Jan 20; 25

DATE: / /20
PAGE NO.

DAA

Algorithm

finite set of well-defined I/S. Steps followed to complete a task.

* Characteristics

- finiteness – should terminate
- Definiteness – definable steps
- Input – 0 or more I/P
- Output – atleast 1 O/P
- Effectiveness – executable algorithm

* Applications

- Sorting data
- Searching in database
- Graph problems
- Cryptography and Security
- AI & ML

* Algorithm Design Paradigms

- Divide & Conquer
  - eg. → Merge Sort
  - → Binary Search
- Dynamic Programming
  - eg. → Fibonacci Series $F(n) = F(n-1) + F(n-2)$
  - → Knapsack problem
  - → longest common subsequence
- Greedy Method
  - eg. → Kruskal's Algo. → Dijkshtra
  - → Prim's algo. → Huffman Co

DATE : / /20
PAGE NO.

22-1-25

→ Back tracking
  • Brute force Method • Bounding function
• BT known as Constraint Satisfying Method
  • it follows DFS method
eg. • N-Queen problem (Queen in chess-board)
  • Sudoku solver

→ Branch and Bound
  • it follows BFS
eg. • Travelling Salesman problem
  • 0/1 knapsack

Concept of Algorithmic Efficiency
• measured by time and space, which depends on input size (n).
• order of growth →

• space → fixed and variable

Time - Space Trade off
↳ compromise
T↑, S↓ or T↓, S↑ ; i.e., $T \propto \frac{1}{S}$

eg.

DATE: / /20
PAGE NO.

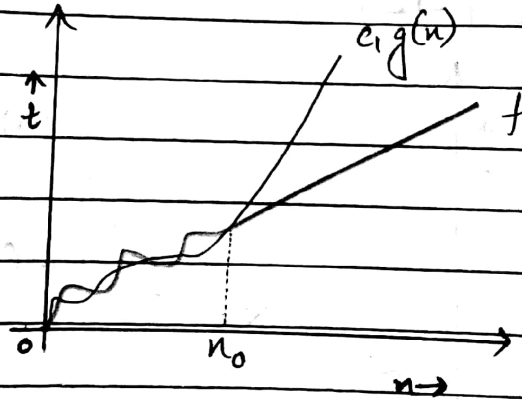
Types of run-time analysis

- Worst case → Big-O (O) → upper bound
- Best case → Big-Omega (Ω) → lower bound
- Average case → Big-Theta (Θ) → both

Asymptotic Notations

used to measure the efficiency of algo. w.r.t. time and space.

- **Big-O (O)**



closest upper bound (worst case)

$$f(n) \leq c_1 * g(n)$$

→ Time of loops, operations etc.
→ Diff.

$$f(n) \leq c_1 * g(n)$$
$$2n^2 + n \leq c_1 \cdot g(n)$$

max.,
$$n^2 \leq c_1 \cdot n^2$$

divide by n^2,
$$\frac{2n^2}{n^2} + \frac{n}{n^2} \leq c_1 \cdot \frac{n^2}{n^2}$$
$$2 + \frac{1}{n} \leq c_1$$

$2.-- \leq 3$, hence, $c_1 = 3$

$$2n^2 + n \leq 3n^2$$

for n_0,

$$2n^2 + n \leq 3n^2$$
$$n \leq 3n^2 - 2n^2$$
$$n \leq n^2$$

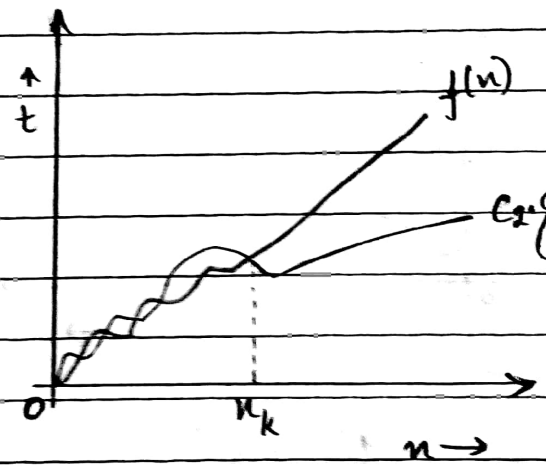
divide by n,

$$1 \leq n$$

or $\boxed{n \geq 1}$

hence, $\boxed{n_0 = 1}$

- **Big - Omega (Ω)**



closest lower bound (best case)

$\boxed{f(n) \geq c_2^* g(n)}$

$$f(n) \geq c_2^* g(n)$$
$$2n^2 + n \geq c_2^* g(n)$$

max.,

$$n^2 \geq c_2 \cdot n^2$$

divide by n^2,

$$\frac{2n^2}{n^2} + \frac{n}{n^2} \geq \frac{c_2 n^2}{n^2}$$

$$2 + \frac{1}{n} \geq c_2$$

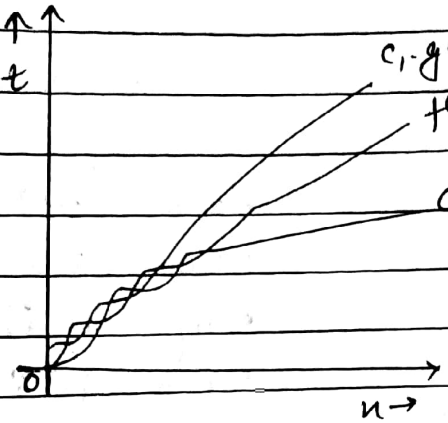
$2. \ldots \geq 2$, hence, $c_2 = 2$

$2n^2 + n \geq c_2 n^2$

$2n^2 + n \geq 2n^2$

$n \geq 0$

- **Big - Theta (Θ)**



(average case)

both lower & upper bound

$c_1 g(n) \geq f(n) \geq c_2 g(n)$

$3n^2 \geq 2n^2 + n \geq 2n^2$

✱ Divide and Conquer

It involves 3 basic steps -

- Divide
- Solution
- Combine

DATE: / /20
PAGE NO.

Union - Find Algorithm

follows divide & conquer with disjoint sets.

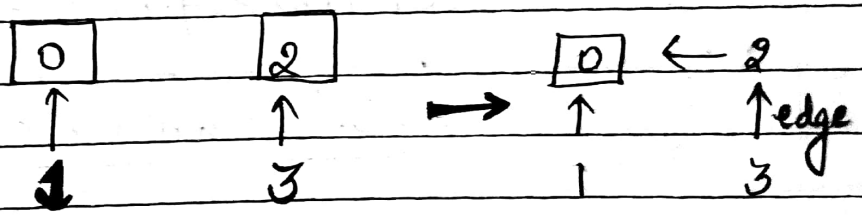
→ Union (combine)
→ Find

Example → People

0 1 2 3 4 5 6 7

make friends, all find

- mk = {0,1}
- mk = {2,3}
- aj = {0,2} → False
- mk = {0,2} - union
- aj = {0,2} → True - find



Hence, 0 = {0, 1, 2, 3}

Min - Max Algorithm (Minimum - Maximum)

LINEAR METHOD

eg →

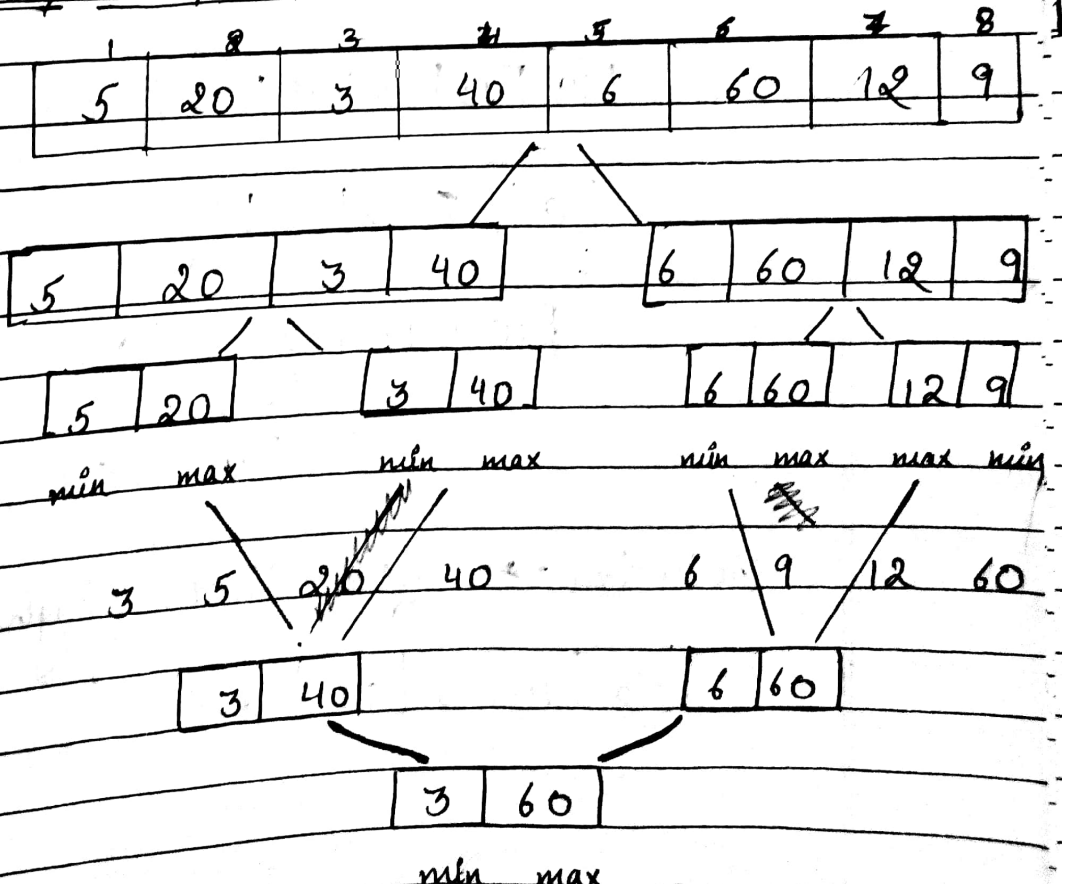
1	2	3	4	5	6
20	3	40	6	60	12

```
Min = num [ ]
Max = num [ ]
for i = 2 to n do
    if num [i] > max then
        max = num [i]
    else
        if num [i] < min then
            min = num [i]
return (max, min)
```

→ min = 3
max = 60

(2) DIVIDE AND CONQUER

eg. →



DATE : / /20
PAGE NO.

20-2-25

Sorting

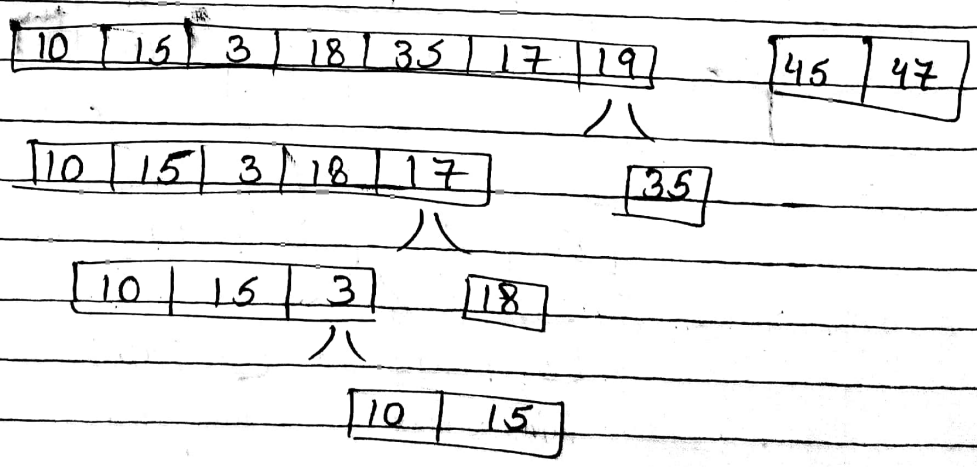
→ ## QUICK SORT

Quick sort Algo. is based on Divide & Conquer approach. It is based on the idea of choosing one element as a pivot element. Left side of the pivot element is less than the pivot & right side of the pivot element is greater than the pivot.

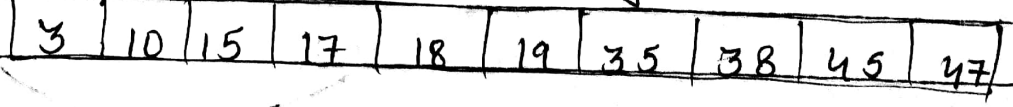
$O(n^2)$ - worst , $O(n \log n)$ - best

Eg. A = {10, 15, 3, 18, 45, 35, 17, 19, 47, 38}

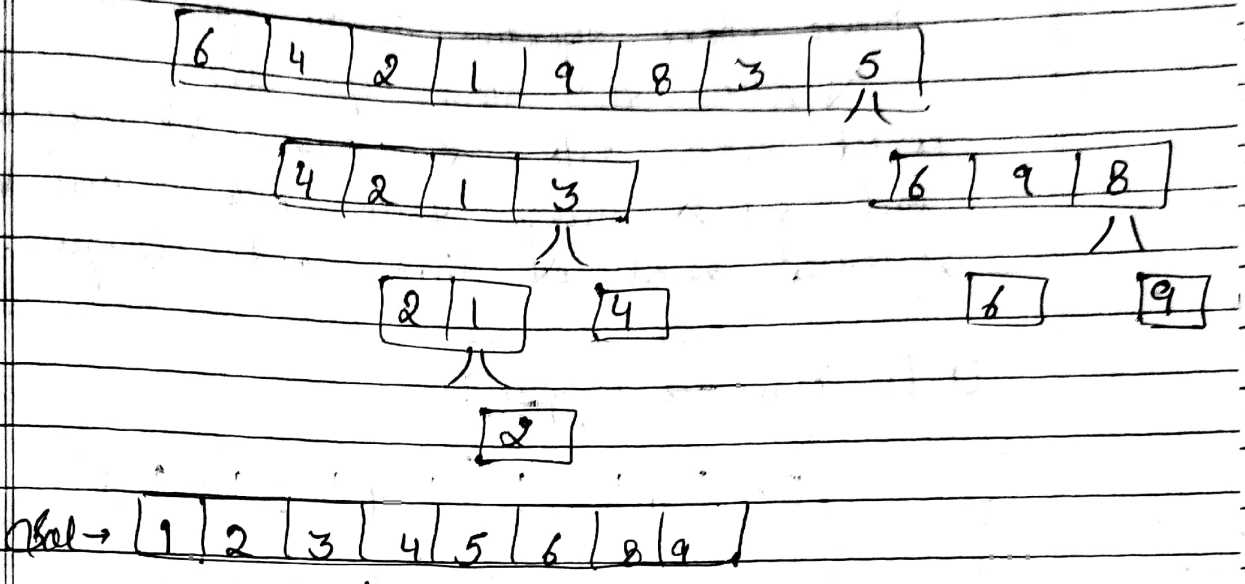
10, 15, 3, 18, 45, 35, 17, 19, 47, 38



Solution → left Pivot Right → 2 PR



Eg. A = {6, 4, 2, 1, 9, 8, 3, 5}



Algorithm

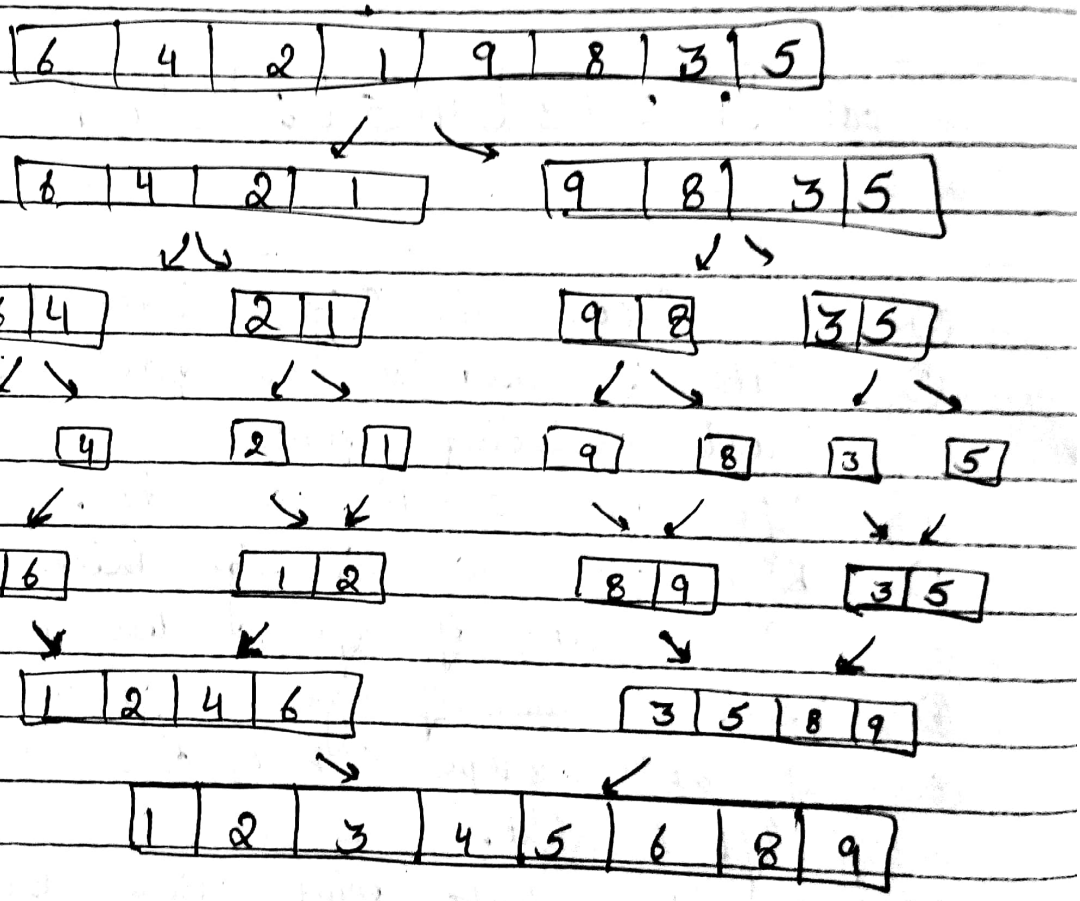
1. Choose the highest index value as pivot.
2. Take 2 variables to point left & right of the list excluding pivot.
3. left point to the low index.
4. Right point to the high index.
5. While value of left is less than pivot, move right.
6. While value of right is greater than pivot, move left.
7. If both steps 5 & 6 does not match, swap left & Right.
8. If left > Right point where they meet a new pivot.

→ MERGE SORT

Merge Sort Algorithm is an algorithm that splits the item of array to be sorted order into 2 groups, recursively sort each grp & merge them into final sorted sequence; based on divide and conquer.

O (n log n) → complexity

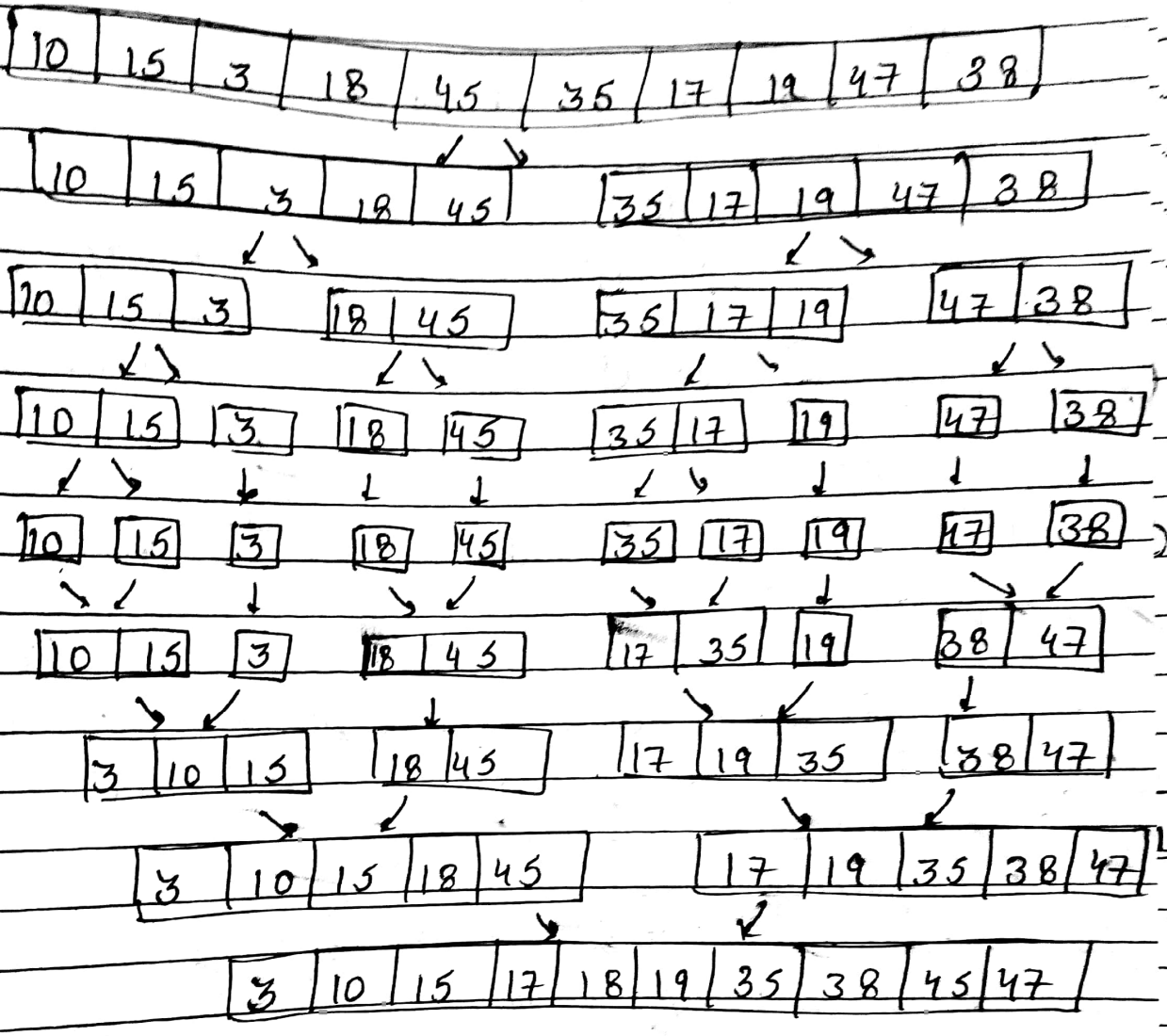
eg:



Algorithm

1. If it's only 1 element in the list, consider it already sorted, so we return
2. Divide the list recursively in 2 grps, until it can not more to be divided
3. Merge the smaller list into new list in sorted order.

eg.



21-2-25

Heap and Heap sort

types → Max. & Min.

Heap sort is an efficient sorting technique based on Heap data structure. It is nearly complete binary tree where parent node should be max. or min.

main operations in the procedure are:

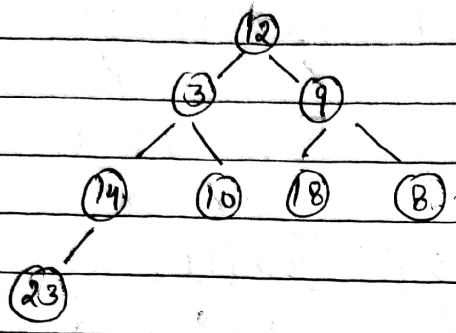
1. Build a heap H from the I/P data using heapify method.
2. Delete the root element and repeat until all the I/P elements are processed.

DATE : / /20
PAGE NO.

MAX HEAP

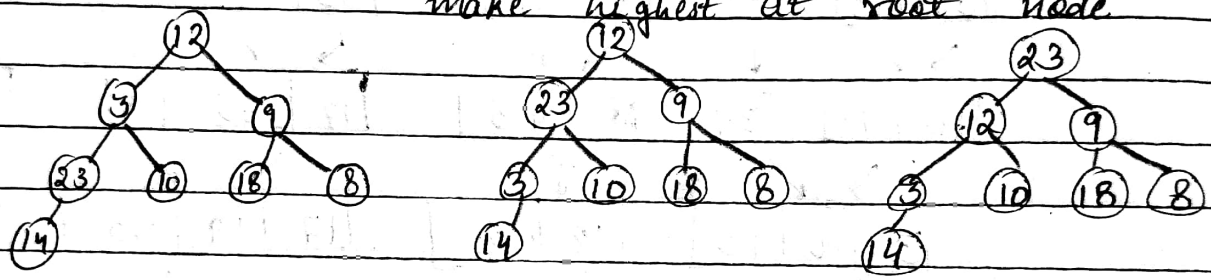
Eg.

12	3	9	14	10	18	8	23

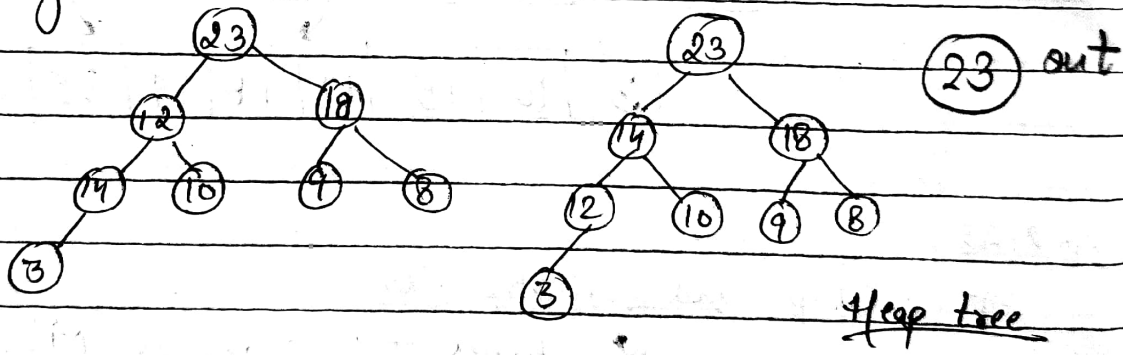
→ 

form a normal binary tree [has atmost 2 children order-wise]

→ make highest at root node



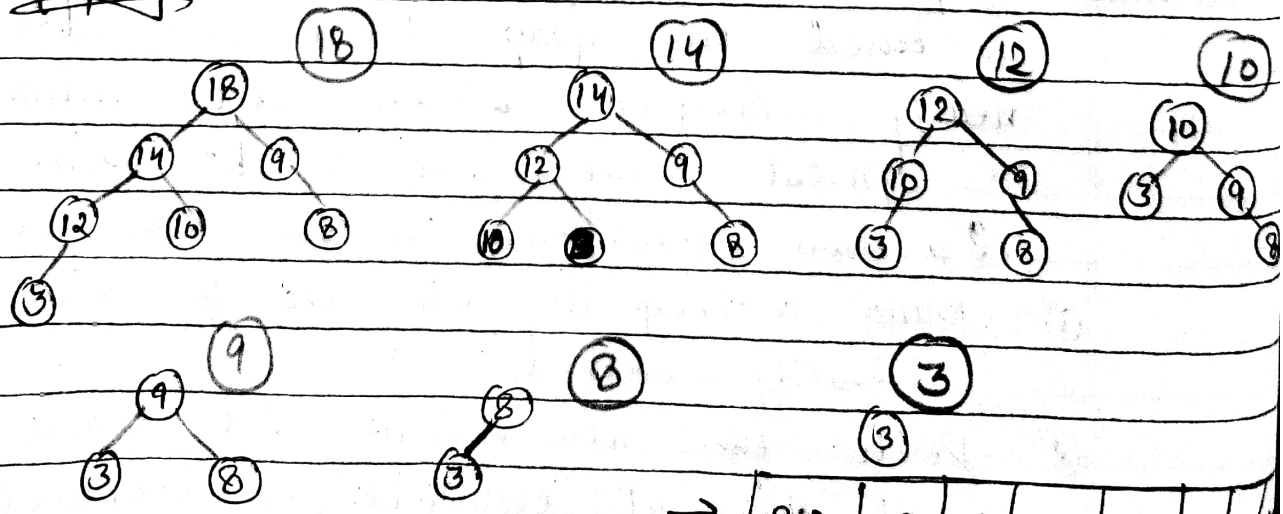
→ rearrange



23 out

Heap tree

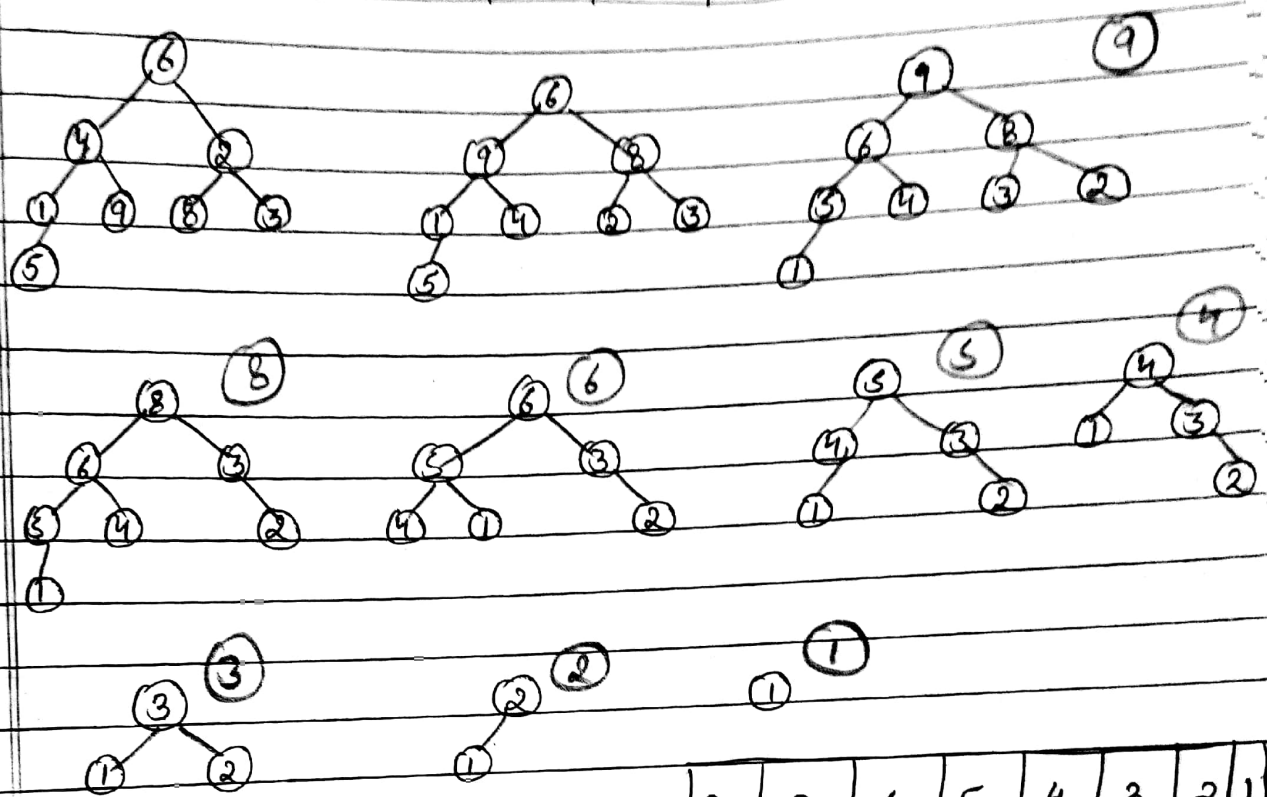
→ Heaping



→ | 23 | 18 | 14 | 12 | 10 | 9 | 8 |

2.

6	4	2	1	9	8	3	5

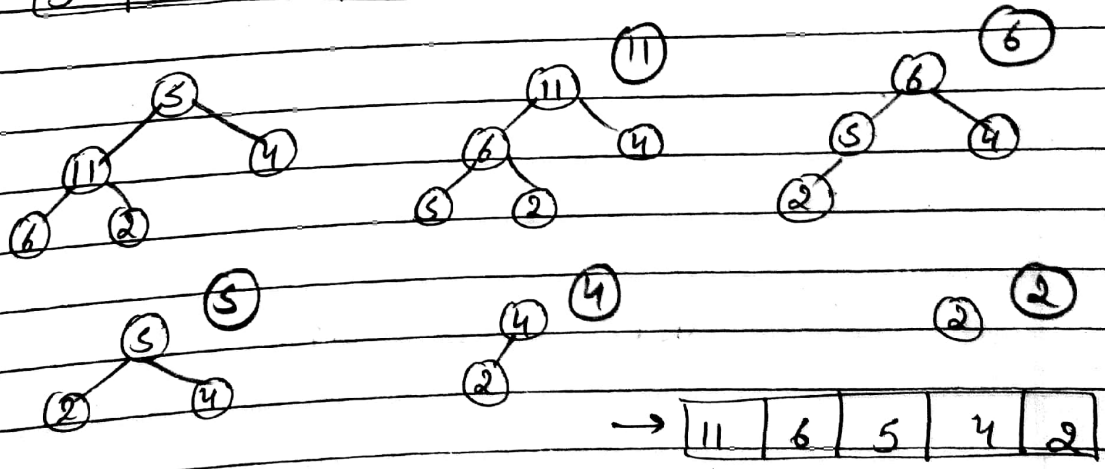


→

9	8	6	5	4	3	2	1

3.

5	11	4	6	2



→

11	6	5	4	2

Heapify method → Convert tree into heap data structure.

Combination of Insertion & Merge sort.

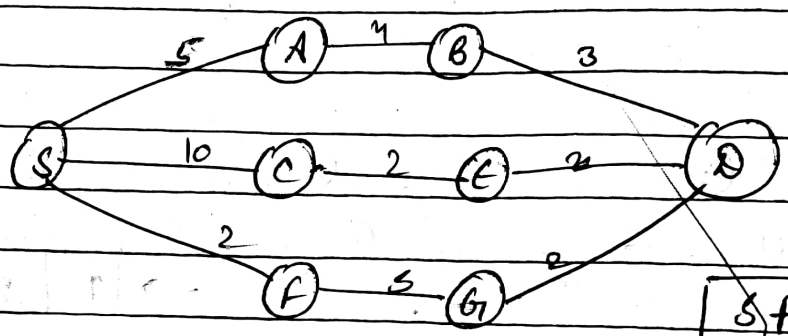
$O(n \log n)$ → complexity

March 8, 25

Greedy Algorithm

Min. cost, Min. risk, Max. profit

Greedy technique → It follows local optimal choice of each stages with intend of finding global optimum. It's totally based on optimal solution based on min. cost & profit will be max.



S → G → D = 9

Applications

1. Knapsack
2. Job sequencing
3. Minimum spanning tree
4. Optimal Merge Pattern
5. Huffman's Coding

Optimal Storage of Tapes using greedy technique

We have 'n' programs and we need to save all these programs in computer tape (tape is nothing but magnetic tape) that provides sequencing pattern, we can access these programs & we have to store all these programs in minimal retrieval time.

Eg.

Consider $n=3$, lengths are $(l_1, l_2, l_3) = (5, 10, 3)$. find the optimal storage order. Tape = 1

→

$P_1 = l_1 = 5$

$P_2 = l_2 = 10$

$P_3 = l_3 = 3$

Order	Retrieval time	total	
1 2 3	5 + (5+10) + (5+10+3)	38	
1 3 2	5 + (5+3) + (5+3+10)	31	
2 1 3	10 + (10+5) + (10+5+3)	43	
2 3 1	10 + (10+3) + (10+3+5)	41	
3 1 2	3 + (3+5) + (3+5+10)	29	✓optimal
3 2 1	3 + (3+10) + (3+10+5)	34	

Eg. → $n=10$,

l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}
10	20	45	70	1	3	7	54	23	67

tape = 3.

→ ascending order, 1 3 7 10 20 23 45 54 67 70

Tape	Order	Retrieval time	total
0	1, 10, 45, 70	1+(1+10)+(1+10+45)+(1+10+45+70)	194
1	3, 20, 54	3+(3+20)+(3+20+54)	103
2	7, 23, 67	7+(7+23)+(7+23+67)	134
			431

as we have 3 tapes, $\frac{431}{3} = 143.67$

mean retrival = 143.67

DATE : / /20
PAGE NO.

Eg → ~~n=3, l_1=10, l_2=20, l_3=45, l_4=70, l_5=1, tape=1~~

n=3, l_1=10, l_2=5, l_3=30, tape=1

→

Orders	Retrival time	total
123	10+(10+5)+(10+5+30)	70
132	10+(10+30)+(10+30+5)	95
231	5+(5+30)+(5+30+10)	85
213	5+(5+10)+(5+10+30)	65 ✓ optimal
321	30+(30+5)+(30+5+10)	110
312	30+(30+10)+(30+10+5)	115

4-3-25

Knapsack Algorithm

Given 'n' items where each item has some weight & profits associated with it and also given a bag capacity 'w' i.e. The task is to put the items into the bag such that sum of profits associated with them is max. possible.

Eg.

Object	Obj 1	Obj 2	Obj 3
Profit	25	24	15
Weight	18	15	10
P/w	1.3	1.6	1.5

capacity = 20 (kg)

- Greedy about profit (more profit) = $1 \times 25 + \frac{2}{15} \times 24$

$= 25 + \frac{48}{15}$

$= 28.2$

obj 1 + $\frac{2}{15}$ obj 2

DATE: / /20
PAGE NO.

- Greedy about ~~profit~~ weight (less weight) = $15 + \frac{10}{15} \times 24$

$= 15 + \frac{240}{15}$

$= 31$

obj 3 + $\frac{10}{15}$ obj 2

- Greedy about $\frac{\text{Profit}}{\text{weight}}$ (highest ratio) = $24 + \frac{5}{10} \times 15$

$= 24 + \frac{75}{10}$

$= 31.5$

obj 2 + $\frac{5}{10}$ obj 3

Hence, (ratio of profit by weight)'s profit is highest

Eg. Find the optimal solution for knapsack problem $n = 7$, $w = 15$.

obj	1	2	3	4	5	6	7
profit	10	5	15	7	6	18	3
weight	2	3	5	7	1	4	1
p/w	5	1.6	3	1	6	4.5	3

Greedy about profit = $18 + 15 + 10 + \frac{4}{7} \times 7$

$= 43 + 4$

$= 47$

(obj 6 + 3 + 1 + $\frac{4}{7}$ 4)

Greedy about weight = $6+3+10+5+18+\frac{4}{5}\times 15$

obj $(1+2+\frac{2}{5}(18)+5+6+7)$ = $24+18+12$

= 54

Greedy about both = $6+10+18+15+3+\frac{2}{3}\times 5$

obj $(1+2+\frac{2}{3}(18)+3+5+6+7)$ = $52+\frac{10}{3}$

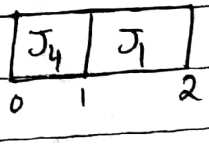
= 55.3

Hence, greedy about both weight & profit ratio gives max. possible profit.

10-3-25

Job Sequencing Problem

Input	1	2	3	4
Deadline	2	1	2	1
Profit	50	15	10	25



Profit = $50+25 = 75$

Definition → Given 3 array's I-d, deadline, profit where each job I is associated with I-d, deadline, profit. Each job take 1 unit of time to complete and only 1 job can be scheduled at a time. We'll ~~Deadline~~ earn the profit associated with the job only if it is completed by it's deadline.

The task is to find the max. profit that can be gained by completing the jobs & the count of job completed to earn max. profit.

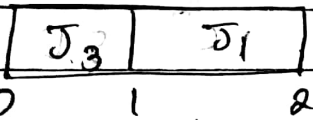
Algorithm

① Arrange all jobs in decreasing order of their profit.

② for each job do linear search to find particular slot in array of size m, where m = max. deadline, n = total jobs.

Eg.

Jobs	1	2	3	4	5
Deadline	2	1	2	1	1
Profit	100	19	27	25	15



Profit = 27 + 100 = 127

* Minimum spanning tree

acyclic, connected, all vertices visited, min. weighted, undirected, subgraph.

- Kruskal's → weight min. (ascending order) priority queue (parent - key)
- Prim's

DATE: / /20
PAGE NO.

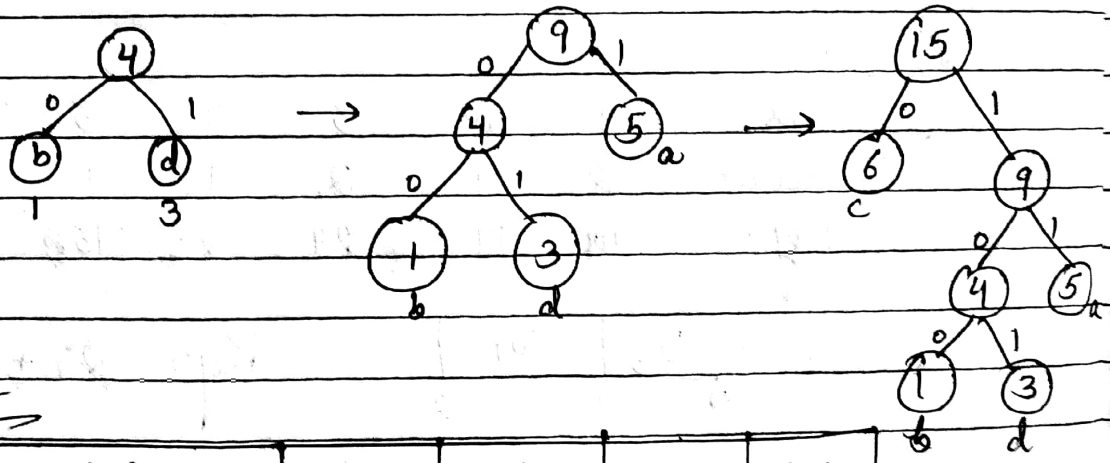
11-9-25
Thursday

Huffman Coding

It's a technique of compressing data to reduce its size, without losing any of the detail. It was developed by Davi Huffman. It's generally used to compress the data in which there are frequently occuring characters.

Eg. → Given frequencies

a	b	c	d
5	1	6	3



13-3

Result

traversing (0,1)	11	100	0	101
Values	a	b	c	d
frequency	5	1	6	3
bit frequency	2X5	3X1	1X6	3X3
	10	3	6	9

= 28 bits

Algorithm

① Calculate the frequency of each character in the string.

② Sort the characters in increasing order of the frequency. This is stored in priority queue.

(3) Make each unique character as a leaf node.
(4) Create empty node z assign the min. frequency to be left or second min. frequency to the right side. Set the value as sum of two min. frequency.
(5) Remove these 2 frequency from queue and add the sum of the list frequency
(6) Insert node z into the tree. Repeat the step 3-5 for all characters.
(7) For each leaf node assign to left node (0) zero and assign right to the (1) one.

13-3-25

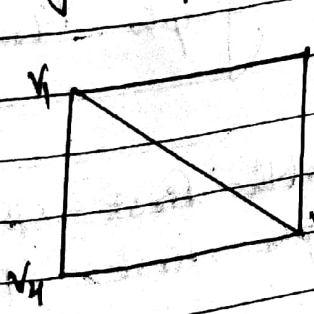
Graph Algorithm

These are the methods used to manipulate and analyze graphs, solving various range of problems like finding the shortest path, cycles detection.

2 types of Graph representation →
① Adjacency Matrix Representation
② Adjacency list Representation

① Adjacency Matrix Representation

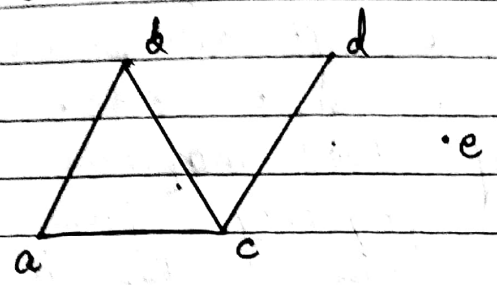
4 vertices, hence 4×4 matrix



	v_1	v_2	v_3	v_4
v_1	0	1	1	1
v_2	1	0	1	0
v_3	1	1	0	1
v_4	1	0	1	0

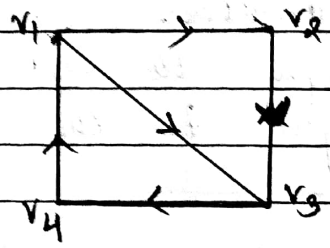
DATE: / 120
PAGE NO.

② Adjacency list matrix



Vertex	adjacency list
a	b, c
b	a, c
c	a, b, d
d	c
e	ϕ

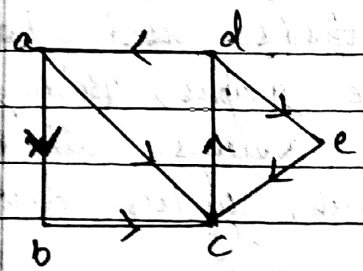
Ques→



Matrix

	v_1	v_2	v_3	v_4
v_1	0	1	1	0
v_2	0	0	1	0
v_3	0	0	0	1
v_4	1	0	0	0

Ques



List

Vertex	adjacency list
a	b, c
b	c
c	d
d	a, e
e	c

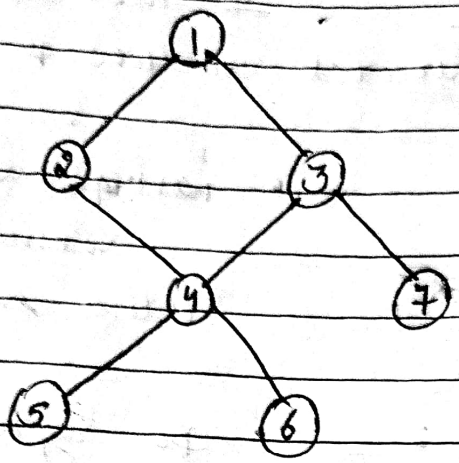
✱ Graph Traversal

Process of visiting & exploring a graph for processing is called graph traversal.

Two types →

① BFS (Breadth - First - Search)
② DFS (Depth - First - Search)

Ques.



BFS (in queue) (FIFO)

1	2	3	4	7	5	6

DFS

7
3
6
5
4
2
1

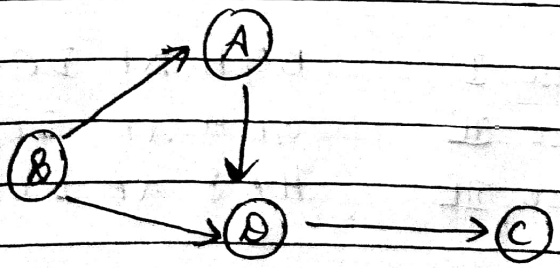
1	2	4	5	6	3	7

(In stack)
(LIFO)

17-3-25

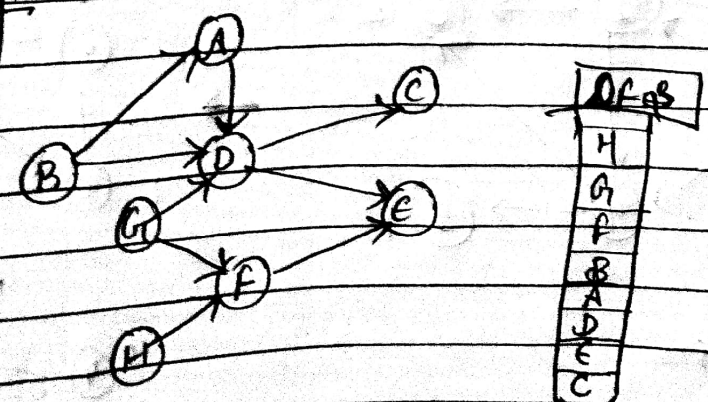
Topological Sort

It's linear ordering of graph vertices such that for every directed edge u, v from vertex u to vertex v. u comes before v in ordering, graph should be directed acyclic graph DAG. Every DAG should be have atleast one topological order.



topological sort,
→ BADC

eg.



DFS

H
G
F
B
A
D
E
C

I method

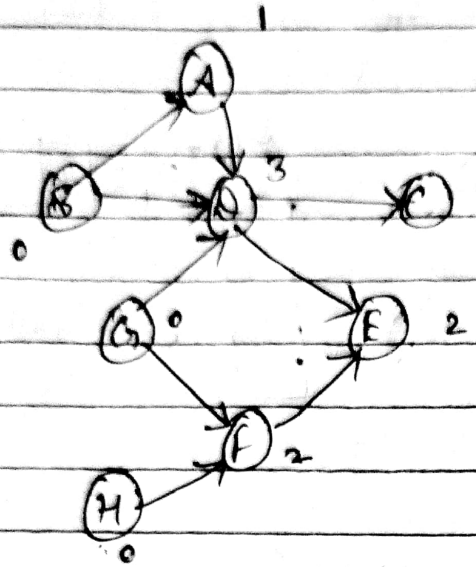
B → A → D → C → D → E
G → F
H

DATE : / /20
PAGE NO.

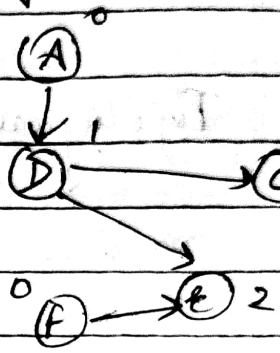
Linear Ordering

Degree of vertices → directed vertices as edges

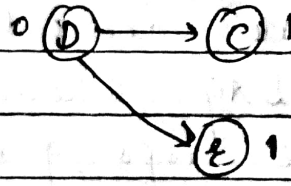
II method



- having degree 0, remove those vertices & connected edges
- so graph becomes

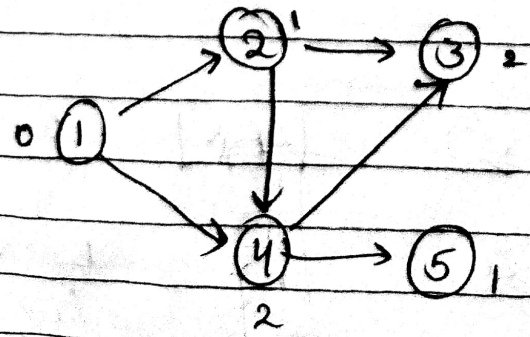


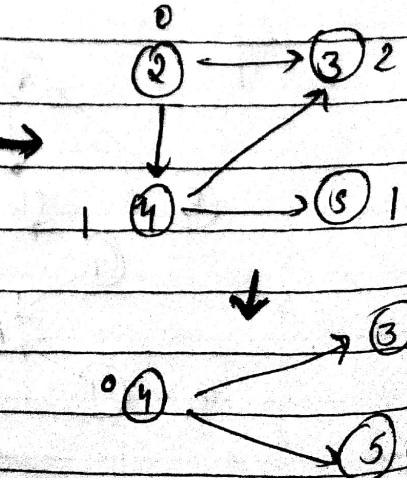
- graph



hence,

Case I BGH AF DCE
Case II GBH AF DCE
Case III HGB AF DCE

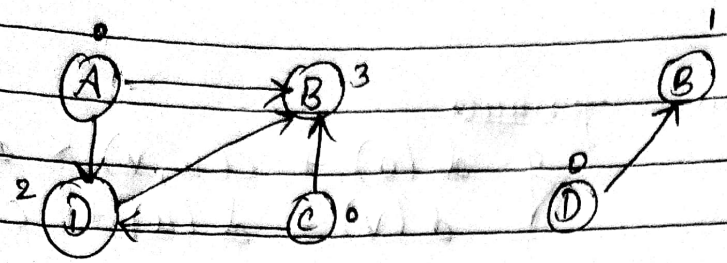




1, 2, 4, 3, 5

DATE: / /20
PAGE NO.

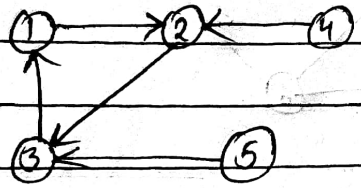
Ques-



Case I A C D B

Case II C A D B

Ques



a cycle.

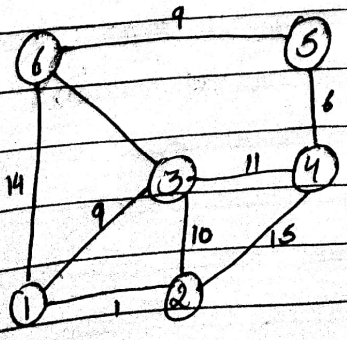
Single Source Shortest paths

Dijkastra's Algorithm

find the shortest path from one vertices to other vertices.

- it does so by repeatedly selecting nearest unvisited vertex and calculate distance to all the unvisited neighbouring vertices.
- We can use both directed and undirected graph in this algorithm.

Ques

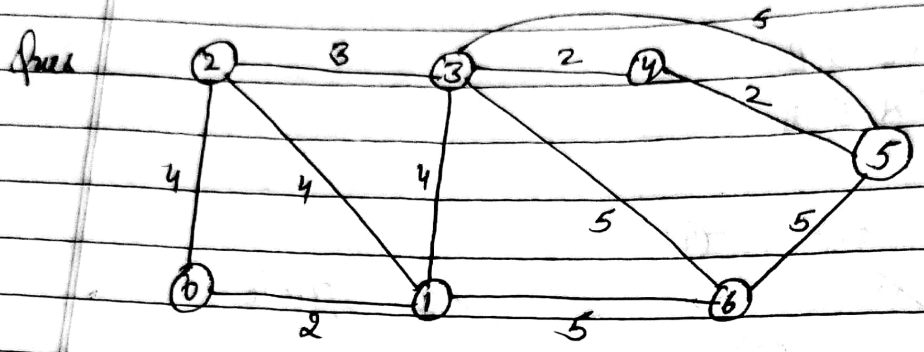


source	destination					
1	1	2	3	4	5	6
1,2,3,6	0	1	9	∞	∞	11
1,2,3,6,4	0	1	9	16	∞	11
1,2,3,6,4,5	0	1	9	16	20	11

Relaxation formula

If $d(0) + c(u,v) < d(v)$

$d(v) = d(u) + c(u,v)$



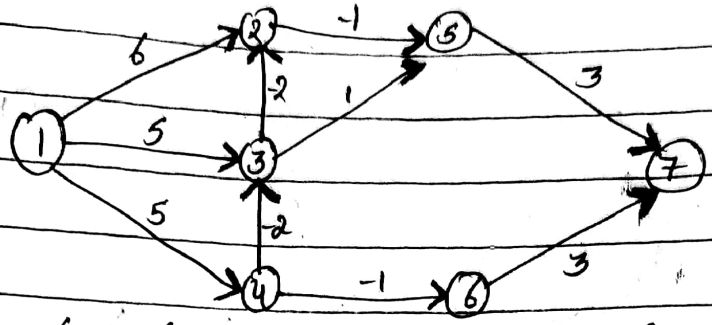
Source	Destination					
0	1	2	3	4	5	6
0,1,2	(2)	(4)	∞	∞	∞	∞
0,1,2,3,6	(2)	(4)	(6)	∞	∞	(7)
0,1,2,3,6,4	(2)	(4)	(6)	(8)	∞	(7)
0,1,2,3,6,4,5	(2)	(4)	(6)	(8)	(10)	(7)

24-3-25

Bellman Ford's Algorithm

It works when there is negative weight edge. It also detects negative weight cycle. Dynamic programming approach is taken to implement this algorithm.

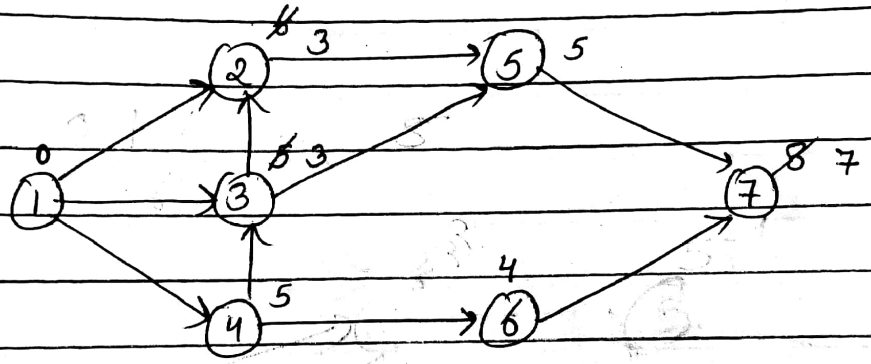
Ques



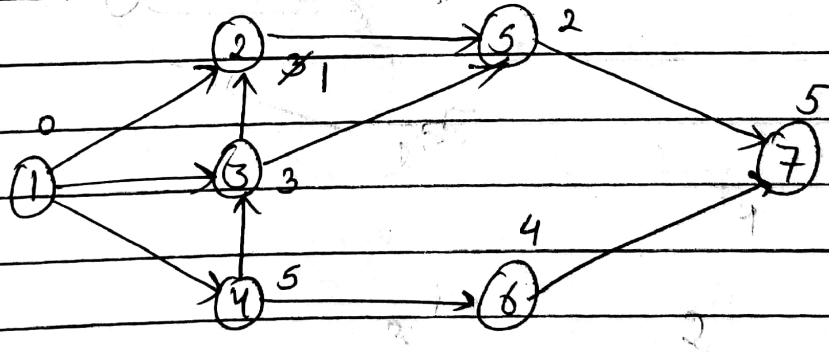
(1,2) (1,3) (1,4) (2,5) (3,2) (3,5) (4,3) (4,6) (5,7) (6,7)

v-1 = 7-1 = 6 times (solution)

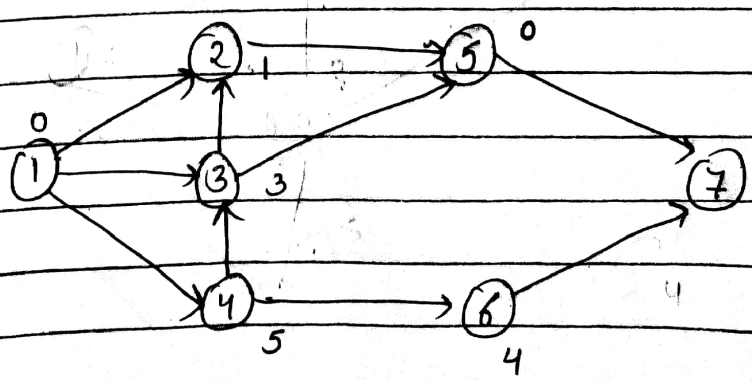
time- (1)



time- (2)



time- (3)



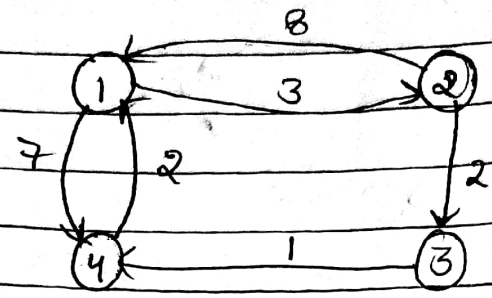
Solved.

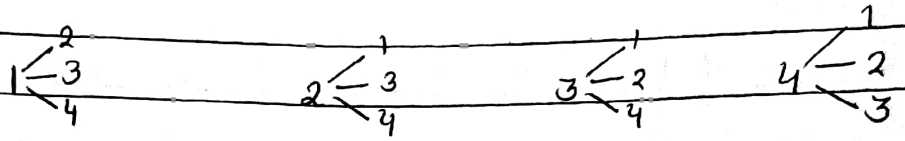
DATE : / /20
PAGE NO.

25-3-25

Warshall Algorithm

(1)





A^0 =

	1	2	3	4
1	0	3	∞	7
2	8	0	2	∞
3	∞	∞	0	1
4	2	∞	∞	0

A^1 =

	1	2	3	4
1	0	3	∞	7
2	8	0	2	15
3	∞	∞	0	1
4	2	5	∞	0

I row & column same, (2,3) → (2,1) + (1,3)

2 8 + ∞

2 < 8

hence, 2

DATE : / /20
PAGE NO.

A^2 =

	1	2	3	4
1	0	3	5	7
2	8	0	2	15
3	∞	∞	0	1
4	2	5	∞	0

A^3 =

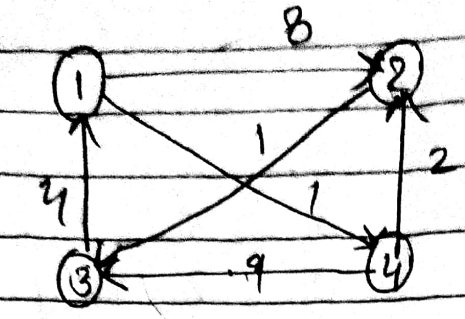
	1	2	3	4
1	0	3	5	7
2	8	0	2	3
3	∞	∞	0	1
4	2	5	∞	0

A^4 =

	1	2	3	4
1	0	3	5	7
2	8	0	2	3
3	3	∞	0	1
4	2	5	∞	0

DATE : / /20
PAGE NO.

(2)



$A^0 =$

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

$A^1 =$

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	9	0

$A^2 =$

	1	2	3	4
1	0	8	9	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	3	0

$A^3 =$

	1	2	3	4
1	0	8	9	1
2	5	0	1	∞
3	4	12	0	5
4	13	2	3	0

DATE : / /20
PAGE NO.

$$A^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 9 & 1 \\ 5 & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ 13 & 2 & 3 & 0 \end{bmatrix} \end{array}$$

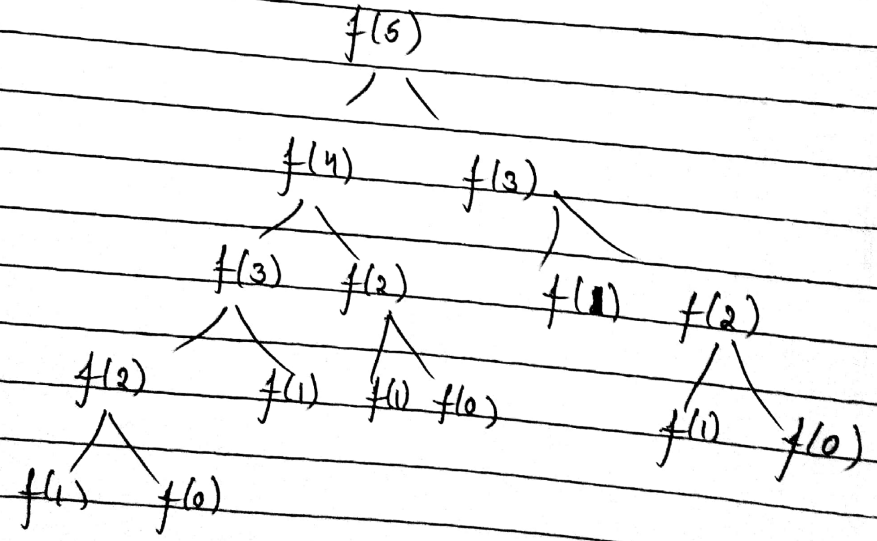
DATE: / /20
PAGE NO.

Unit - III

Dynamic Programming

Fibonacci Series

```
f(n)
{ if (n==0)
    return 0;
  if (n==1)
    return 1;
  if (n>1)
    return (f(n-1) + f(n-2));
}
```



f(0)	f(1)	f(2)	f(3)	f(4)	f(5)
0	1	1	2	3	5

In the dynamic programming, a problem can be divided into subproblems and subproblems are dependent on each other. Solutions are stored for future use & no need to calculate again & again.

* Difference

Divide & Conquer	Dynamic Programming
① Solve the problem by dividing the problem into subproblems.	Solve the problem by dividing the problem into subproblems.
② Sub-problems are not dependent on each other.	Sub-problems are dependent on each other.
③ Eg. → Merge Sort & Quick Sort.	Eg. → Matrix chain multiplication & 0/1 Knapsack.
④ Does not store "sol" of subproblems.	Stores the "sol" of subproblems