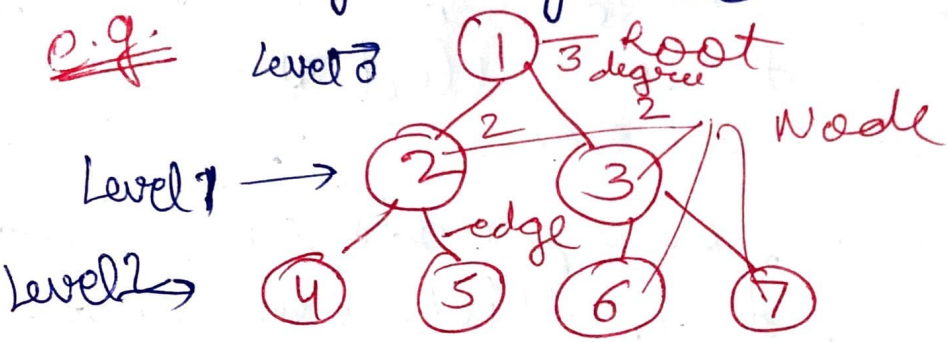
Unit-2

Trees: → A tree is a non linear data structure in which items are arranged in a sorted sequences.

→ It's a hierarchical data structure which stores the information naturally in the form of hierarchy style.

e.g.



Tree terminology :-

1. Root
2. Node
3. Degree of a Node
4. Degree of a tree (Max degree of one of it's node)
5. Terminal Node (Node having zero degree)
6. Non-terminal Node (Not having zero degree)
7. Siblings
8. Level
9. Edge (which connects two nodes)
10. Path 1 to 7 path — 1→3→7 (1, 3)(3, 7)
11. Forest (having many trees)

Properties of Trees :-

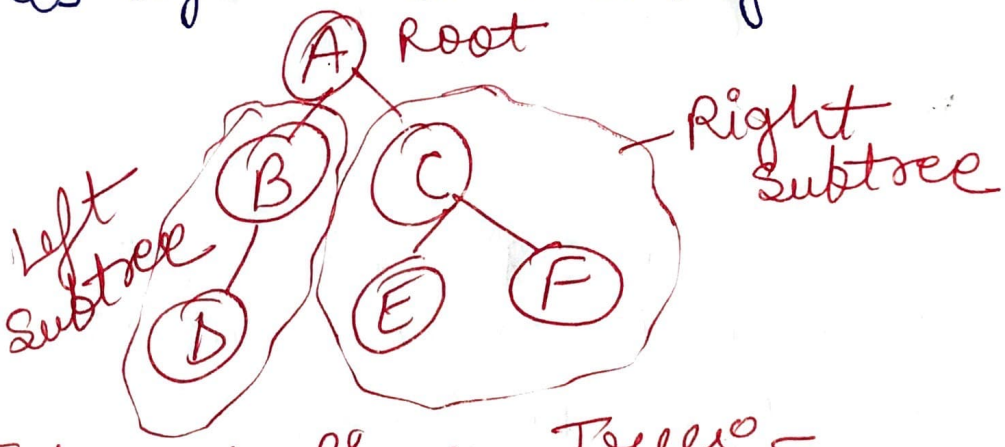
1. Any node of a tree may be root of the tree.
2. Each node except the root node has parent node.

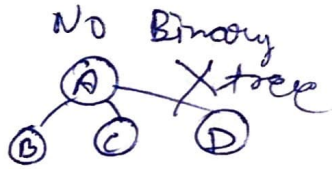
③ Every edge/Branch connects a node to it's parent so the number of branch will be n-1 in a tree, where n is the number of nodes.

Binary Trees:-

Binary tree is a finite set of data item which is either empty or consists of a single item called root and two disjoint binary tree called the left subtree and right subtree.

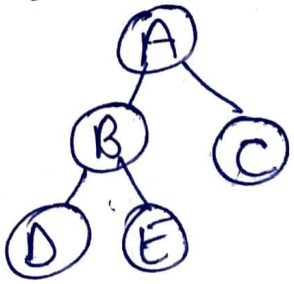
In binary tree, Every node can have maximum of 2 children which are known as left child and Right child.

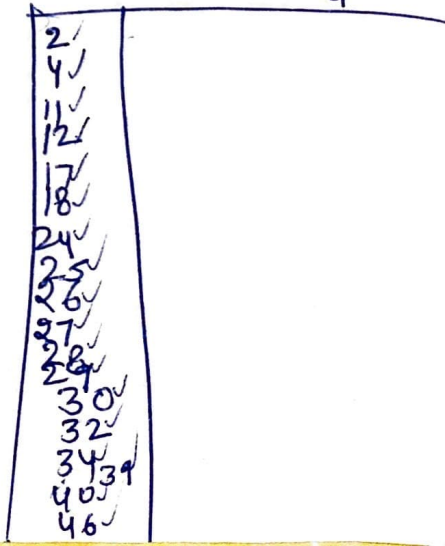




Types of Binary Trees:-

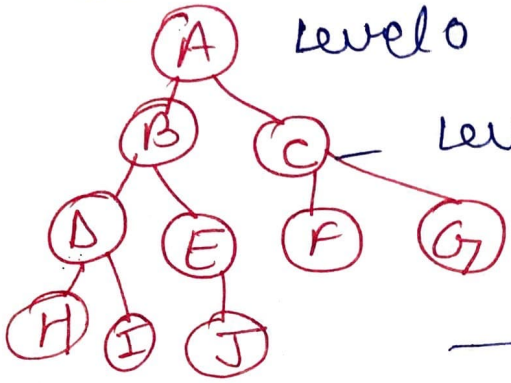
① **Full Binary Tree:-** A Binary Tree is full if every node has 0 or 2 child.





② Composite Binary tree –

A Binary tree is complete binary tree if all levels are completely filled except possibly the last level and the last level has all keys as left as possible.



level 0 $2^0 = 1$

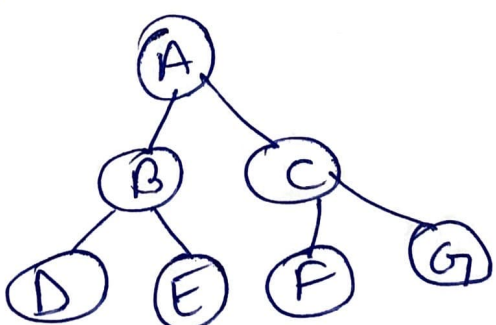
level 1 $2^1 = 2$

level 2 $2^2 = 4$

level 3 $2^3 = 8$

③ Perfect Binary tree :–

A tree in which all internal nodes has two children and all leaves are at the same level, in which all level has 2^n child.

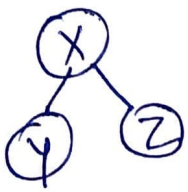


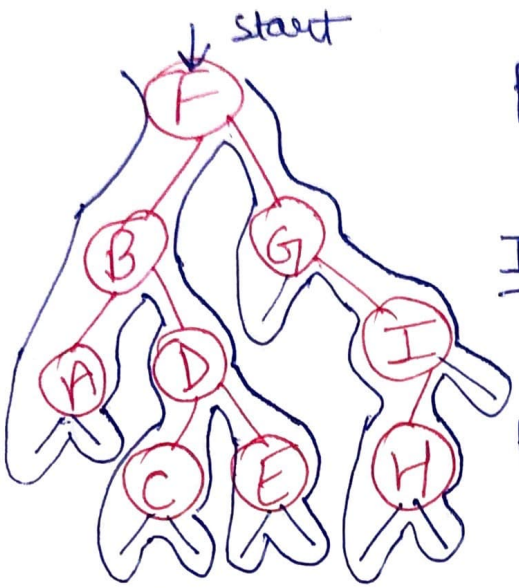
Binary Tree Traversal :–

Pre Order – Root left right X Y Z

In Order – Left Root right Y X Z

Post order – Left Right Root Y Z X



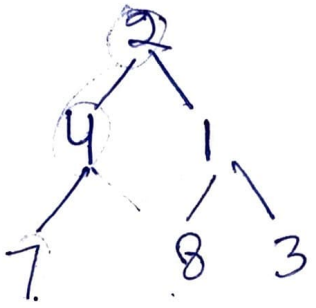


1st time
Preorder - F B A D C E G I H

2nd time
In Order - A B C D E F G H I

3rd time
Post order - A C E D B H I G F

e.g ②



Pre = 2 4 7 1 8 3

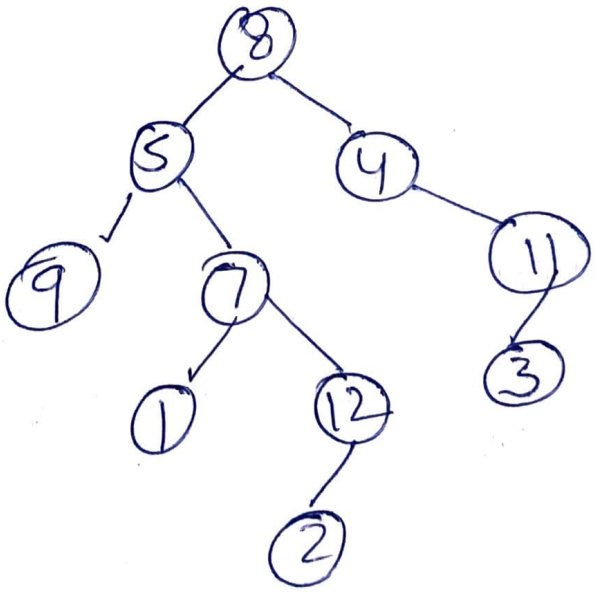
In - 7 4 2 8 1 3

Post - 7 4 8 3 1 2

construct a binary tree from Post order & Inorder -

Postorder - 9, 1, 2, 12, 15, 8, 7, 5, 3, 11, 4, 8, 10

Inorder - 9, 5, 1, 7, 2, 12, 8, 4, 3, 11

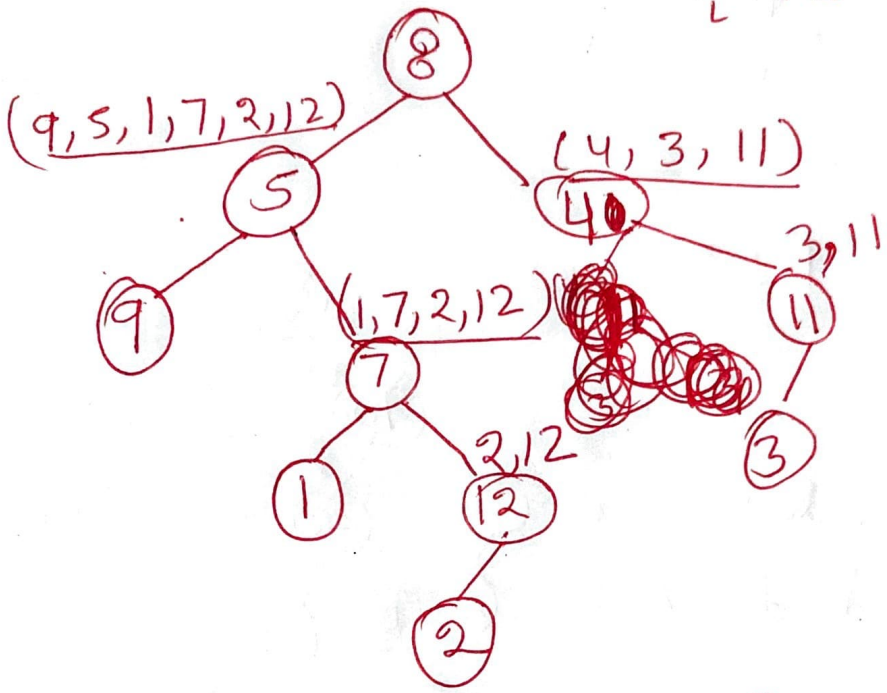


Q- Construct Binary Tree from given Preorder & Postorder.

(Travelling from here)

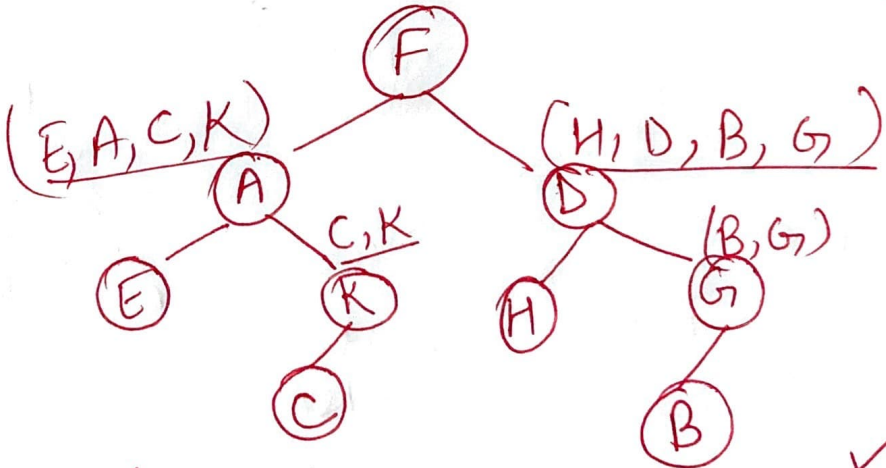
Post Order - 9, 1, 2, 12, 7, 5, 3, 11, 4, 8 (L R Root)

In Order - 9, 5, 1, 7, 2, 12, 8, 4, 3, 11 (L Root R)



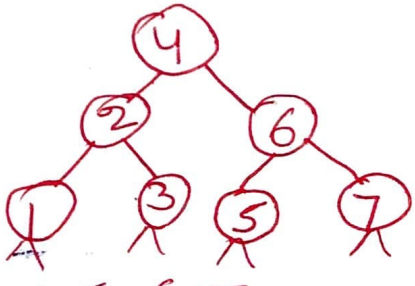
Q2- Inorder - E A C K F H D B G

Post Order - E C K A H B G D F



Create Binary Search Tree (BST) with following keys -

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



Inorder of a BST is always sorted.

1 2 3 4 5 6 7

→ Left child must be less than the root
→ Right child must be more than the root.

Left Biased

Deletion in BST–

Node
- Leaf
  - 0 child
    - Simply deleted without any changes
- Non leaf
  - 1 child or 2
    - add with Grand Parent

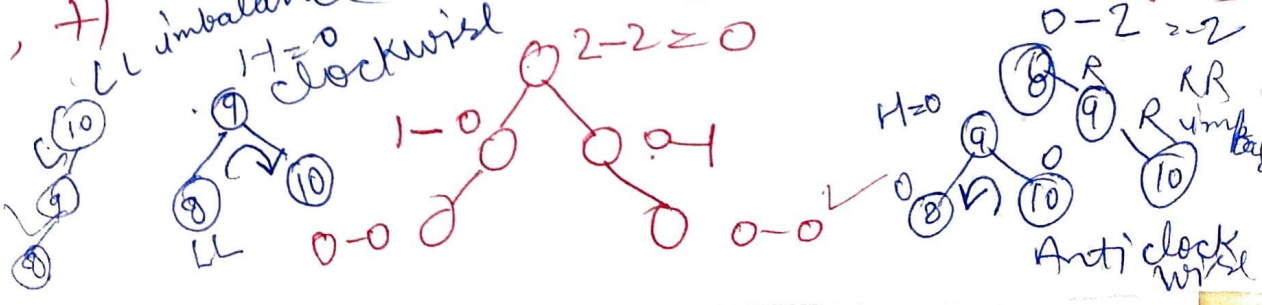
AVL Def–

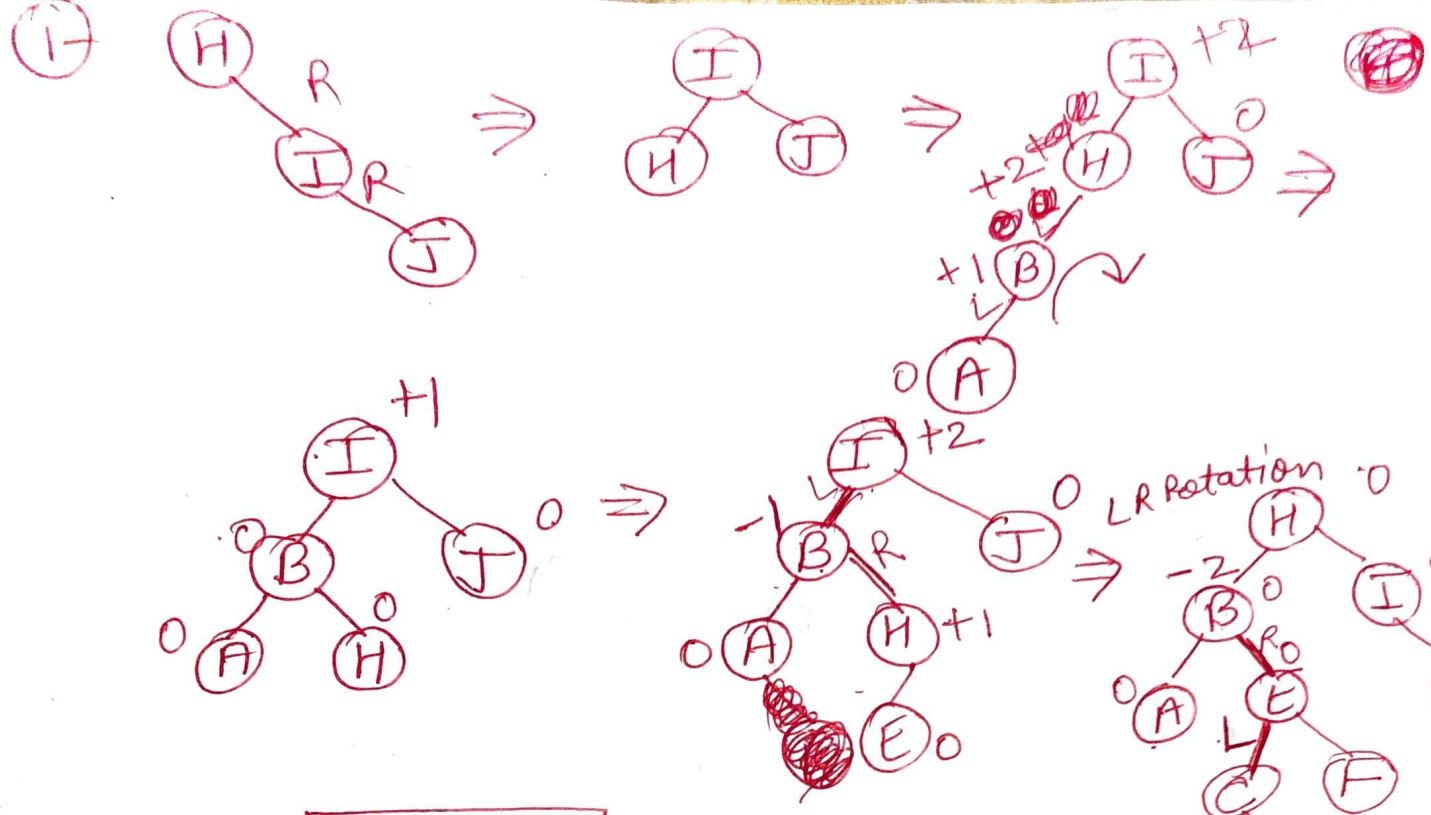
(OR)
Balanced BST

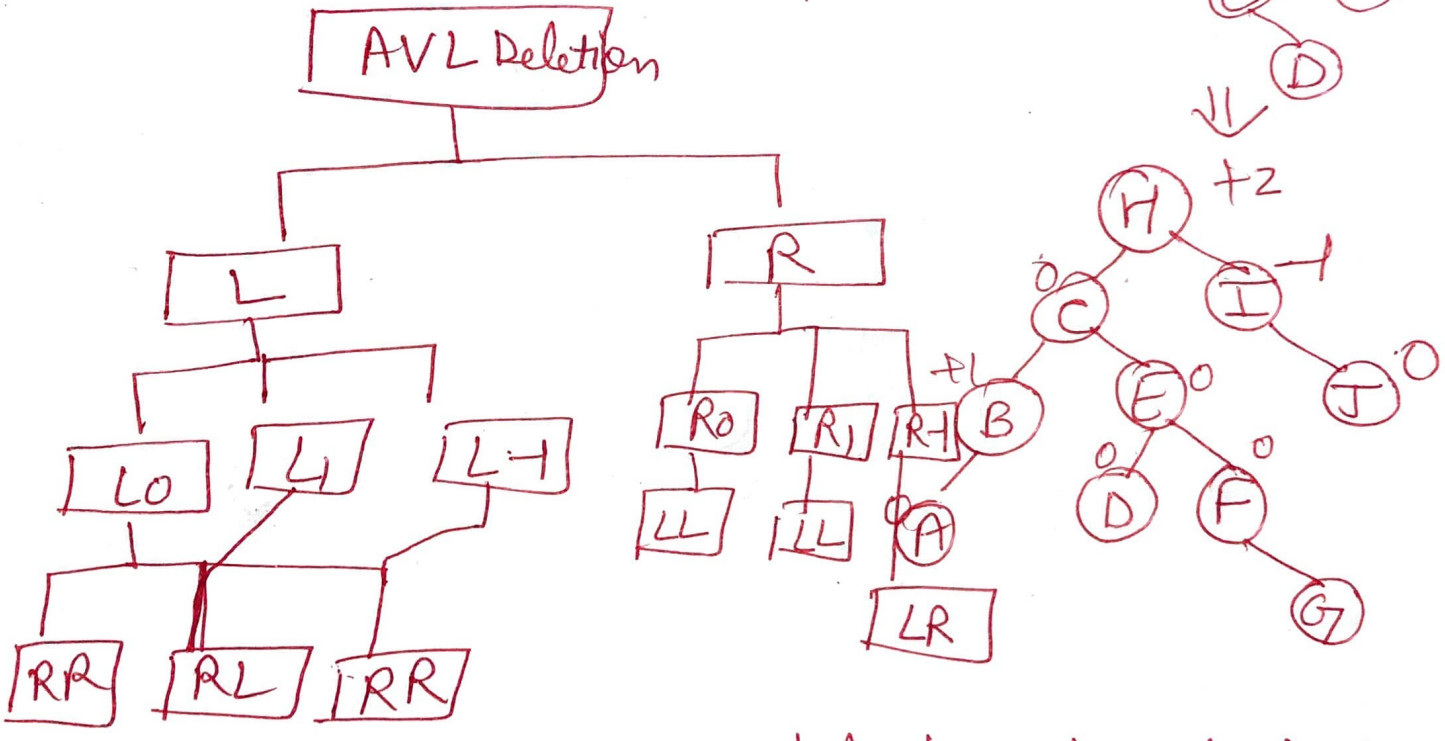
(height = log n)
AVL

Balance Factor = (height of LST − height of RST)

−1, 0, +1

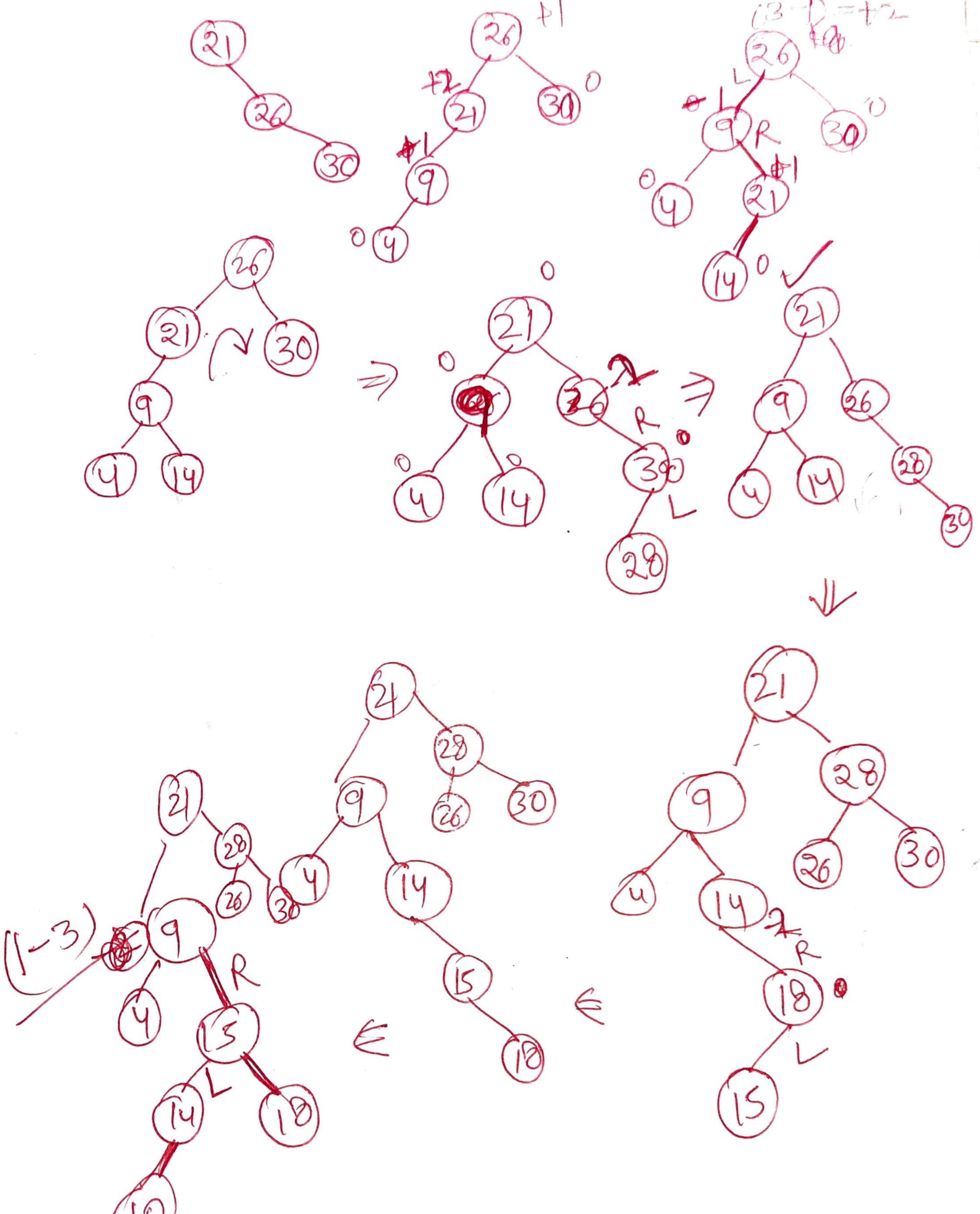






if left's side deletion hua hai to left ka other side children ka BF ?

Q⇒ 21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7 Insert in AVL tree.

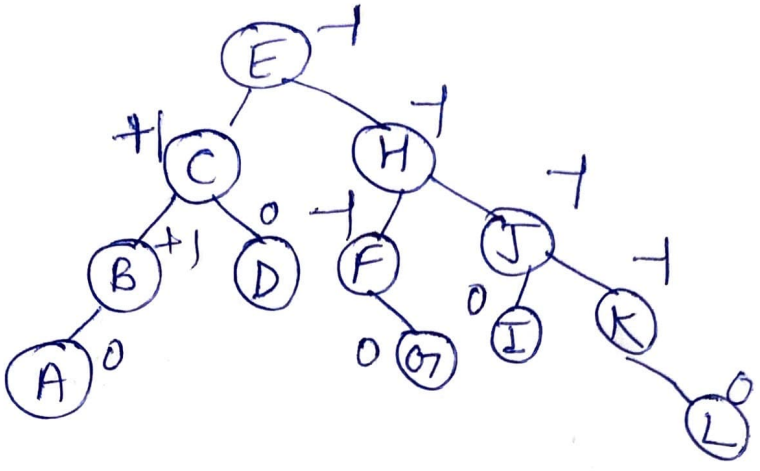


Anti clockwise

Q1- Create an AVL search tree from the given set of values:-

H, I, J, B, A, E, C, F, D, G, K, L

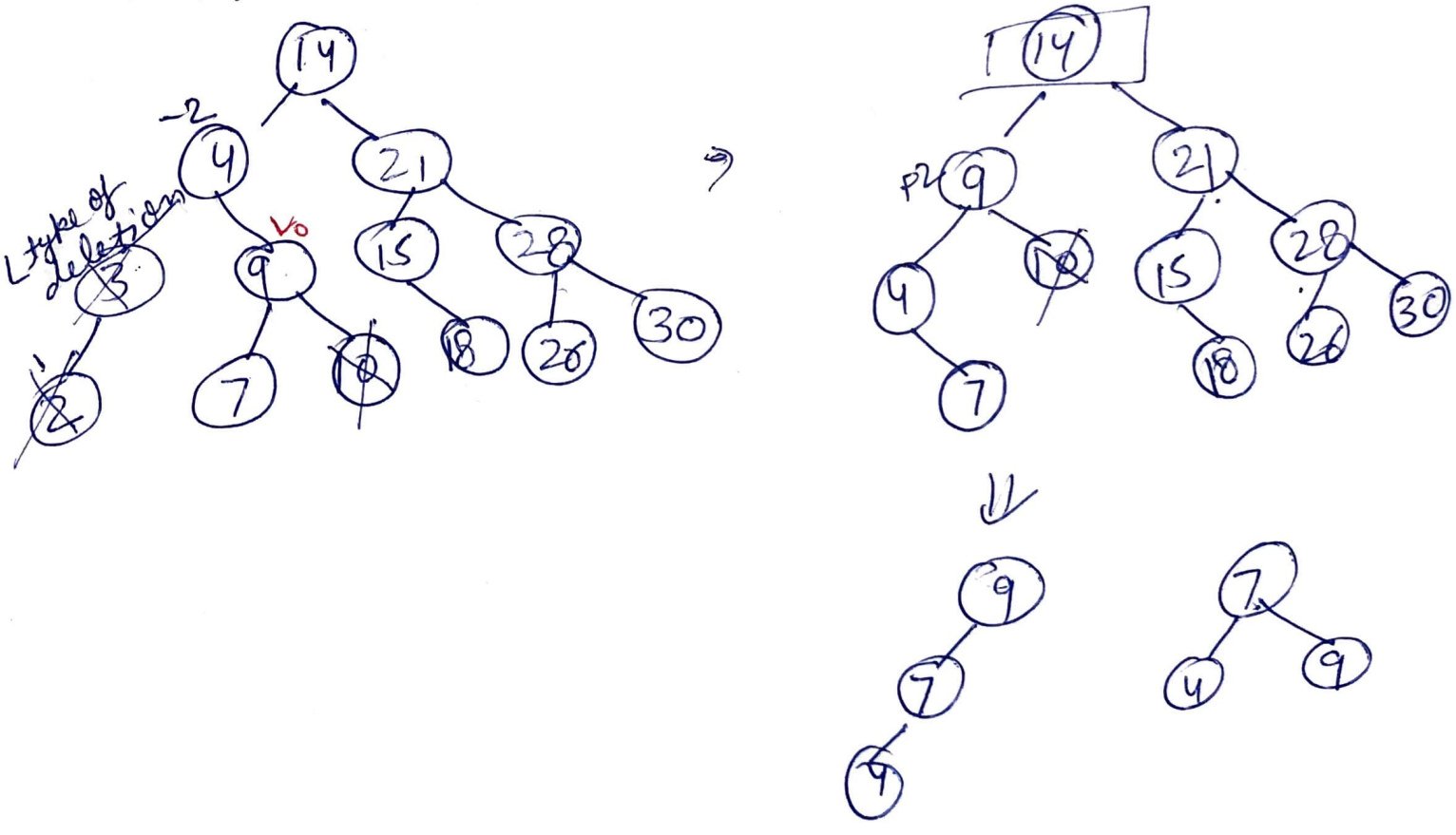
Sol^n



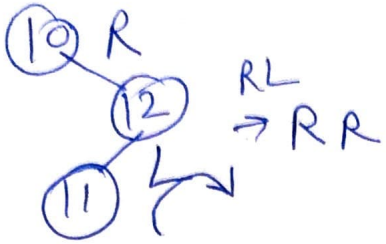
Deletion in AVL Tree -

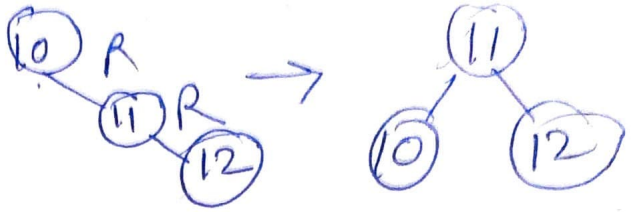
Delete the following nodes in sequence

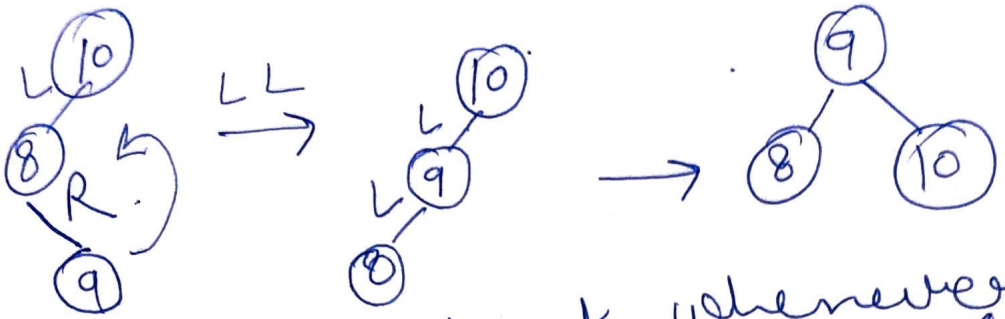
2, 3, 10, 18, 4, 9, 14, 7, 15



III case -



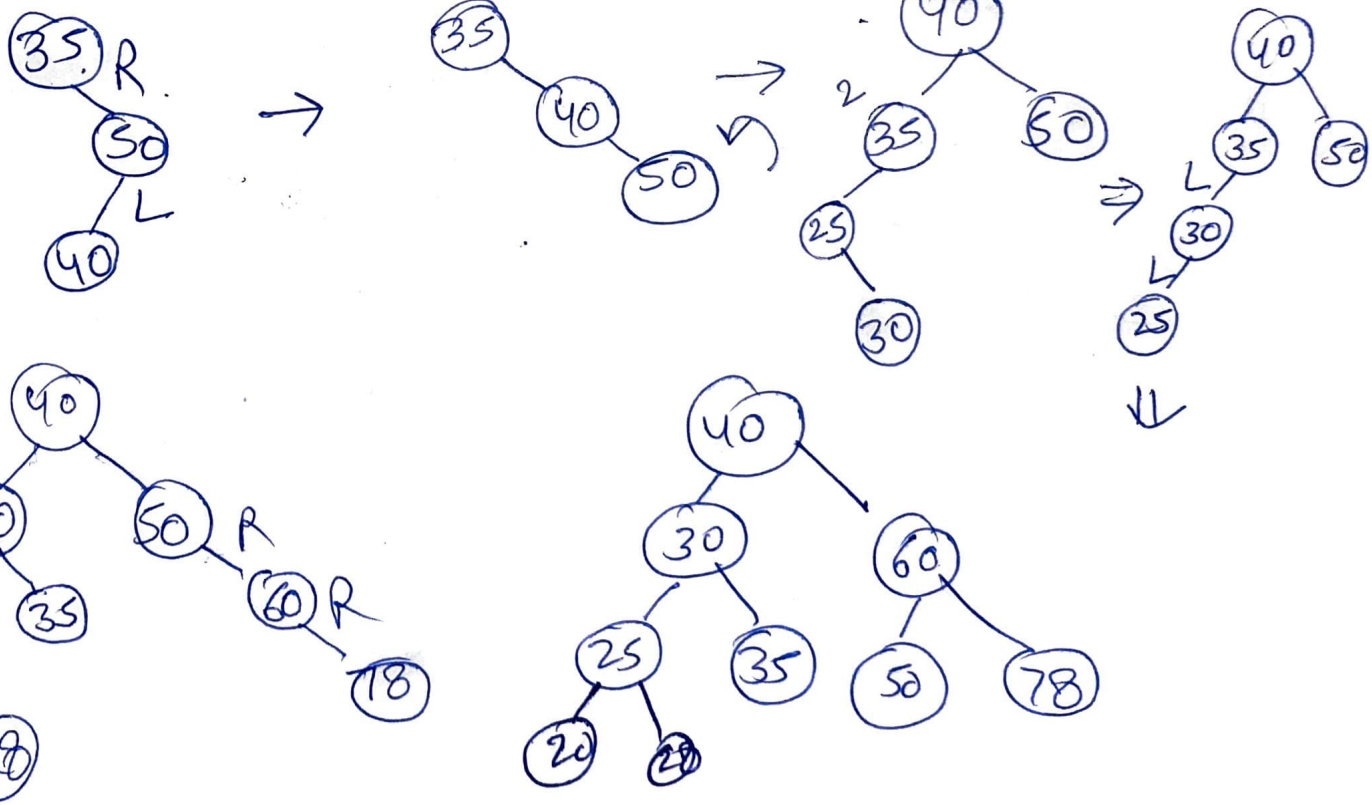




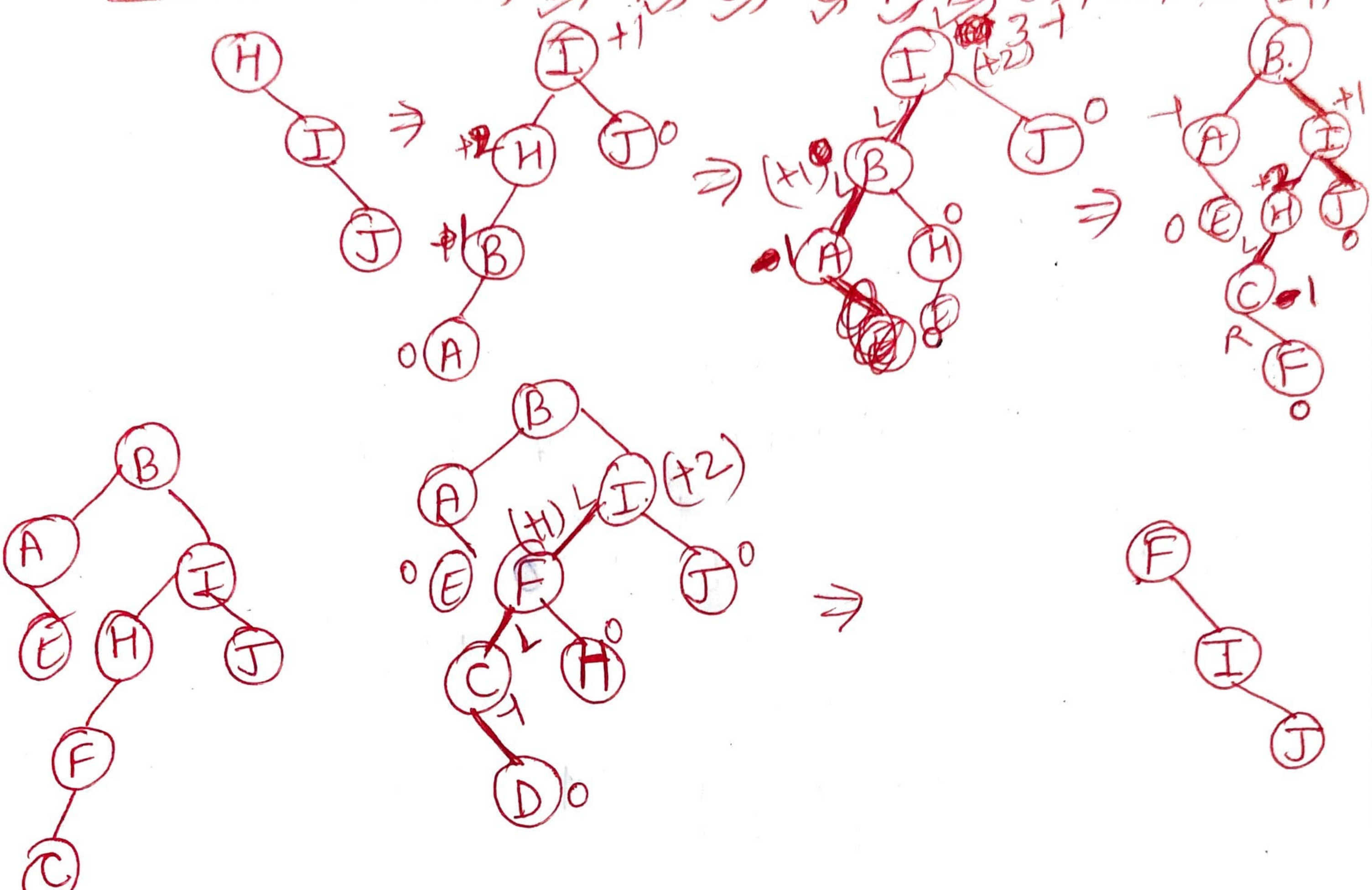
Everytime check whenever insert new element in upward direction.

Q- How many rotations are required during construction of an AVL tree if the following elements are added in order given. (-1, 0, +1)

35, 50, 40, 25, 30, 60, 78, 20, 28



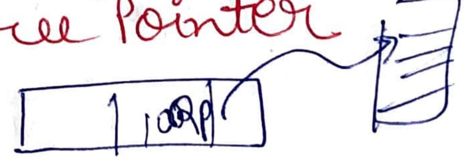
eg: Soln H, I, J, B, A, E, C, F, D, G, K, L



B Tree – (Dynamic Multilevel Index) –

Tree is acyclic –

Block Pointer
(OR)
Tree Pointer



Keys
Data Pointer
Record Pointer

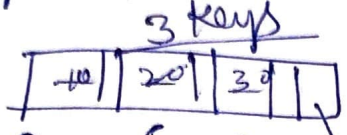
Keys = Record Pointer
(Points the record in SM)

Block pointer = No of children

Block pointer = Order = p = Max no of children

Children	Root	Intermediate Node (children)
Max	p	p
Min	2	$\lceil p/2 \rceil$

if $p = 4$



Max keys $= p - 1$

$R_p = p - 1$

Insertion in B Tree –

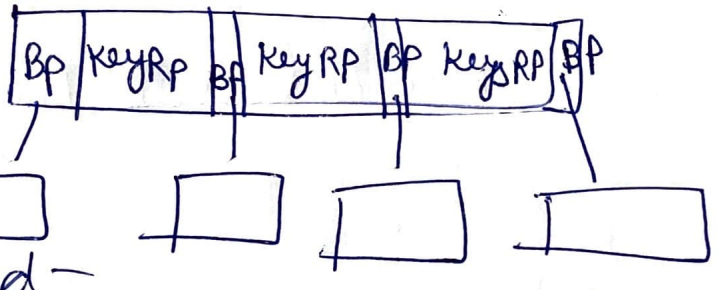
Q- Insert the following keys in B-Tree, if order of B Tree = 4 (Max 4 children)

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

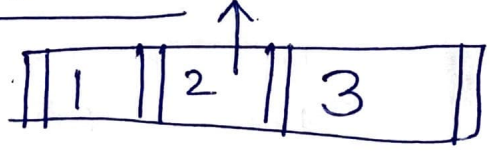
Max keys $= p - 1$ $\quad BP = 4$

$= 4 - 1 = 3$

worst case Min keys $= \lceil \frac{p}{2} \rceil - 1 = 1$



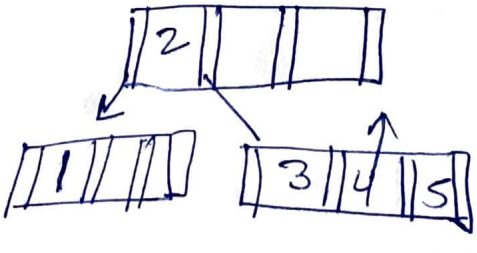
sorted –



4 overflow

1, 2, 3, 4

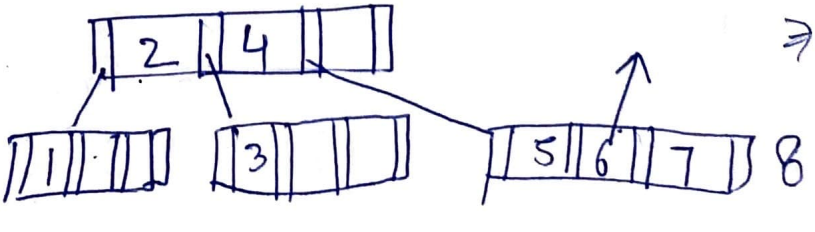
Median $= \frac{n}{2}$



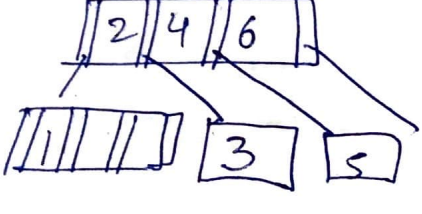
left

3, 4, 5, 6

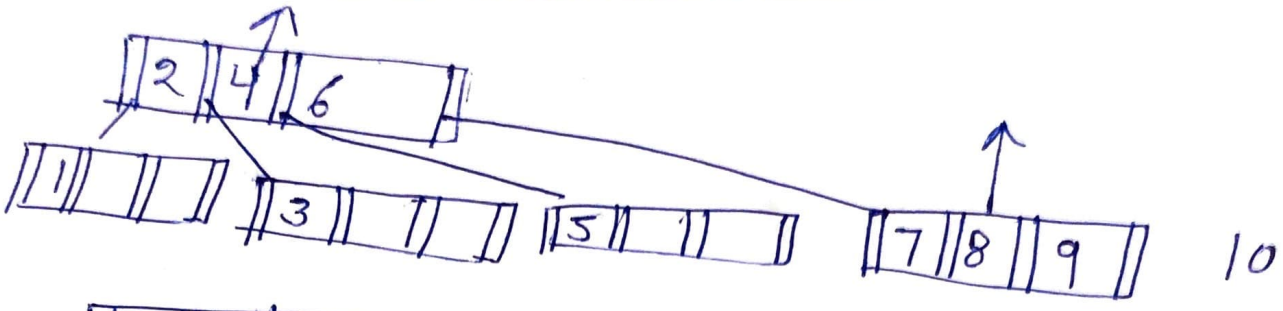
median

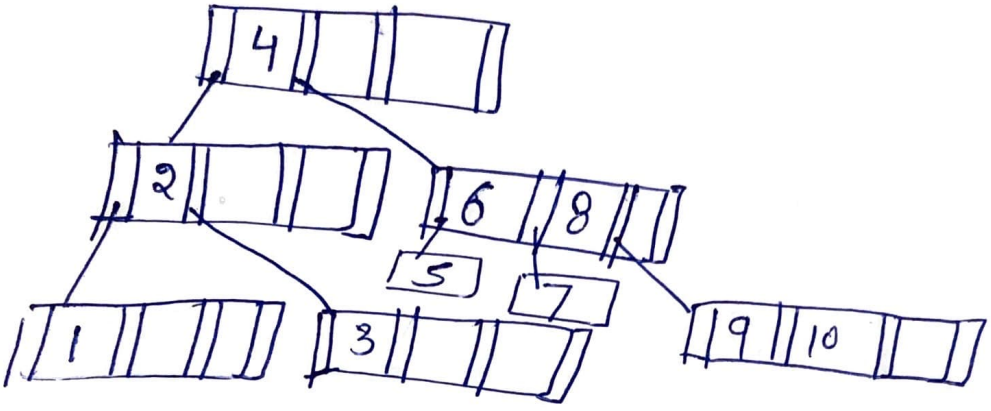


⇒



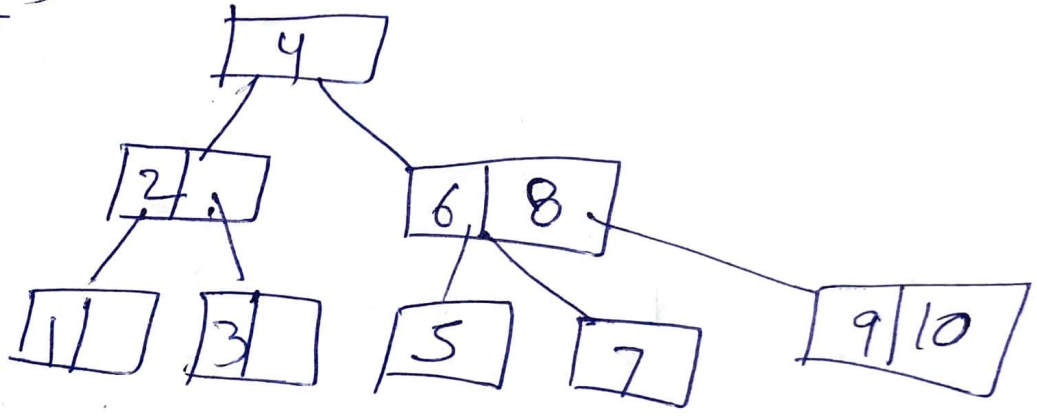
7 8 9 10





height = 2
level = 3

Order = 3



Q→ consider a B-Tree with key size = 10 bytes, block size 512 bytes, data pointer is of size 8 bytes and block pointer is 5 bytes. Find the order of B-Tree?

if $B_p = n$

key = n-1

$$n \times B_p + (n-1)\,\text{keysize} + (n-1)\,\underset{\text{size}}{RP} \leq \text{block size}$$

(RP = R_p; or Node)

$$n \times 5 + (n-1)(10+8) \leq 512$$

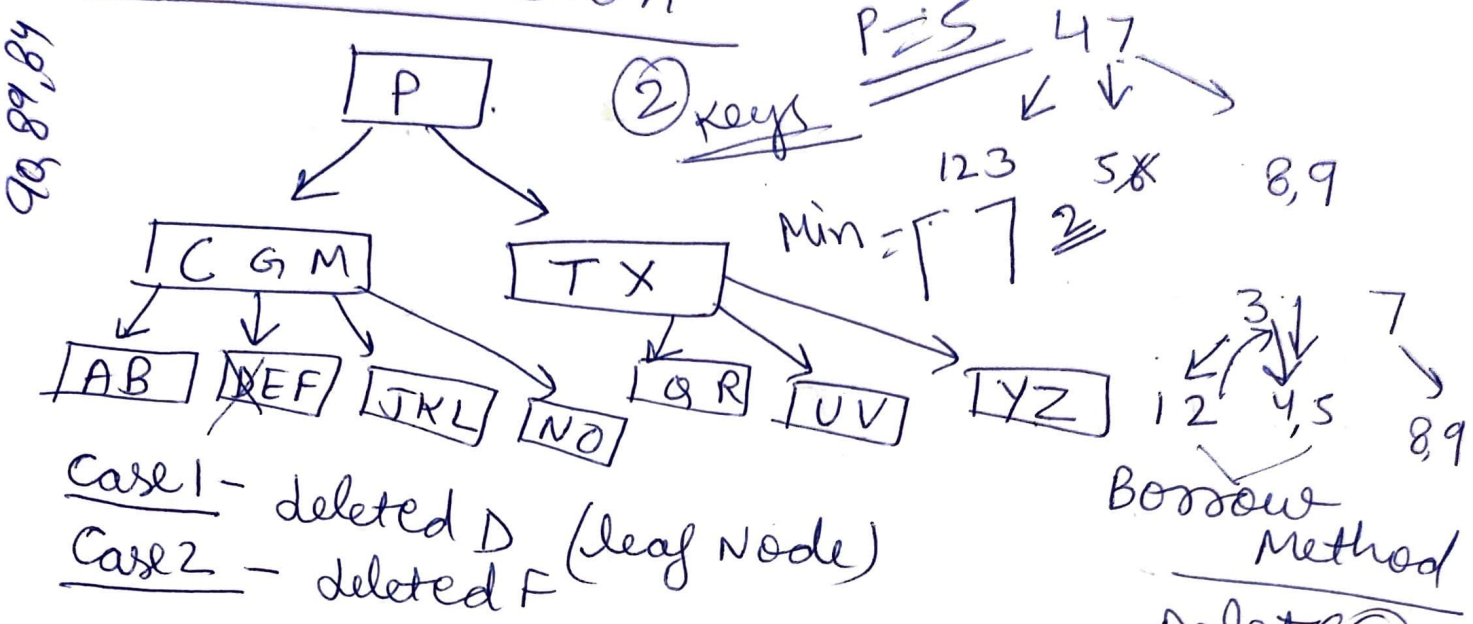
$$5n + 18n - 18 \leq 512$$

$$23n \leq 512 + 18$$

$$n \leq 23.04$$

$(n = 23)$

B Tree Deletion –



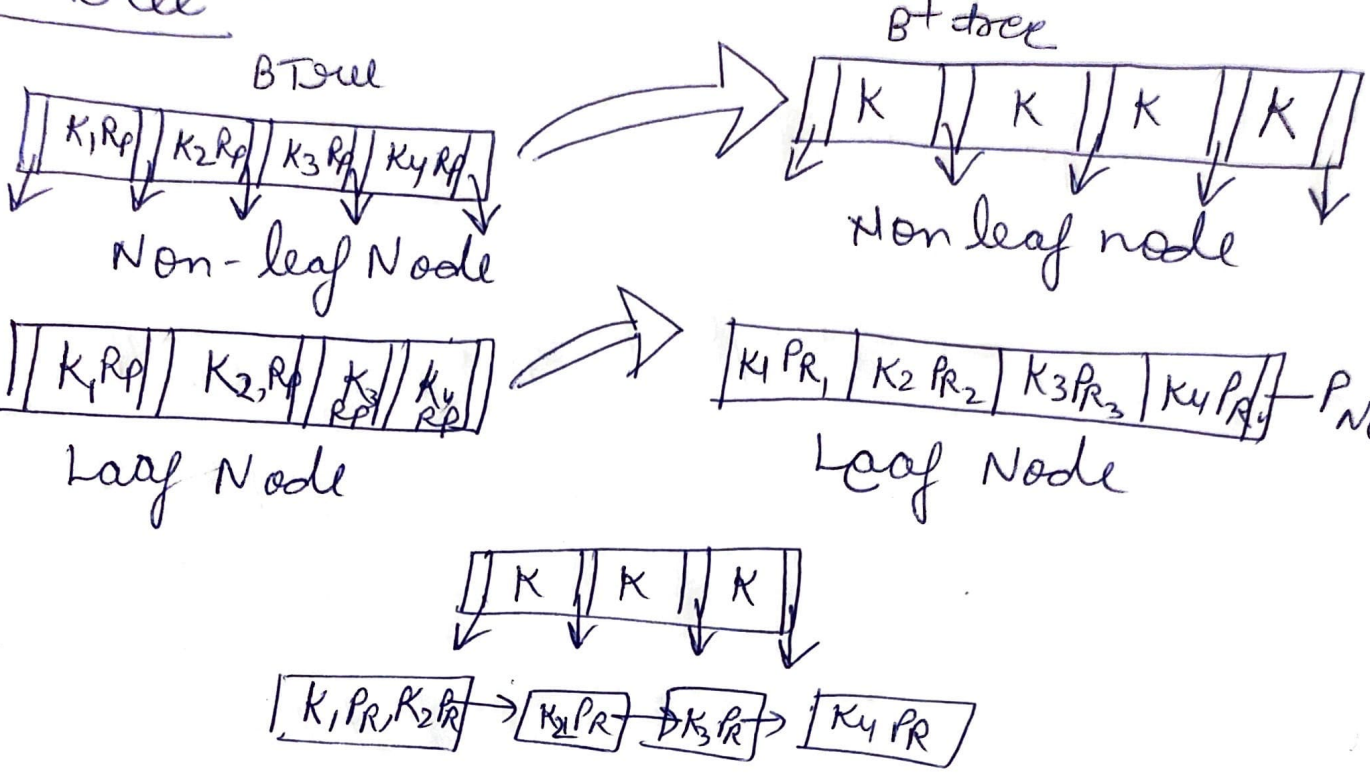
Case 1 – deleted D (leaf Node)

Case 2 – deleted F

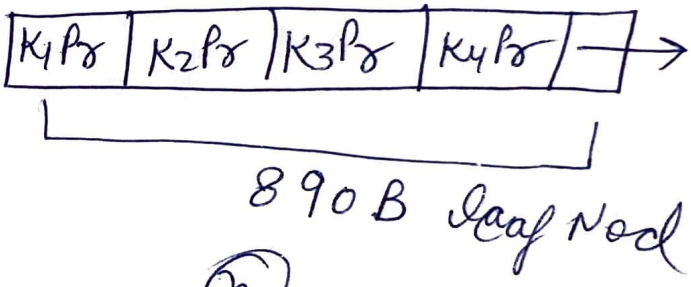
Difference b/w B Tree & B+ Tree (P1)

B Tree	B+ Tree (4)
→ Data is Stored in leaf as well as internal nodes	→ Data is stored only in leaf nodes.
→ Searching is slower deletion complex.	→ Searching is faster deletion easy (directly from leaf node)
→ No redundant Search key present.	→ Redundant keys may present.
→ Leaf Nodes not linked together.	→ Linked together like linked list.

B+ Tree –



Q→ Given K = 10B
BS = 890B
P_b = 20B
P_r = 30B



(n)

$$n \times (K + P_r) + P_b \leq 890$$

$$n \times (10 + 30) + 20 \leq 890$$

Sparse Matrix –

Array Representation –

	0	1	2	3	4	5	6	7	8
Row	0	1	1	2	4	4	5	6	6
Col	4	2	6	0	1	5	7	2	4
Value	5	4	6	1	8	3	2	10	7

$3 \times 9 = 27 \times 2$ bytes
$= 54$

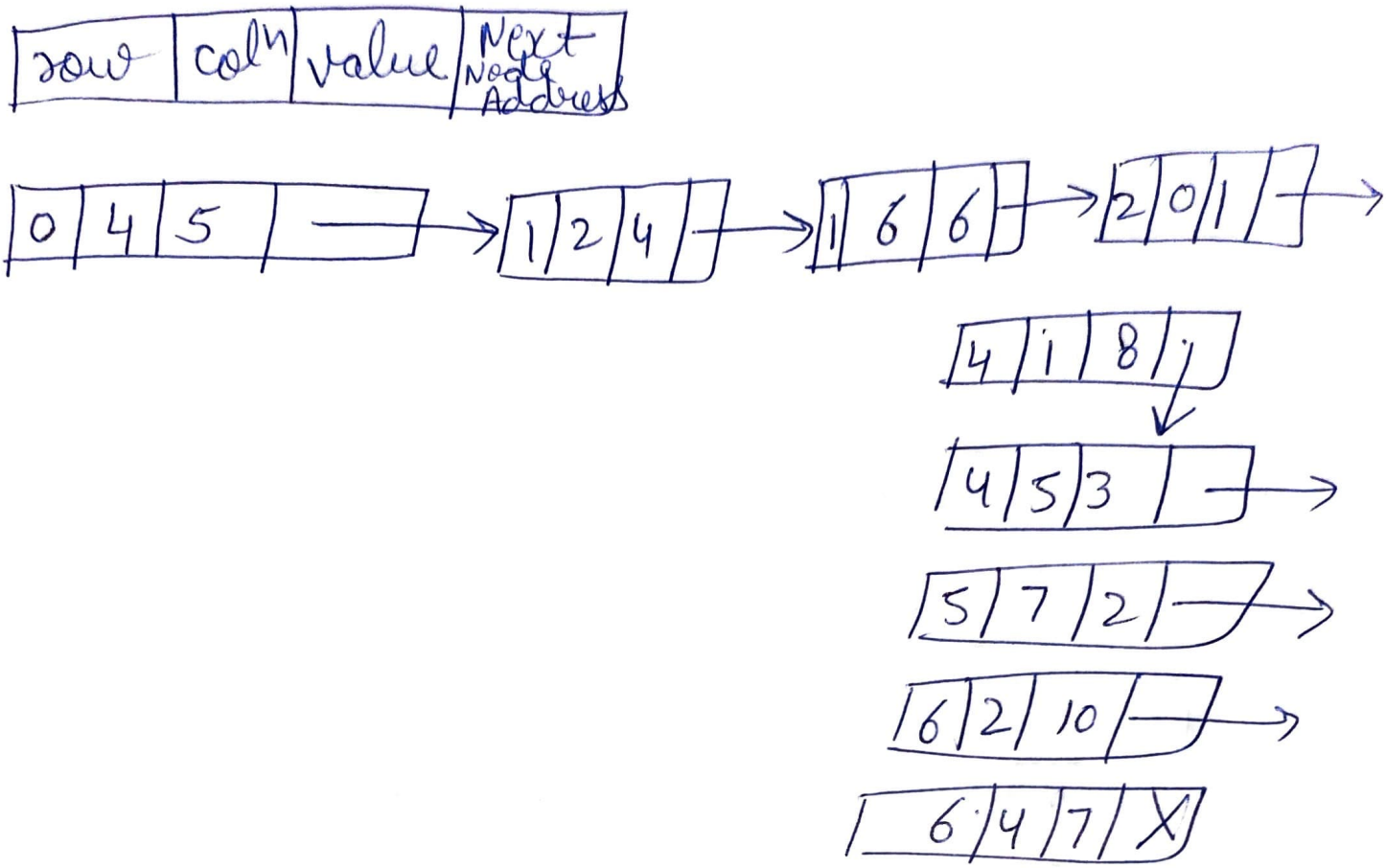
Sparse Matrix

	0	1	2	3	4	5	6	7
0	0	0	0	0	5	0	0	0
1	0	0	4	0	0	0	6	0
2	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	8	0	0	0	3	0	0
5	0	0	0	0	0	0	0	2
6	0	0	10	0	7	0	0	0

7 × 8

$7 \times 8 = 56 \times 2 = 112$ bytes

Linked list Representation



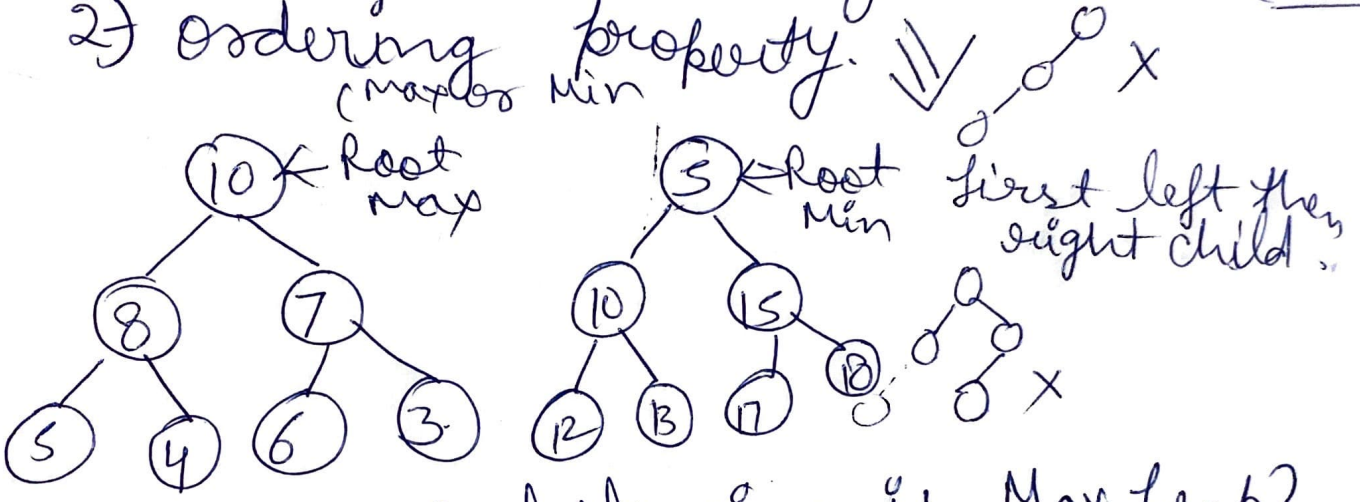
Introduction to Heap Tree

(Insertion, deletion, Heap Sort, Heapify, Array representation, Priority Queue)

- Maxheap (Parent > child)
- Min heap (Parent < child)

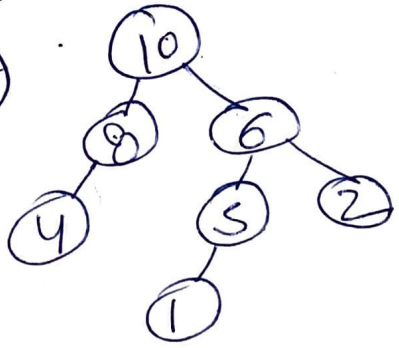
Must follow—

1) Structural Property — It must be ACBT
2) Ordering property (Max or Min)

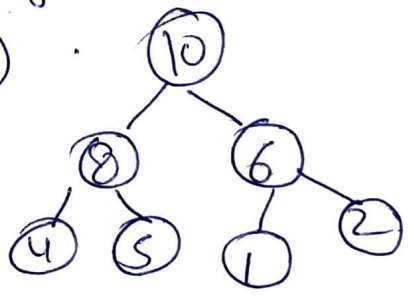


Q→ which of following is Max heap?

a)

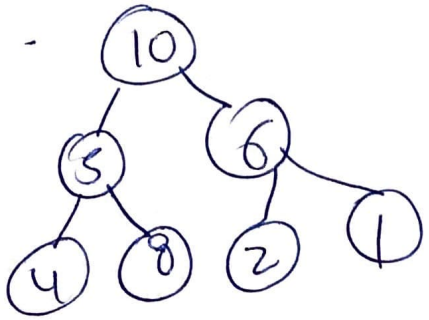


b)

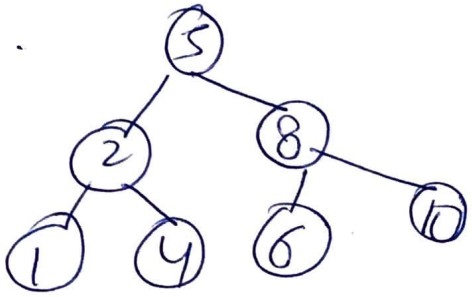


✓

c)

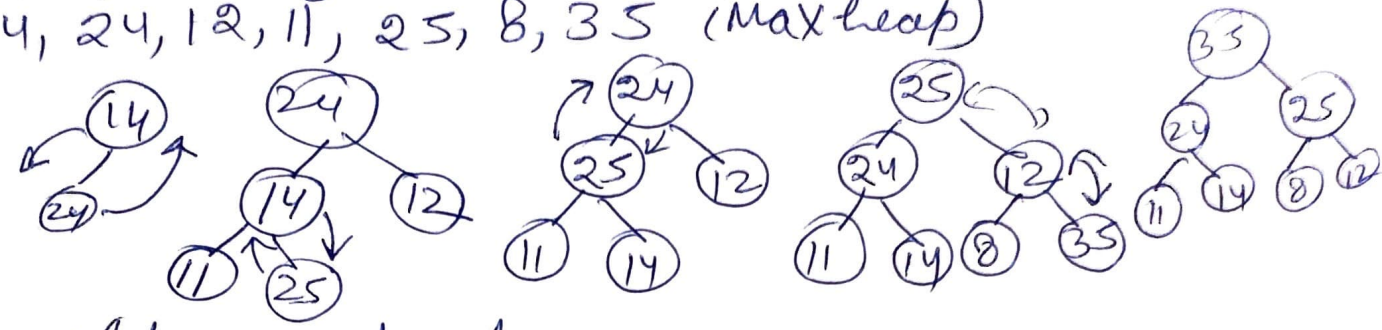


d)



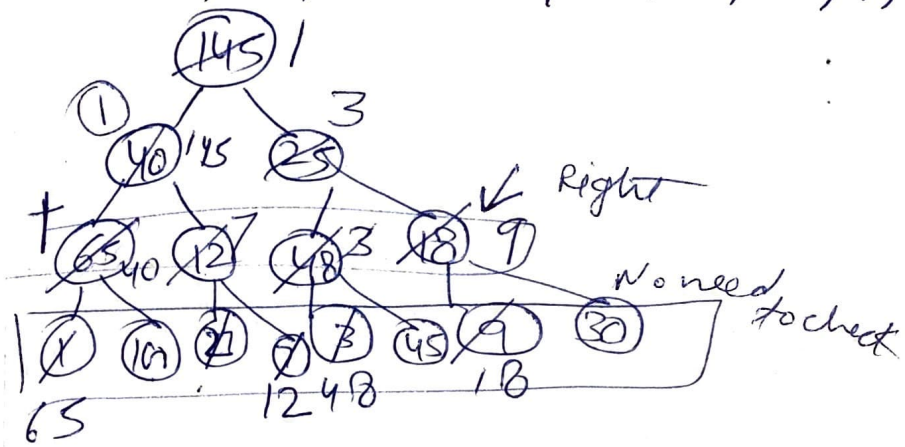
Heap Tree construction –

14, 24, 12, 11, 25, 8, 35 (Max heap)



Heapify Method –

145, 40, 25, 65, 12, 48, 18, 1, 100, 27, 7, 3, 45, 9, 30



Deletion in Heap Tree –