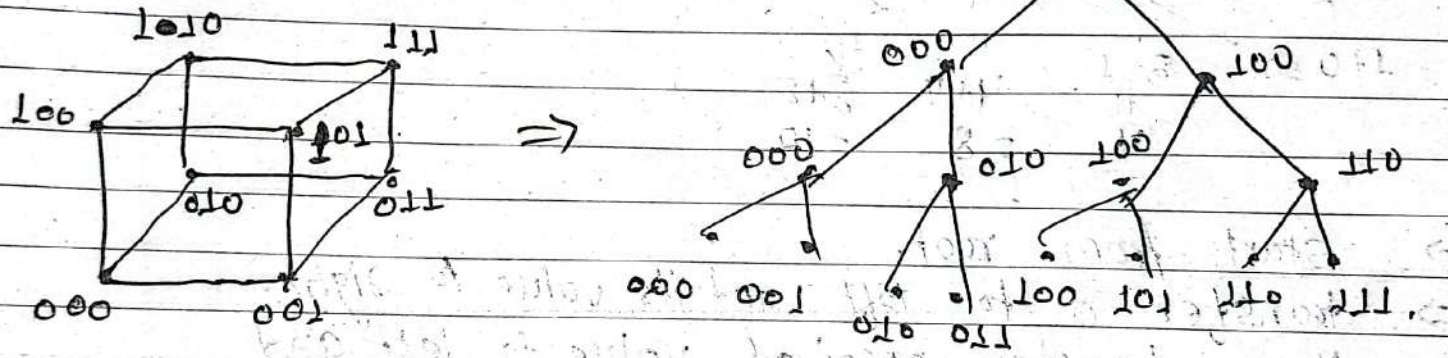
→ Hypercube has features like
* Dimension : 1D 2D 3D. 4D, etc.
* Coding of processor: 0,1 for 1D.
00 10 01 11 for 2D.
000 010 001 011 100 110 101 111 for 3D.
* Hamming distance 1.
coding should be done in such a way that code diff^e bet^n adjacent Proce should be 1.

→ Hd is highly scalable architecture.
2. d-dimensional Hd. can be merged to give d+1- dimensional Hd.

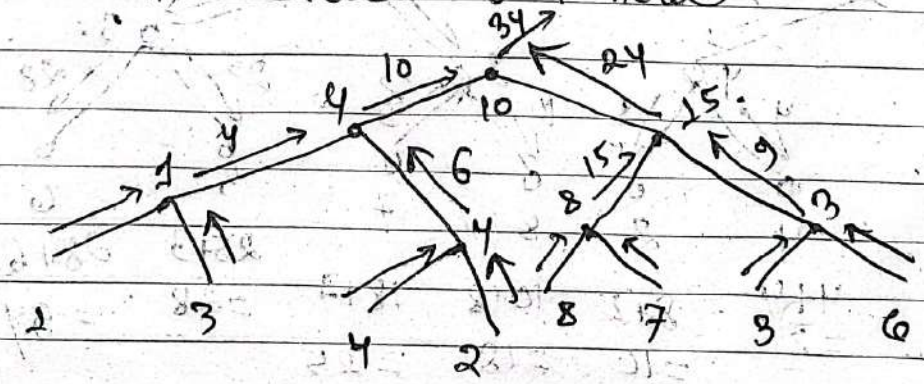
Characteristic of hypercube:

→ each processor code should alter by 1 bit only.
→ Degree of Hd = no. of interconnected P. of reference P.
→ Processor coding is don by following hamming dist.
→ sequential communication / parallel communication.
→

Prefix computation:



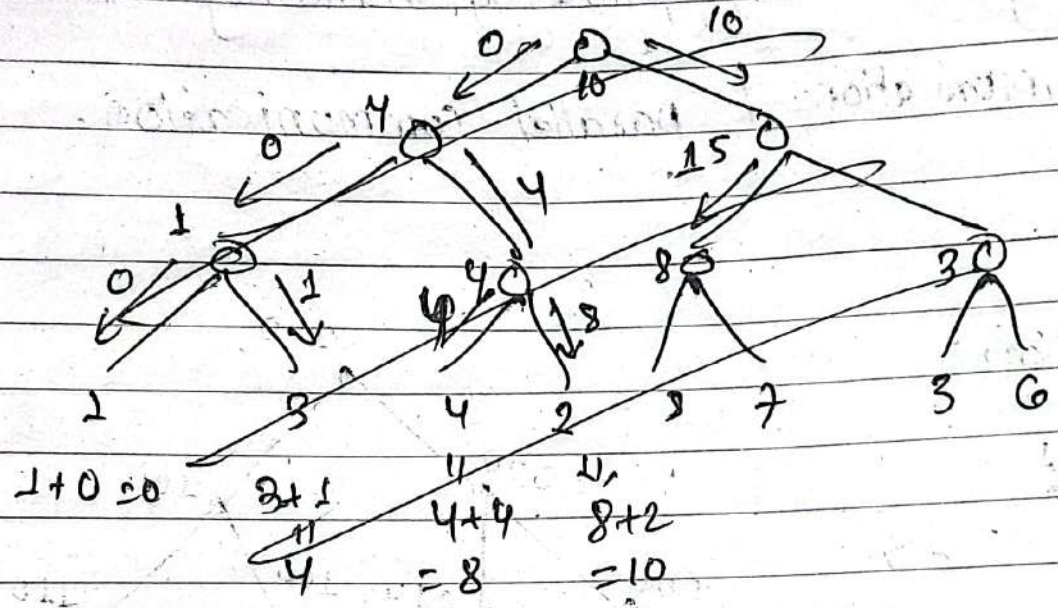
Phase 1: → place value of Hd to T by mapping its pattern.
(1,8) → leaf node will pass its value to upper node (4).
→ u keep left value and pass Hr to its upper node.
→ repeate until reaches root node.



Phase II

→ start from root node.

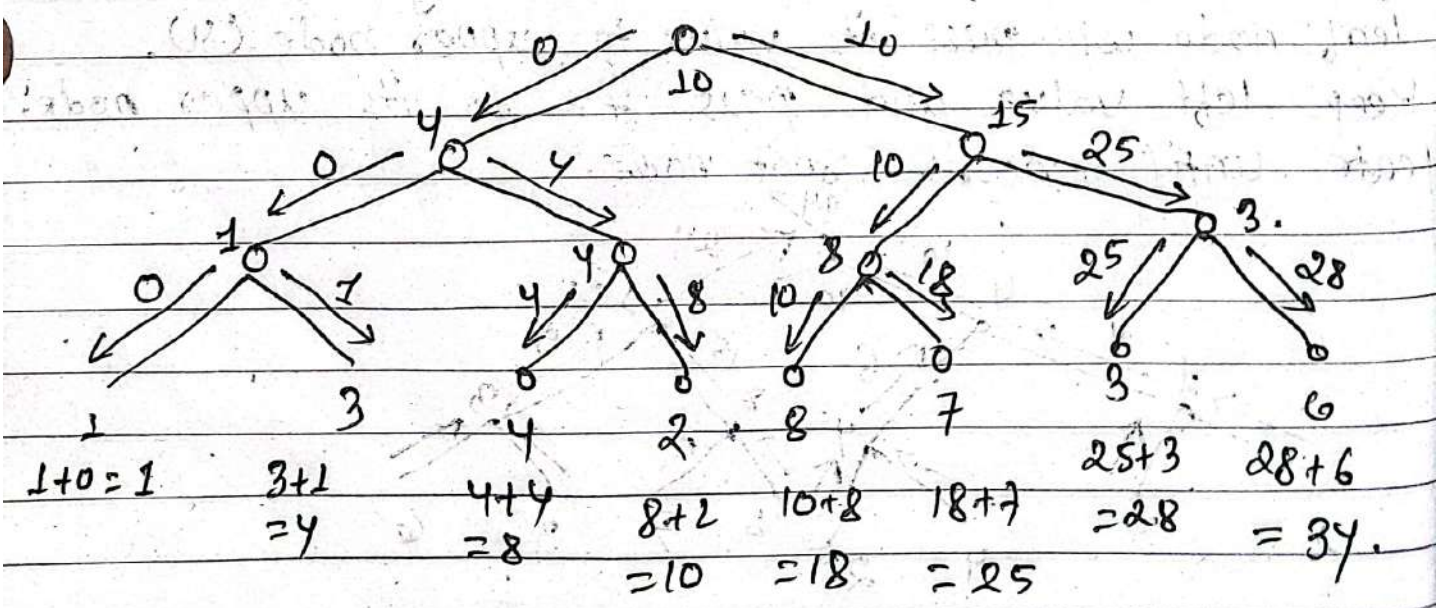
→ transfer 0 to left and its value to right plus received val



→ start from root.

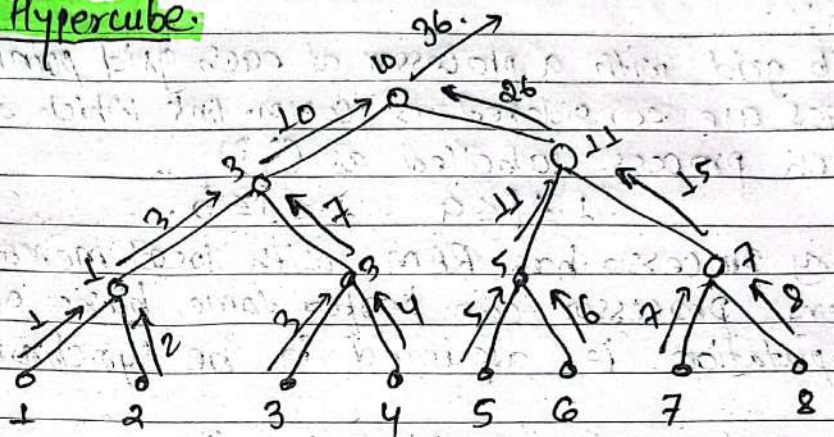
⇒ transfer 0 to left and its value to right.

⇒ then transfer received value to left and received value + its own value to right.

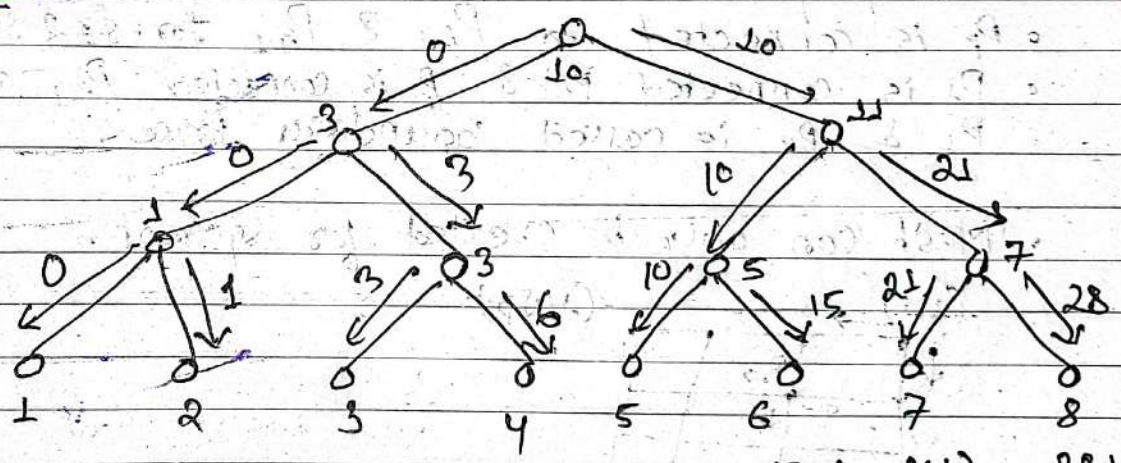


Prefix Comp. on Hypercube.

→ Phase I



Phase II



1+0 = 1 | 2+1 = 3 | 3+3 = 6 | 6+4 = 10 | 10+5 = 15 | 15+6 = 21 | 21+7 = 28 | 28+8 = 36.

#

Mesh:

→ a x b grid with a processor at each grid point.
→ edges are considered as comm. link which are bi-direction
→ each process labelled as (i, j)

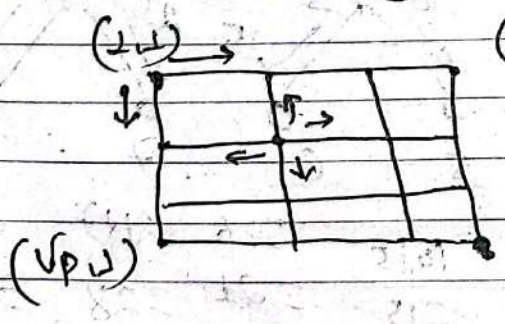
$$1 \le i \le a \quad 1 \le j \le b.$$

→ Every processor has RAM with local memory.
→ every processor can perform some basic operations.
→ Computation is assumed to be synchronous.
→

A related model for Mesh is linear array Model with p processor. $(1, 2, \ldots p)$

- P_i is connected to P_{i-1} & P_{i+1} for $2 \le i \le P-1$
- P_1 is connected P_2 & P_i is connected P_{i-1} only
- P_1 & P_i is called boundary proc

- Mesh can only be created for Sq. root.



Broadcasting in Mesh:

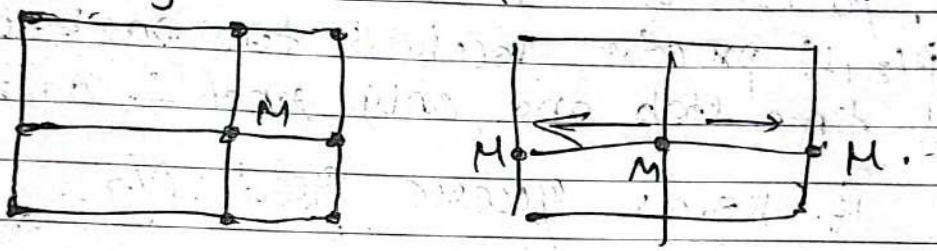
→ in case of $\sqrt{p} \cdot \sqrt{p}$ Mesh.

→ lets assume Message M originate at (i,j)

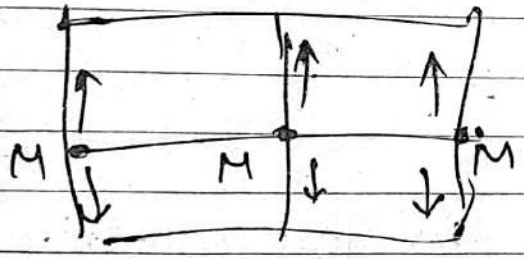
1. first broadcast to every processor of row i
2. then broadcast to all the proc of each col.

for a 3×3 mesh.

consider Message M at 2×2



Message originate Broadcast at same row.



Broadcast at each column.

Packet routing in Mesh.

→ n×n grid, processor at each grid point, com link bidi

→ labeled by tuple (i,j), local Memory, global Clock Sync

→ Packet routing - primitive inter proc comm. operation.

→ each packet has origin & destination.

→ In a p×p Mesh an arbitrary packet q with origin (i,j) and destination (u,v) can travel in two ways.

1. Travel along coln j to reach row u.
2. Travel along row u to reach dest. (u,v).

→ a packet origin (1,1) has dest (p,p)
step 1 takes (p-1)
step 2 takes (p-1).

∴ lower bound of worst case =
$(p-1) + (p-1) = 2p-2 = 2(p-1)$.

→ if all the packets of col 1 are destined to P/2 then (P/2,1) process receives two packet at every time step and only send 1 packet.
hence it needs queue sized P/2.

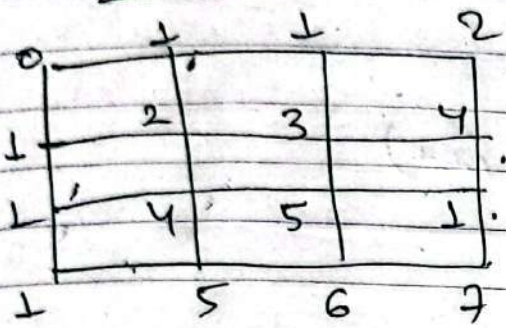
Prefix Computation on Mesh

consider $\sqrt{p} * \sqrt{p}$ Mesh.

* Prefix computation steps:
1. row wise computation.
2. column wise computation: for last col. only and store the result on local Memory.
3. Shift 1 down column wise for last col only.
4. Broadcast row wise to each processor.

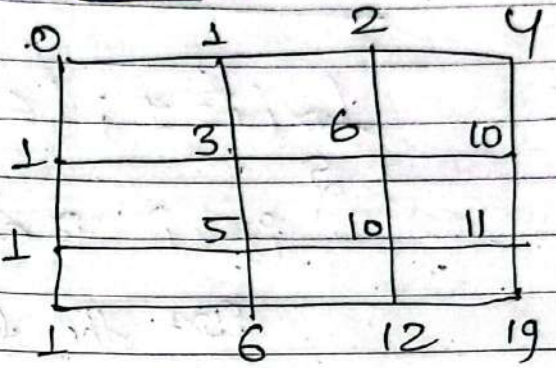
S {0, 1, 1, 2, 1, 2, 3, 4, 1, 4, 5, 1, 1, 5, 6, 7}

Mesh.

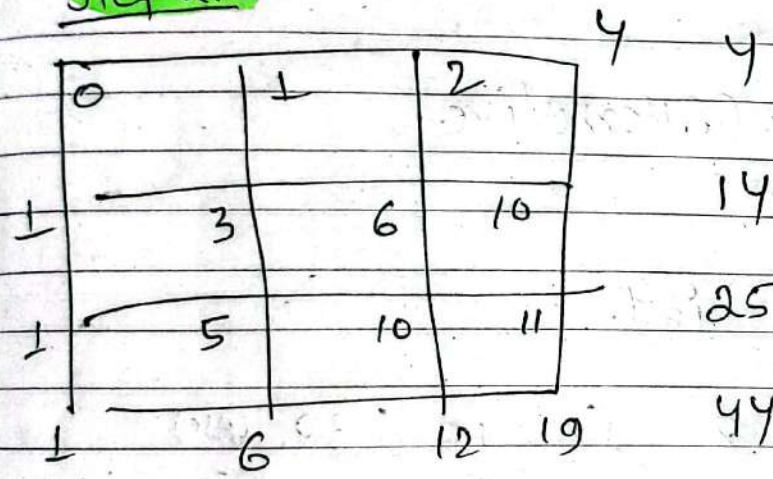


⇒

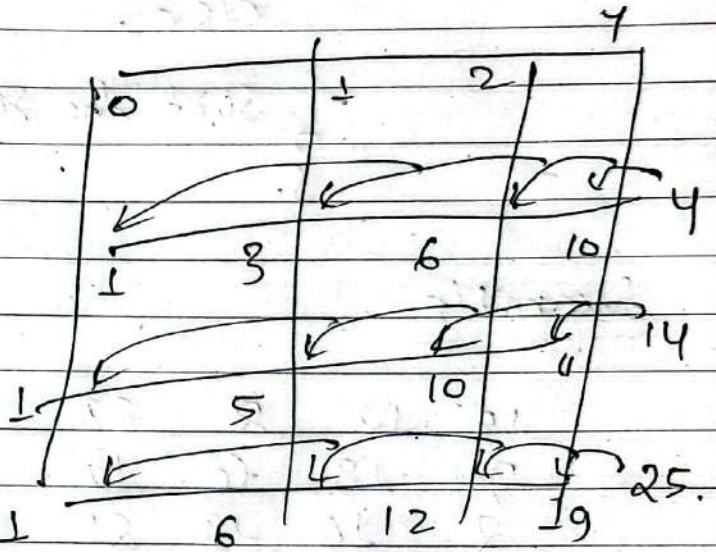
Step: 1

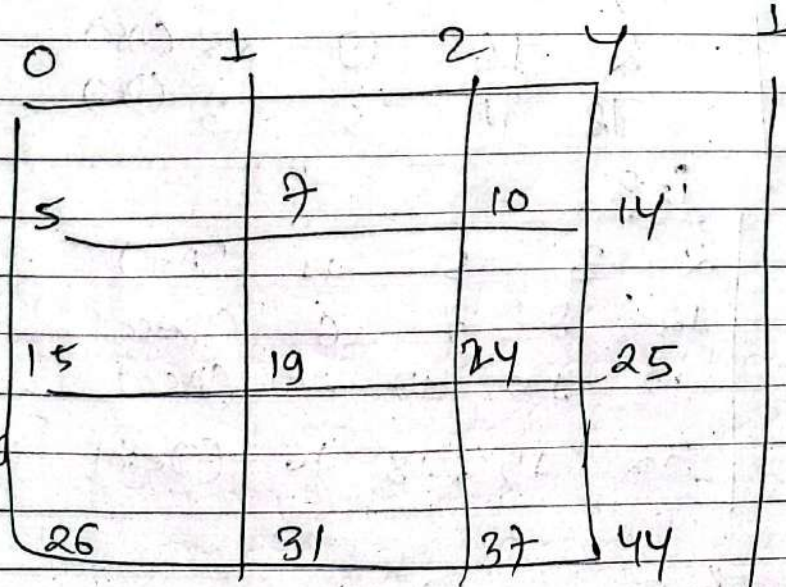


Step 2:



4
14
25
44





Ans

Sheer sort (Mesh Sorting)

Algo:

```
for(i=1; i<=log+1 ; i++)
{
  if i is even.
    Sort the Colum.
  else.
    Sort the row. (Alternative.
}.
```

Example

i=1.

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

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

i=2

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

i=3

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

i=4 =

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

i=5 =

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

$O(\sqrt{p}\log p)$