Name:- Kedar Lele. D.S. Paper NOV. 16.
Roll no:- 28.

DATE:

Q.1A Write an algorithm for:-
(1) Append two lists together.
(2) Count number of nodes in the list.

→ (1) Append two lists together:-
(Concatinate S1, S2 & S3).

a) Initialise TEMP = NULL, PTR = NULL, S3 = NULL

b) While (S1 is not equal to zero):
i) Create new node Temp.
ii) Temp → Data = S1 → data
iii) Temp = Next = NULL.
iv) S1 = S1 → Next.
v) If (*S3 equal to NULL)
*S3 = NULL ?
vi) Else
PTR → Next = Temp.
vii) PTR = Temp.

c) While (S2 is not equal to NULL).
i) Create NewNode Temp.
ii) Temp → Data = S2 → Data.
iii) Temp = Next = NULL
iv) S2 = S2 → Next.
v) If (*S3 equal to NULL)
*S3 = temp.
vi) Else
PTR → Next = Temp
vii) PTR = Temp.
d) ptR → Temp.
e) Exit.

DATE:

(2) Count number of nodes in the list.

i) Initialize N = 0
ii) PTR = START.
iii) While (PTR not equal to NULL).
    N = N+1
    PTR = PTR → next.
iv) Exit.

DATE:

Q.1 b Define heap :-

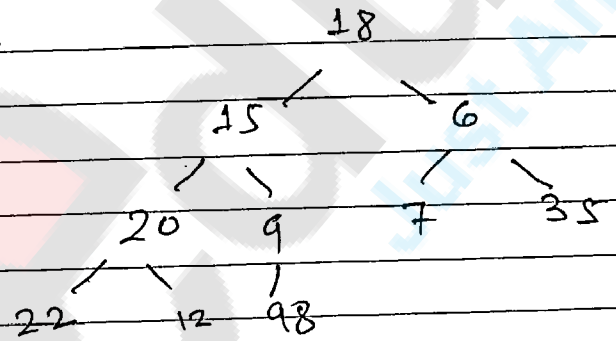
→ Heap is a special case of balanced binary tree data structure where the root node key is compared with children & arranged accordingly. If α has child node β then key(α) ≥ key(β).

As value of parent is greater than that of child this property generates max heap. Based on this heap can be of two types.

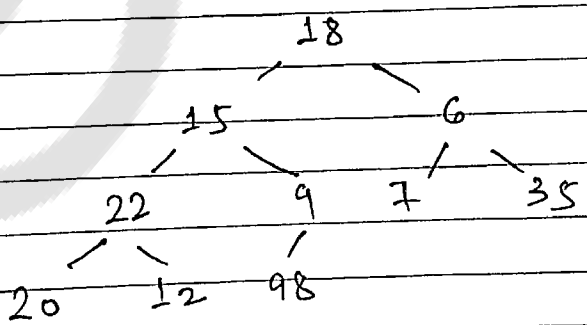
① min heap :- value of root is less than or equal to child
② max heap :- value of root is ≥ to children.

e.g :- 18, 15, 6, 20, 9, 7, 35, 22, 12, 98.

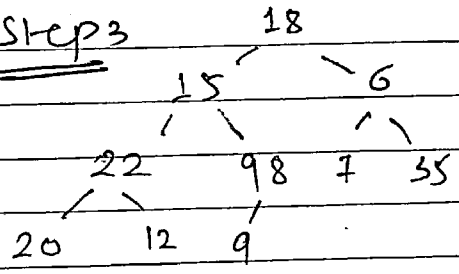
Step I



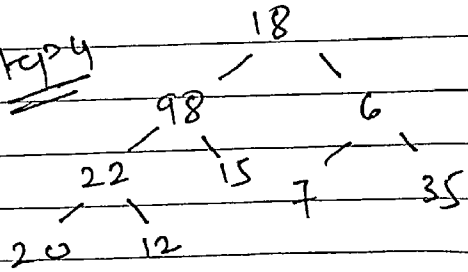
Step 2



Step 3

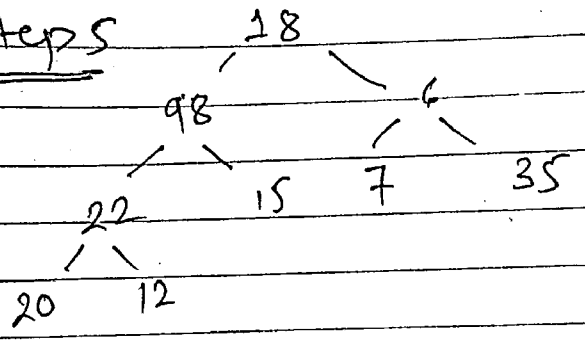


Step 4

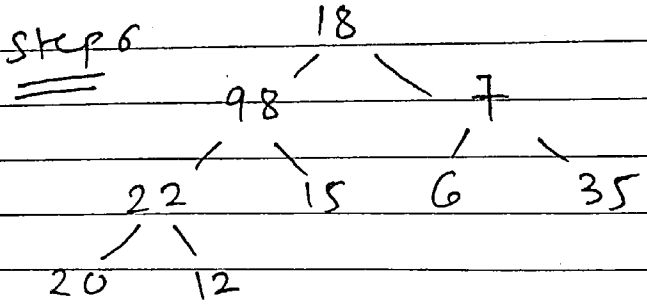


DATE:

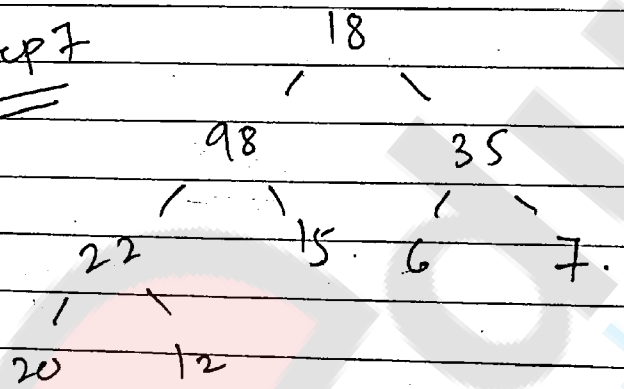
Step 5



Step 6



Step 7



DATE:

Q.2A Binary threaded tree :-

→ A binary tree is threaded by making all right child pointers that would normally be null point to inorder successor of the node (if it exists), and all left child pointers that would normally be null point to inorder predecessor of the node.

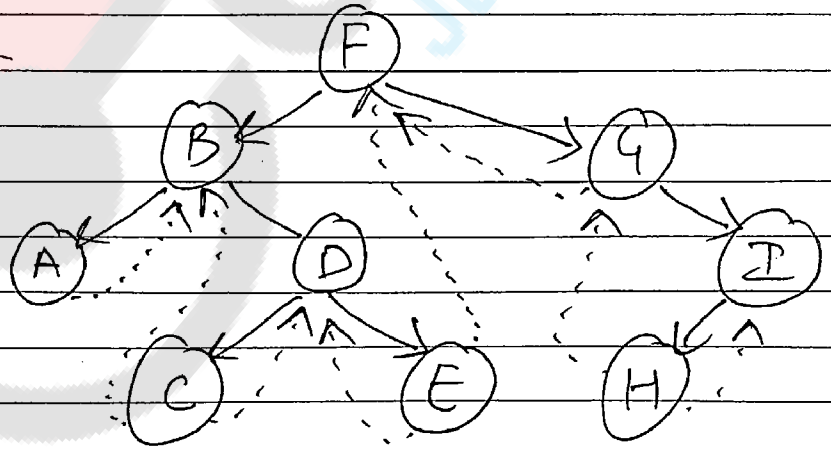
→ Following are the threaded binary tree :-

1. Single threaded.
2. Double threaded.

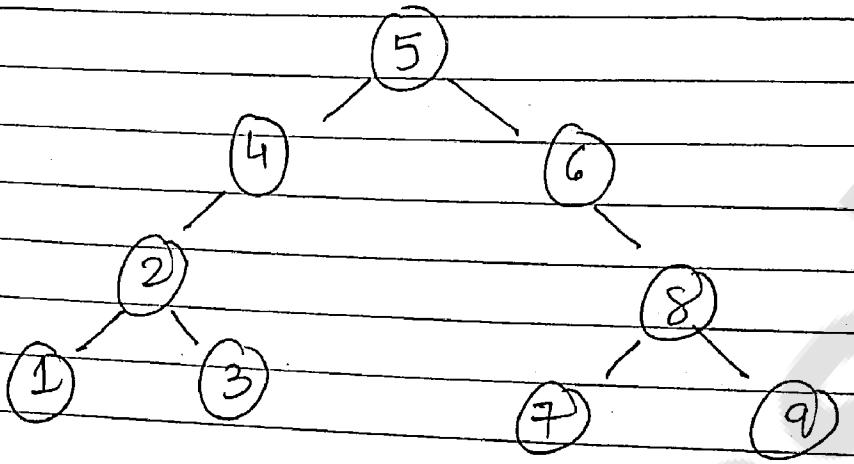
→ In single threaded each node is threaded towards either the in-order predecessor or successor.

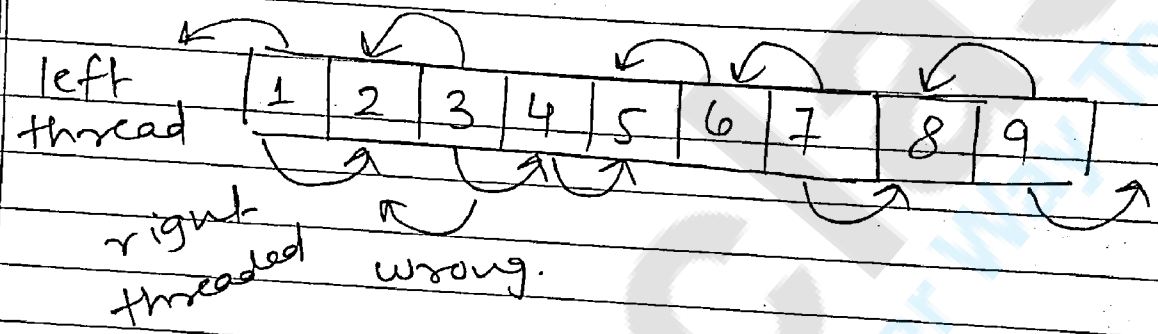
→ In double threaded each node is threaded towards both in-order predecessor & successor.

→ e.g :-



DATE:





DATE:

Q.2b Algorithm for Selection Sort.

→ Pass 1:- Find the location of smallest in the list of N elements.
A[1], A[2] ... A[N] and then interchange A[LOC] and [1]. Then A[1] is sorted.

Pass 2:- Find location of smallest in the list of N-1 elements.
A[2], A[3] ... A[N] interchange A[LOC] with A[2].

Pass 3:- The location LOC of smallest in sublist of N-2 elements A[3], A[4] ... A[N] & then interchange A[LOC] & A[3].

Pass N-1 Find the location LOC of smallest number from a list A[N-1];
A[N] iand interchange A[LOC] & A[N-1].
Thus A is sorted after N-1 passes.

⇒ Given array :- 18, 15, 6, 20, 9, 7, 35, 22, 12, 98

Pass 1:- 6, 15, 18, 20, 9, 7, 35, 22, 12, 98.

Pass 2:- 6, 7, 18, 20, 9, 15, 35, 22, 12, 98.

Pass 3:- 6, 7, 9, 20, 18, 15, 35, 22, 12, 98.

Pass 4:- 6, 7, 9, 12, 18, 15, 35, 22, 20, 98.

Pass 5:- 6, 7, 9, 12, 15, 18, 35, 22, 20, 98.

Pass BG:- 6, 7, 9, 12, 15, 18, 20, 22, 35, 98

Pass 7:- 6, 7, 9, 12, 15, 18, 20, 22, 35, 98.

DATE:

Q.3A Define stack:- A stack is linear structure or ordered collection of data in which items may be insuted or deleted, from one end called top of stack. Stack operates in LIFO manner.

→ Algorithm for push operation:-

1. Allocate new node.
2. Store data in new mode.
3. Make current node second node.
4. Make new node at top.
5. Increament clockcount.
   and Pushstack.

→ Pop operation ~~for~~ p algorithm:-

1. if (stack empty).
   1. Set success to false.
2. else.
   1. set data to data in top node.
   2. make second node the top node.
   3. Decreament stack count.
   4. Set success to ~~fase~~ true.
3. endif.
4. Return success

end popstack.

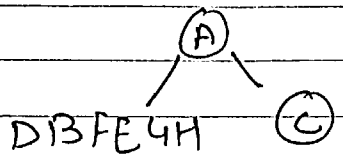
DATE:

Q.3 b Reconstruct the binary tree.

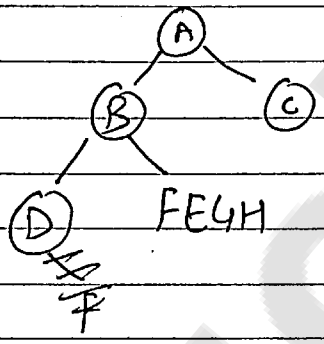
Inorder :- DBFEGHAC.

Preorder :- ABDEFGHC.

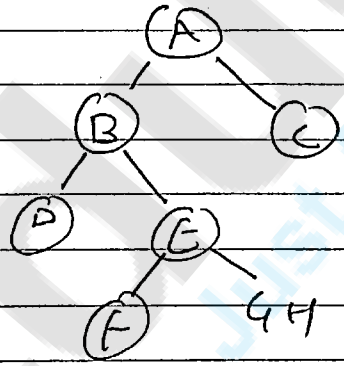
→ Step I



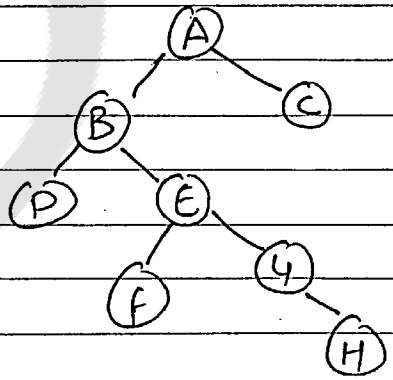
→ step II.



→ Step III



→ Step IV.



DATE:

- Define binary tree:-

→ A Tree in which number of nodes can have more than two subtrees i.e. A node can have 0, 1, or 2 sub trees.

→ The subtrees are called left subtree & right subtrees.

→ Height of binary tree can be represented mathematically

$$H_{max} = N.$$

$$H_{min} = [\log_2 N] + 1.$$

Where N is number of nodes.

DATE:

Q.4A Define graph. Explain structures which are used to store graphs.

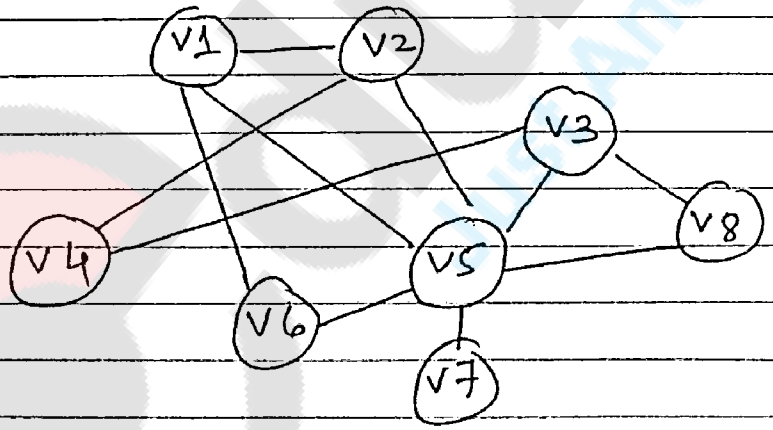
- **Graph:-** A graph is a collection of vertices & edges, G = (V, E) where V is set of vertices & E is set of edges. An edges is defined as pair of vertices, which are adjacent to each other.

- **Graph Storage Structures:-**

1) **Adjacency Matrix:-** We can use adjacency matrix i.e. a matrix whose rows & colums both represent vertices of graph.

→ In such matrix when i^{th} row j^{th} column element is one we say there is an edge between ith & jth vertex.

→ e.g:-



Adjacency graph:-

	V1	V2	V3	V4	V5	V6	V7	V8
V1	0	1	0	0	1	1	0	0
V2	1	0	0	1	1	0	0	0
V3	0	0	0	1	1	0	0	1
V4	0	1	1	0	0	6	0	0
V5	1	1	1	0	0	1	1	0
V6	1	0	0	0	1	0	0	1
V7	0	0	0	0	1	0	0	0
V8	0	0	1	0	1	0	0	0

DATE:

Adjacency list:-

→ The order representation is to prepare the adjacency list for each vertex. We will also see the adjacency relation expressed in linked list.

→ Adjacency list will be:-

V1 → V2 → V5 → V6
V2 → V1 → V4 → V5.
V3 → V4 → V5 → V8.
V4 → V2 → V3
V5 → V1 → V2 → V3 → V6 → V7 → V8.
V6 → V1 → V5.
V7 → V5
V8 → V3 → V5.

DATE:

Q.4 b Use binary search algorithm. trace element 13.

Array :- 8, 13, 26, 35, 65, 108, 139

Step
Pass 1 :- count number of elements :- 7.
Find middle ~~elet~~ element.
$\therefore (1+7)/2 = 4^{th}$ term.

$\therefore$ Middle element is ~~4~~. 35
Begining is 8
Last element is 139.

we want to search element 13.

Step2 Compare 13 with middle value. i.e. 35.
$\therefore$ $35 > 13$
$\therefore$ search on the ~~right~~ left side of middle value.

Step3 Find middle value of array ~~till~~ from 8 to 35
$\therefore$ ~~1 to~~ $4/2 = 2^{nd}$ position. i.e. 13.

Step4 Compare middle value with element we want to find out.

Step5 13 is the middle value & we are searching for 13.

Step6 stop.

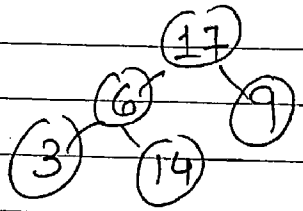
DATE:

Q.5 A What is binary search tree? Give algorithms
1) Delete a node from BST.
2) Find largest node in BST.

→ Binary search tree:-

Binary search tree is a tree with following property
(1) All elements in left subtree are less than root.
(2) All elements in right subtree are greater than root.
(3) Each subtree is also a binary search tree.

e.g.:-



→ Algorithm for deleting node from BST.

(1) If root is NULL.
a) Print tree is empty.
b) Exit.
(2) Set current root node parent to NULL.
(3) While (current not equal to NULL) & (current → data not equal to n).
a) Update parent = current.
b) if (current → data > n).
i) current = current → left.
c) Else.
i) current = current → right.
d) check the value of current. If current is NULL Print an error msg.

DATE:

4) If current nodes has both children then replace the data feild of current node with its inorder successor node 'succ.'.

5) current = succ.

6) If the current node has one child only then set the parent's link appropiatly.

7) free (current).

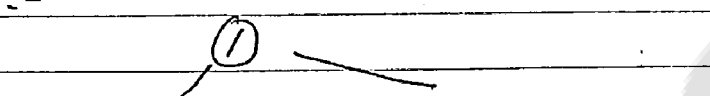
8) exit.

DATE:

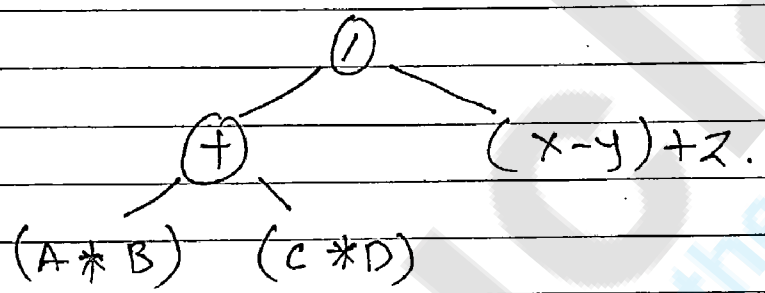
Q.6A Define expression tree. Consider following infix expression.

(A * B) + (C * D) / (x-y)+z.

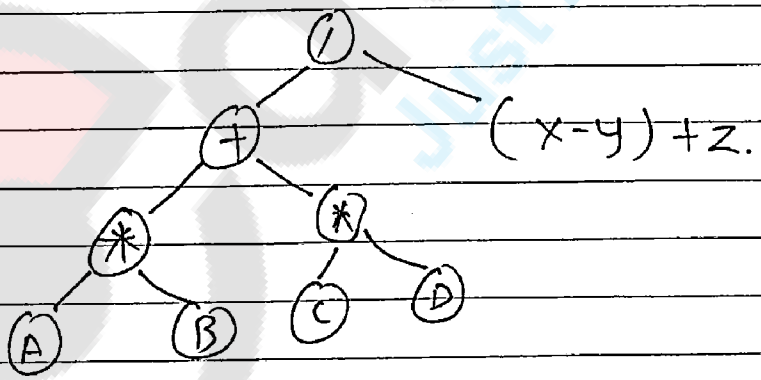
- visit root :-



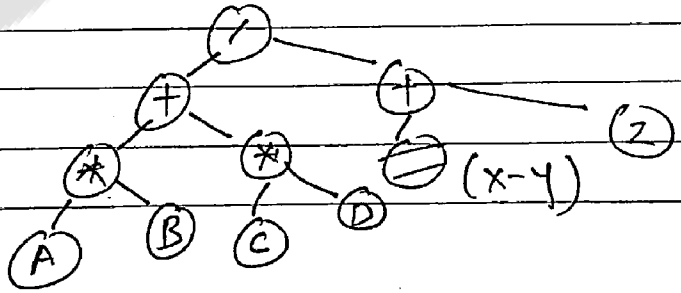
Visit root at (A * B) + (C*D).



Visit roots at (A*B)+(C*D)

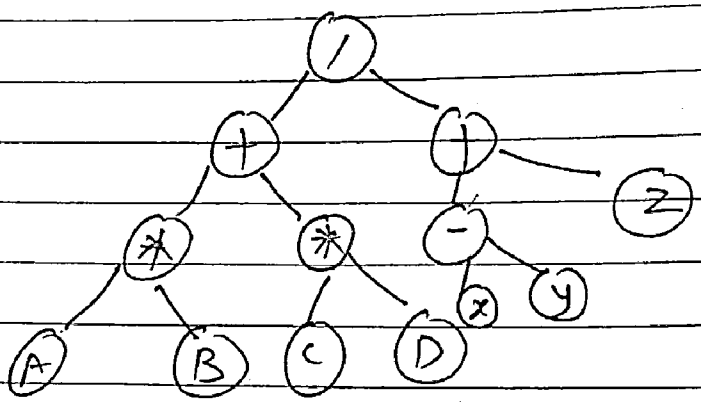


visit root at (x-y)+z.



DATE:

Visit root at (X-Y)



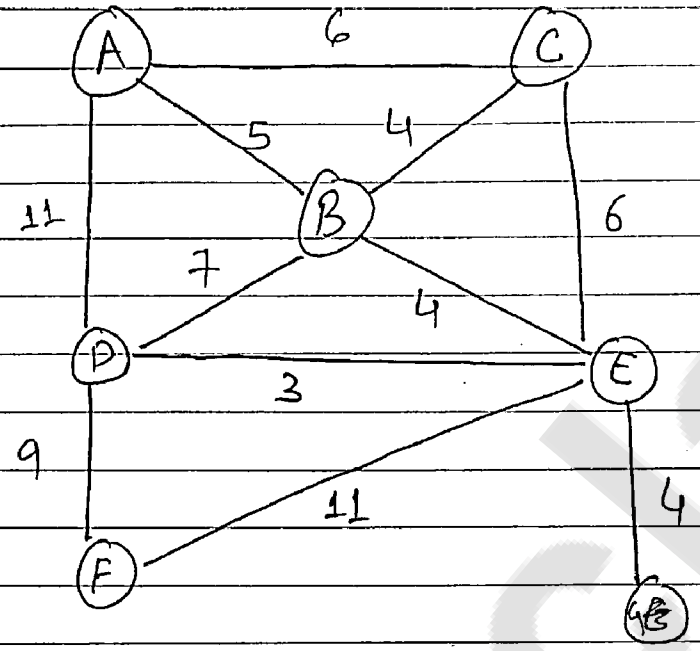
Pre-order :- / + * A B * C D + - x y z.

Post order :- A B * C D * + x y - z + /

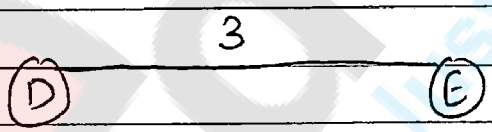
* Expression tree :-

DATE:

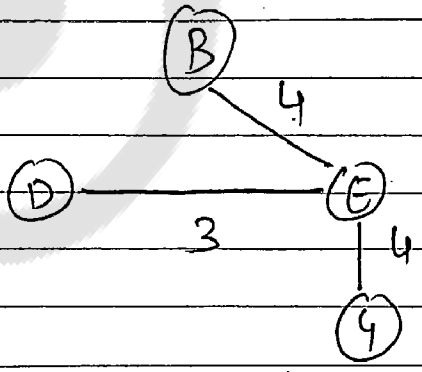
Q.6b Prim's algorithm:-



→ Visit minimum vertices:- D, E.

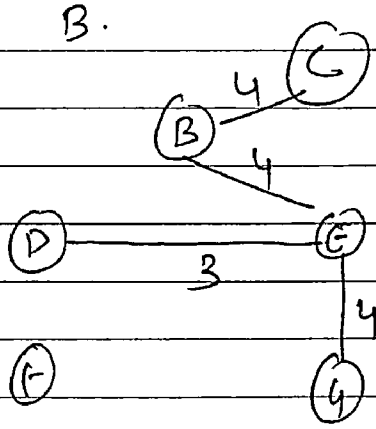


→ Adjacent vertices of E is 4.

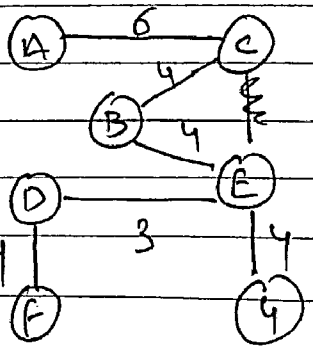


→

→ Adjacent vertices of B.



→ Find adjacent vertices of C.



DATE:

→ ~~Find adjacent vertices A.~~

(A)

Number of vertices are 7.
∴ n = 7.

∴ No. of edges must be n-1
i.e. 7-1 = 6.

The weight of minimal spanning tree is
= 30

Date: / /

Q7 A Complexity of an algorithm & Big-O notation.

→ Complexity of an g/o algorithm has two major aspects.
1) Time complexity.
2) Space complexity.

1) Time complexity :- Time complexity of an algorithm
→ is the amount of computer time needed to execute the program & get desired result.

2) Space complexity :-

→ The space complexity of an algorithm is the amount of computer memory needed to execute the program & get desired result.

* Mathematical notations for demonstration of running time of algorithm.

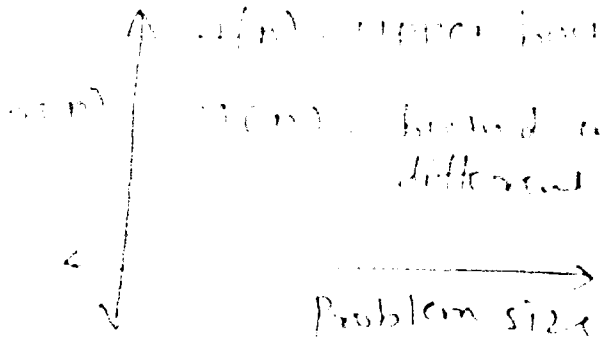
→ For practical problem it is difficult to count number of operations / instructions for worst / avg / best case running time.

→ Hence there is need for mathematical notations to determine Worst-case, Average case & best case running times.

Big O notation:

The time required to run the worst case using given algorithm is determined using O notation.

O( ) is defined as the rate at which the time of execution grows for the size of problem / size.



For non negative functions $T(n)$ & $f(n)$, the function $T(n) = O(f(n))$ if there are positive constants c & n_0 such that $T(n) \leq c * f(n)$ for all n, $n \geq n_0$.

DATE:

Q.7 B-Tree :-

→ A B-Tree is an m-way tree with following properties.

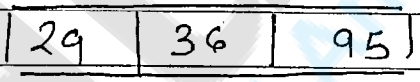
1. The root is either a leaf or it has 2 ... m subtrees.
2. The internal nodes have at least m/2 subtree subtree or m subtree.
3. All the leaf are at the same level i.e. it is perfectly balanced.
4. The leaf node has at least (m/2) - 1 or m-1 entries.

→ In otherwords B-tree is a m-way tree having potential to reduce ~~st~~ height of tree.

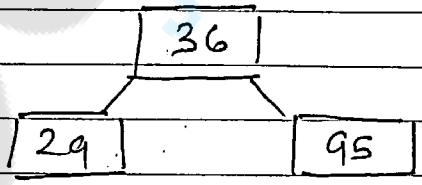
- Order 3 B-tree :-

95, 36, 29, 88, 46, 2, 19, 32, 75, 49, 55, 62, 5.

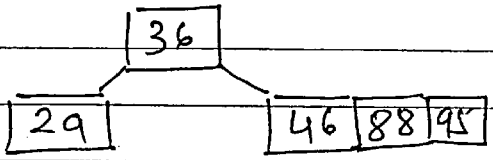
Step I Insert 95, 36, 29.



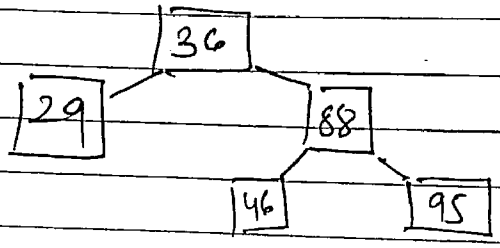
Splitting at 36.



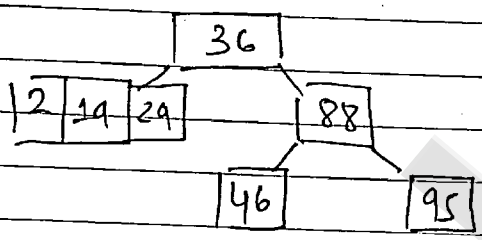
Inserting :- 88, 46.



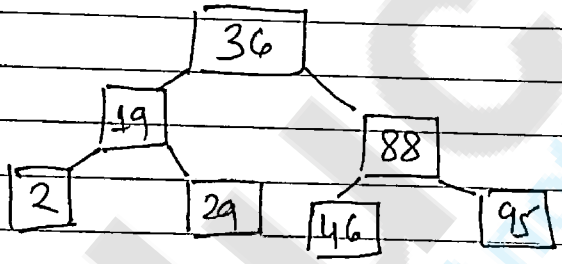
Spliting at 88.



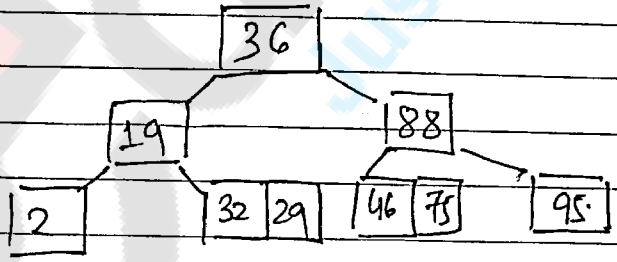
Inseting 2, 19.



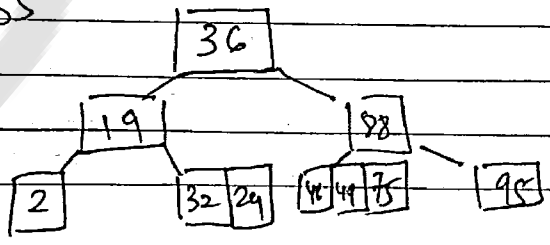
Splitting at 19.



Insert 32, 75.

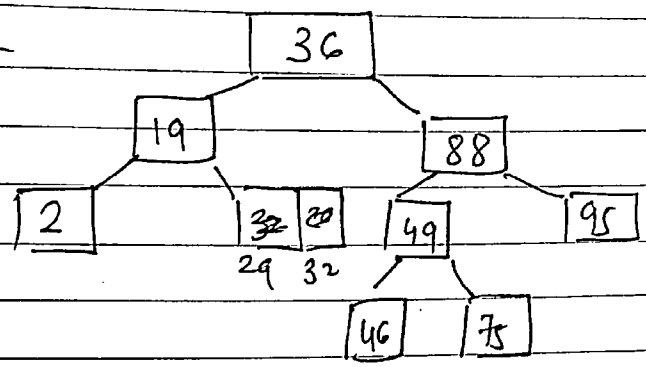


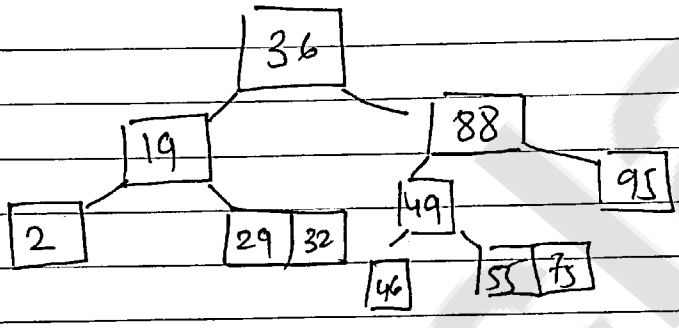
Insert 49, 55



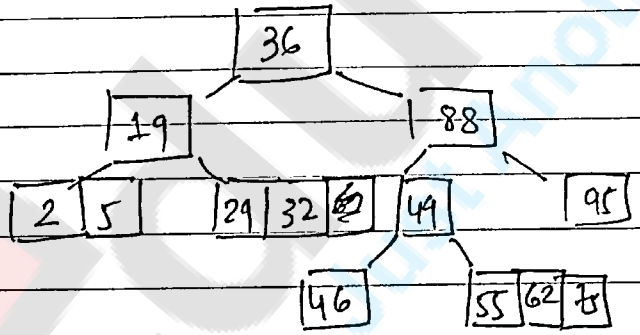
DATE:

Splitting:-





Inserting:- 62, 5.



Splitting:-