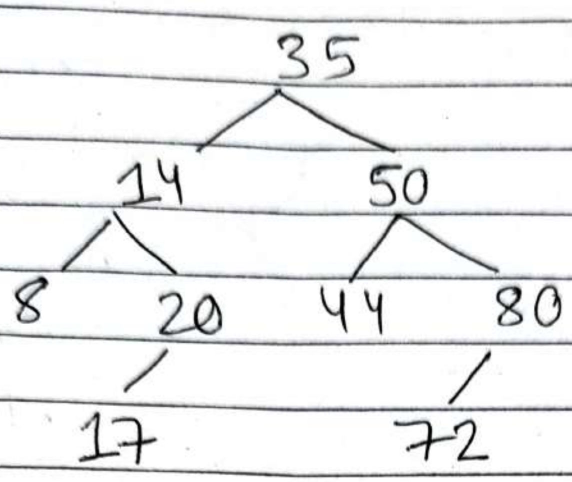
Binary Search Tree (BST)

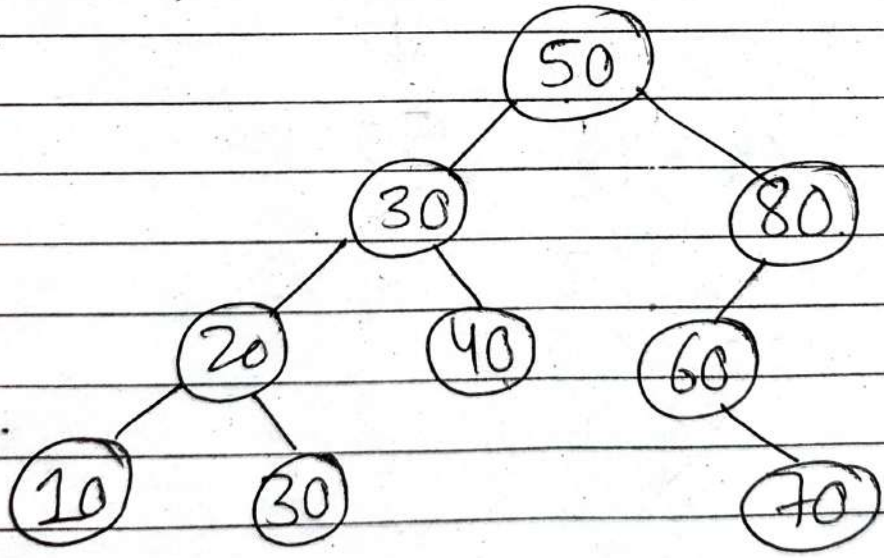


at most 2

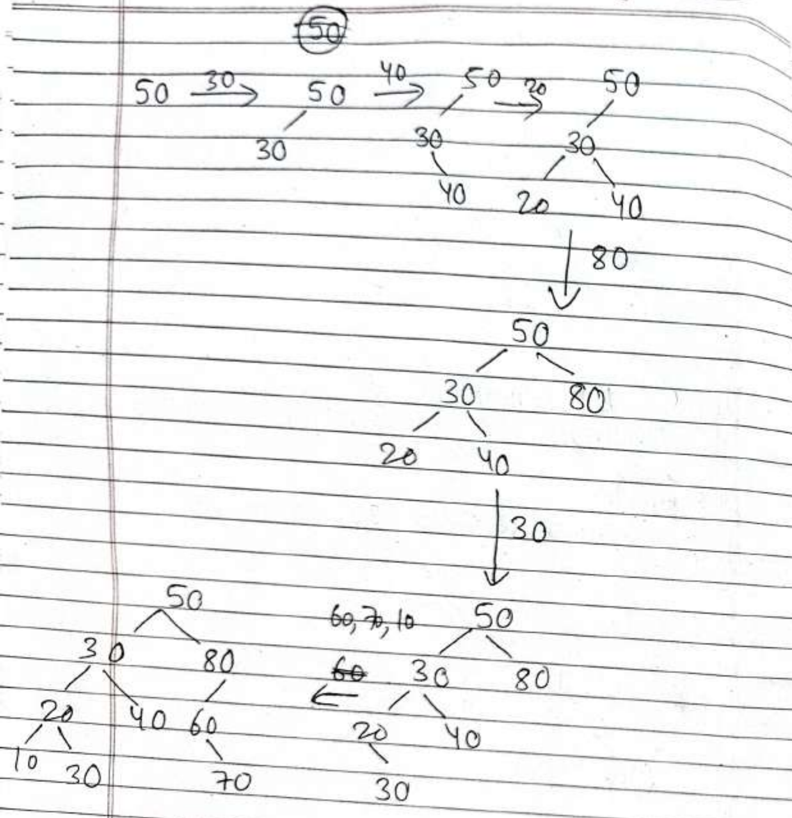
Left ≤ Data < Right
(subtree) (subtree)

Example:

50, 30, 40, 20, 80, 30, 60, 70, 10

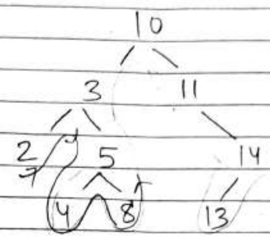


2



3

*) In-Order Traversal (Left, Root, Right)
(print)



$\underbrace{2, 3, 4, 5, 8}_{\text{Left subtree}}$, $\underbrace{10}_{\text{Root}}$, $\underbrace{11, 13, 14}_{\text{Right subtree}}$ (Sorted in ascending order)

*) Deletion :

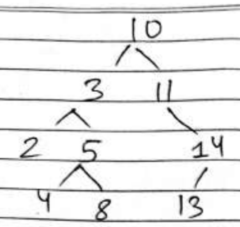
Three cases:

Case I : Deleting leaf node

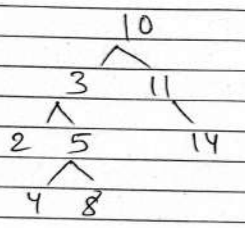
Case II : Deletion of node who has only one Child

Case III : Deletion of node who ~~has~~ has two Childs.

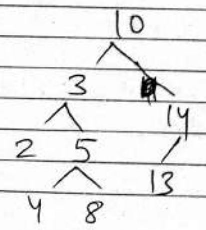
Deletion



<u>Case I:</u> 4, 8, 13 → any leaf node can be deleted.
(13 is deleted.)



<u>Case II:</u> Node with 1 child are 11, 14.
(11 is deleted).



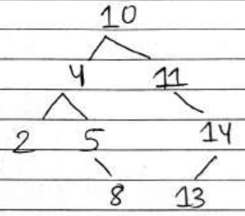
<u>Case III:</u> Node with 2 childs is deleted.

(Nodes with 2 childs are 10, 3, 5.

2, 3, 4, 5, 8, 10, 11, 13, 14 (Inorder traversal)

⇒ If 3 is deleted then, adjacent element (4) will be used to replace 3.

↑
(inorder)



(6)

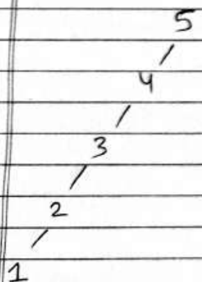
*) Introduction to AVL Tree:

=> also called as height balance tree.
=> it is binary search tree.
=> balance tree's concept was introduced in 1962 by ~~Adelson-Velskil-Landis~~ Adelson-Velsky & Landis

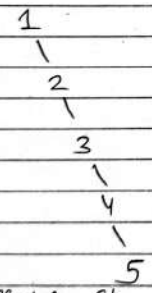
-> self-balancing ~~tree~~ binary search tree.

Balance Factor $= h(T^L) - h(T^R)$

Problem with BST:



$\boxed{O(n)}$



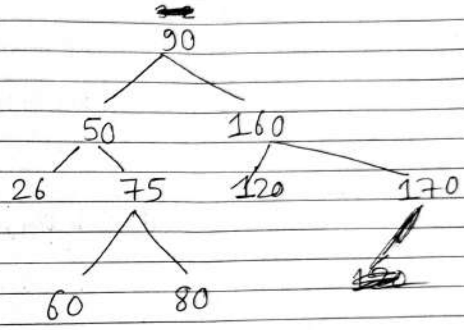
Left-Skewed BST   Right-Skewed BST

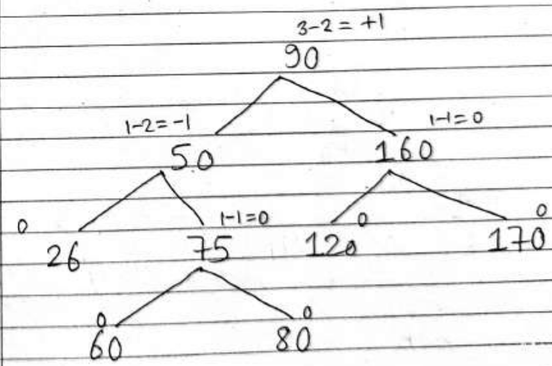
=> Time complexity of BST : log(n).

=> Worst case: O(n)
(Linear)

(7)

Balance Factor $= h(T^L) - h(T^R)$ [0, +1, -1]

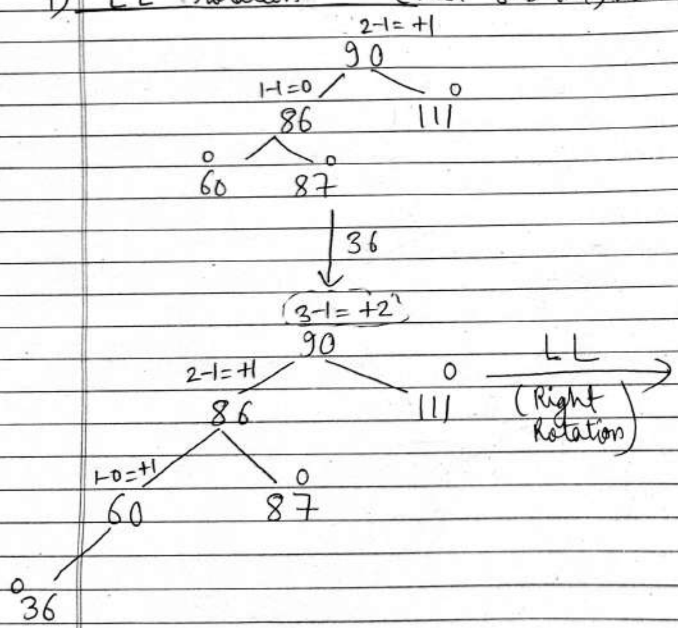


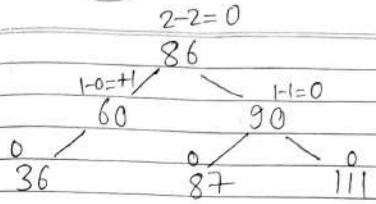


*) Insertion into an AVL tree
(Rotations in AVL tree):

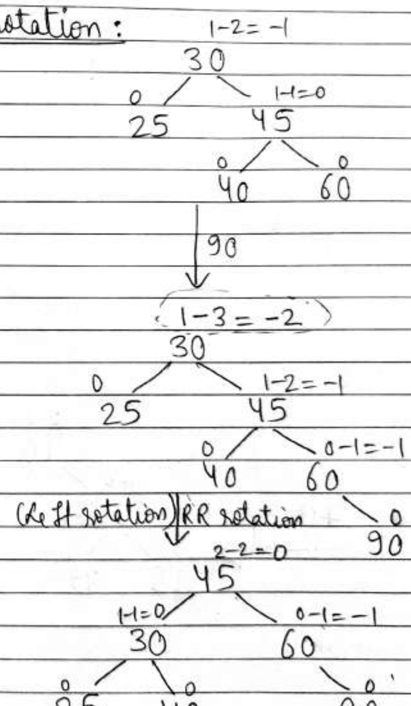
i) LL rotation (Right rotation) } single rotation
ii) RR rotation (Left rotation) }
iii) LR rotation } Double rotation
iv) RL rotation }

i) LL rotation (Left sub-tree, Left node):





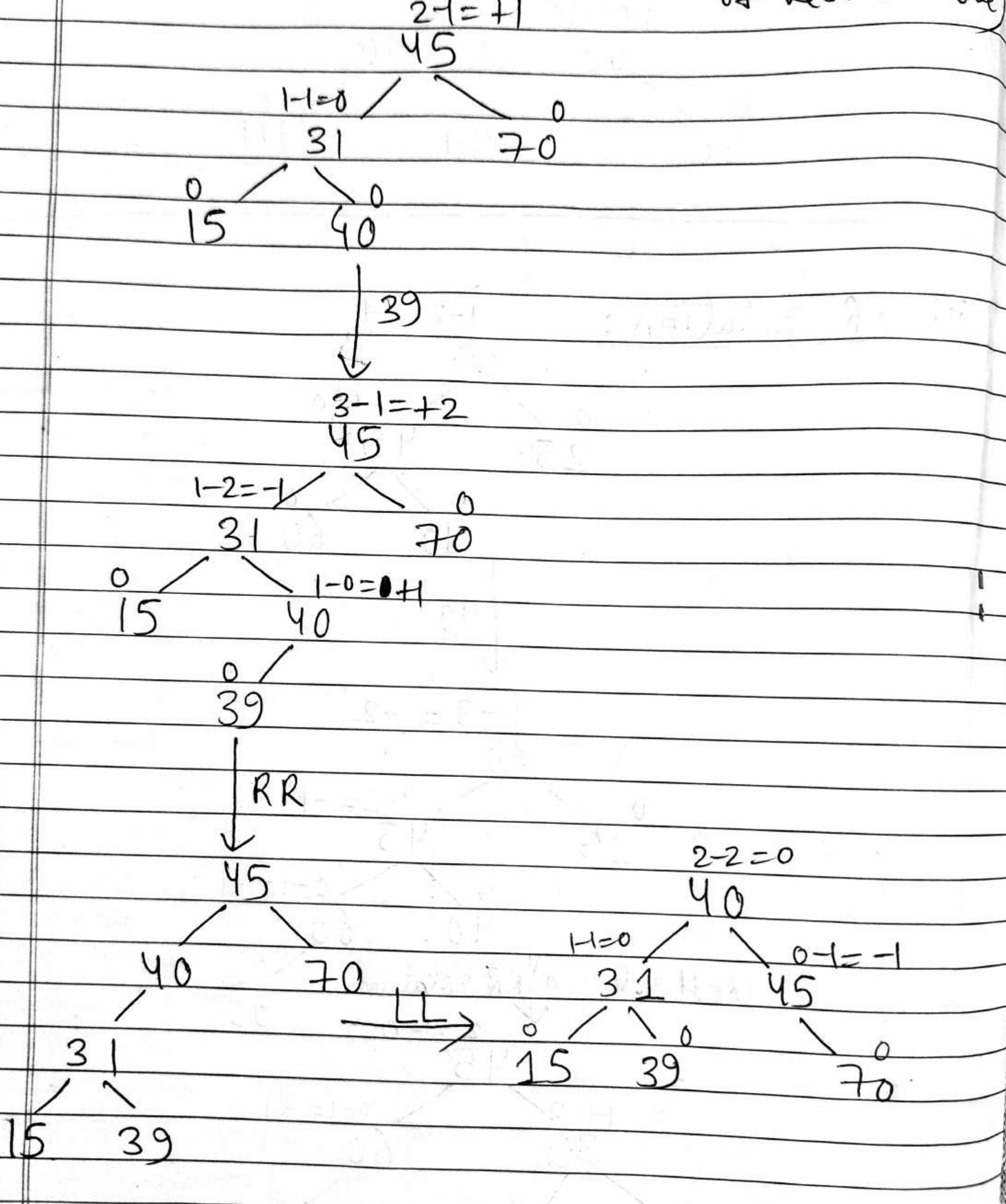
ii) RR rotation:



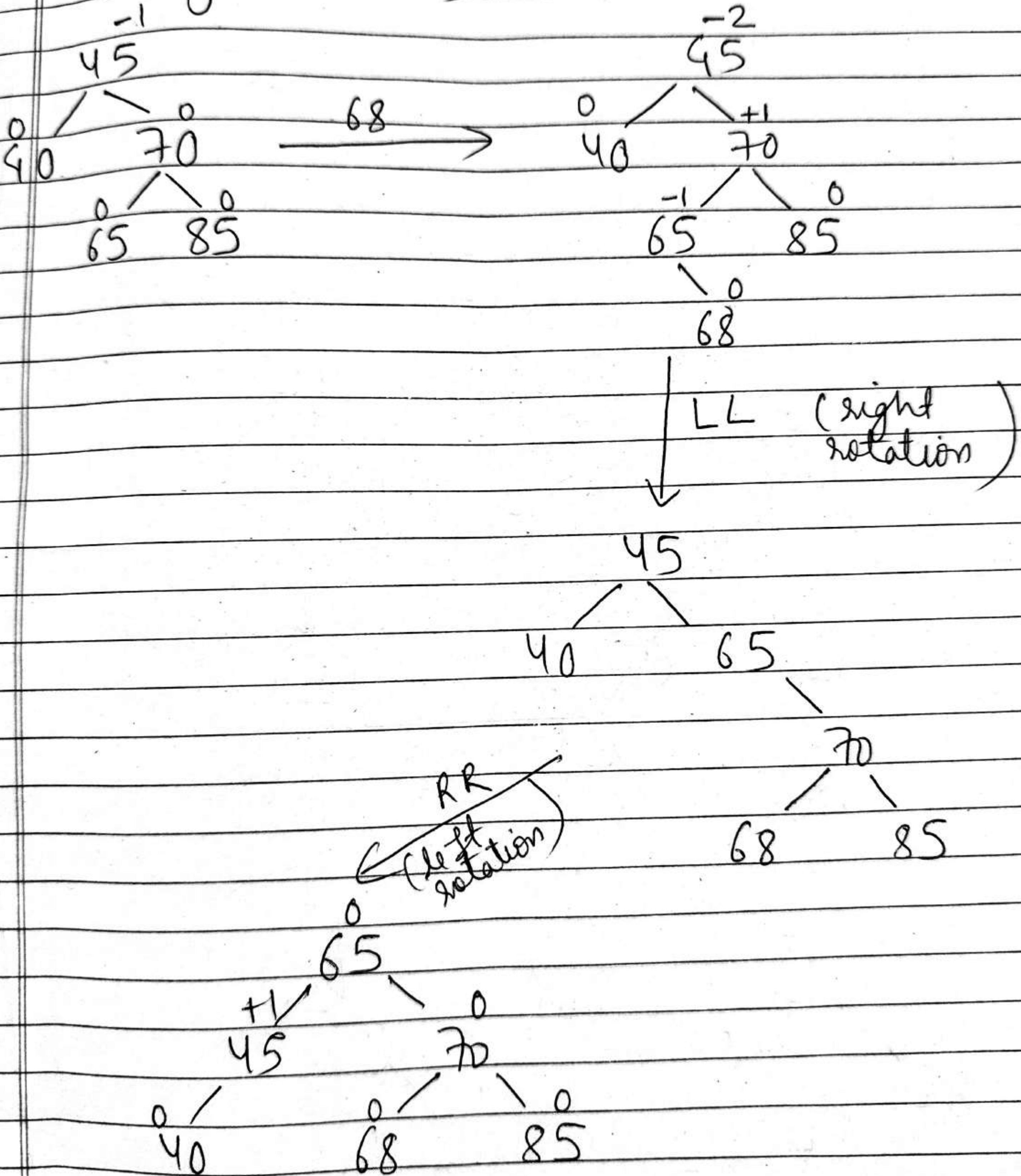
10

(RR → LL)

iii) <u>LR rotation :</u> (Add new node to right node of Left sub-tree)

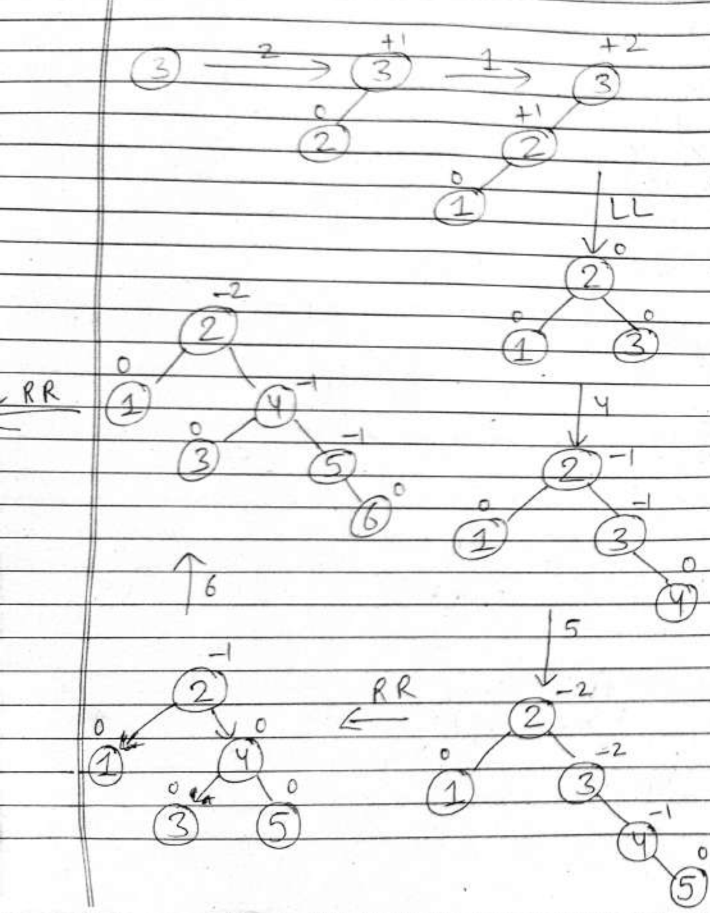


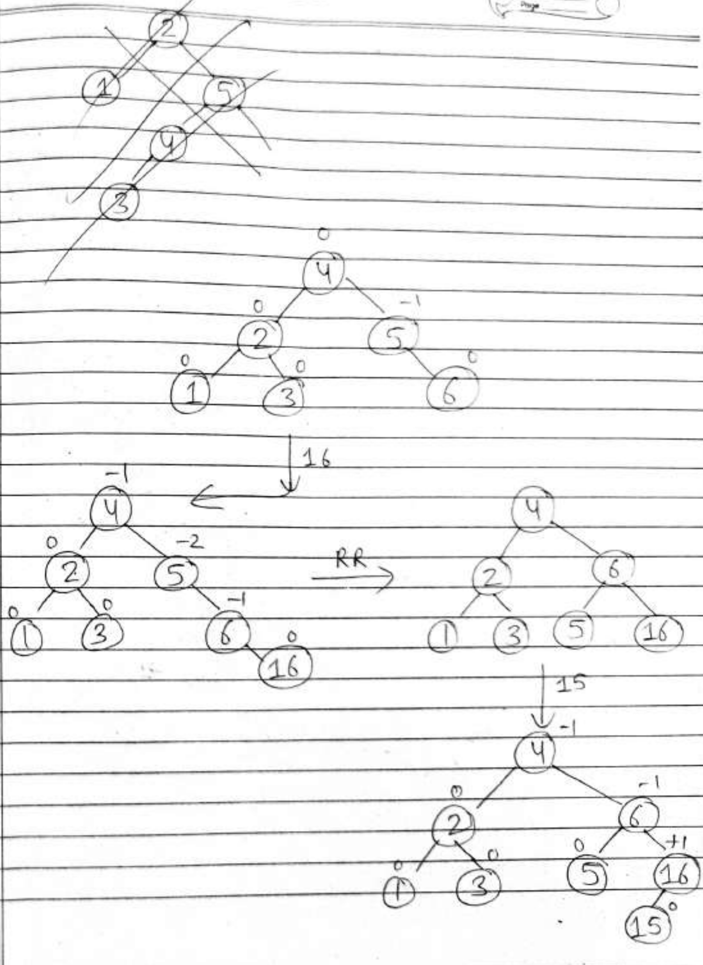
IV) RL rotation (LL→RR):
(Add new node to left node of right subtree).



*) Construct AVL tree:

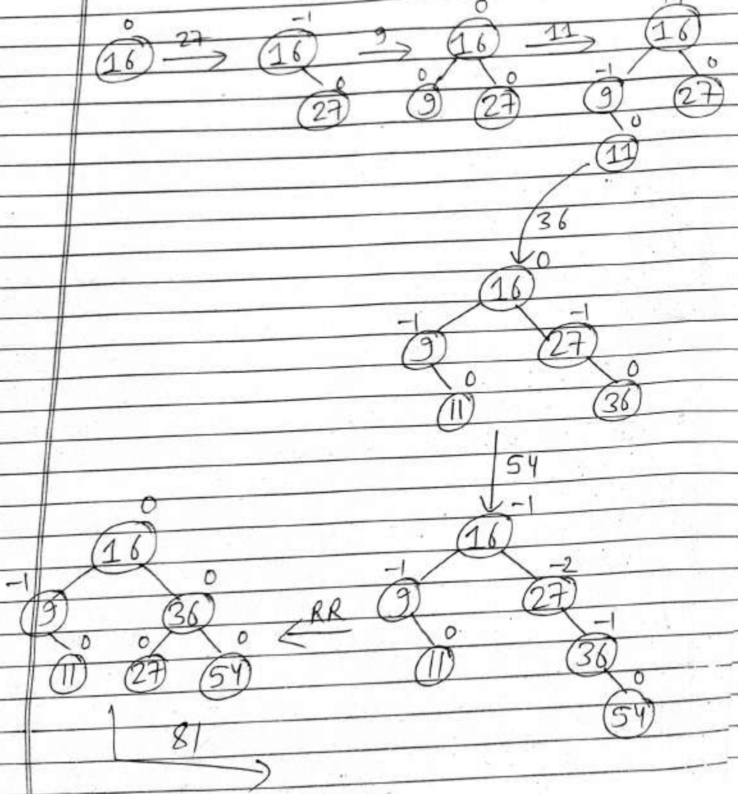
3, 2, 1, 4, 5, 6, 16, 15

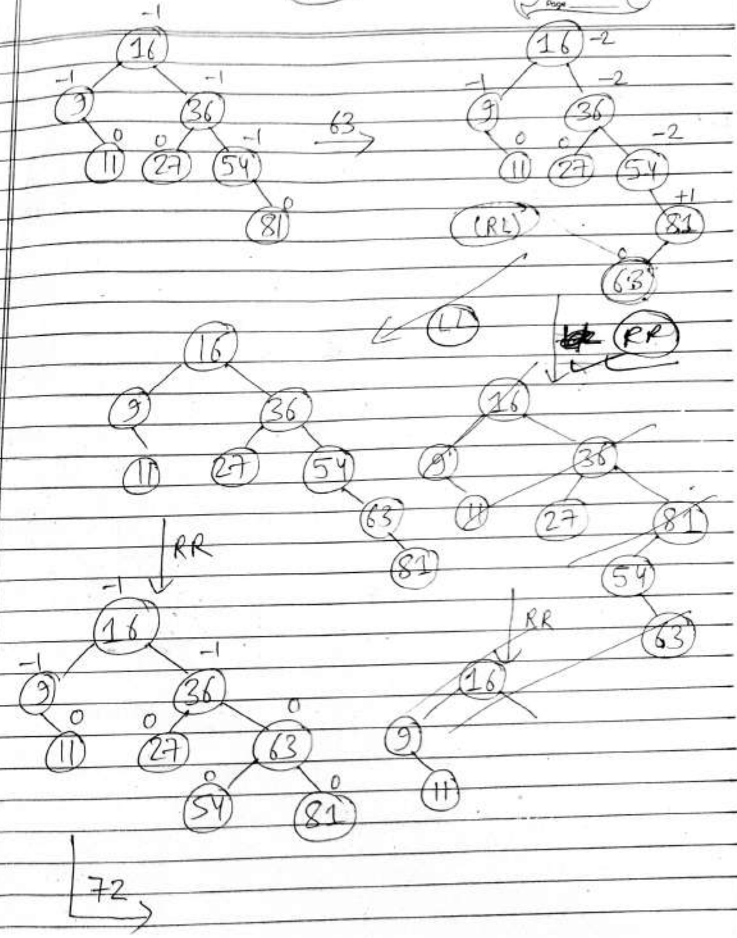




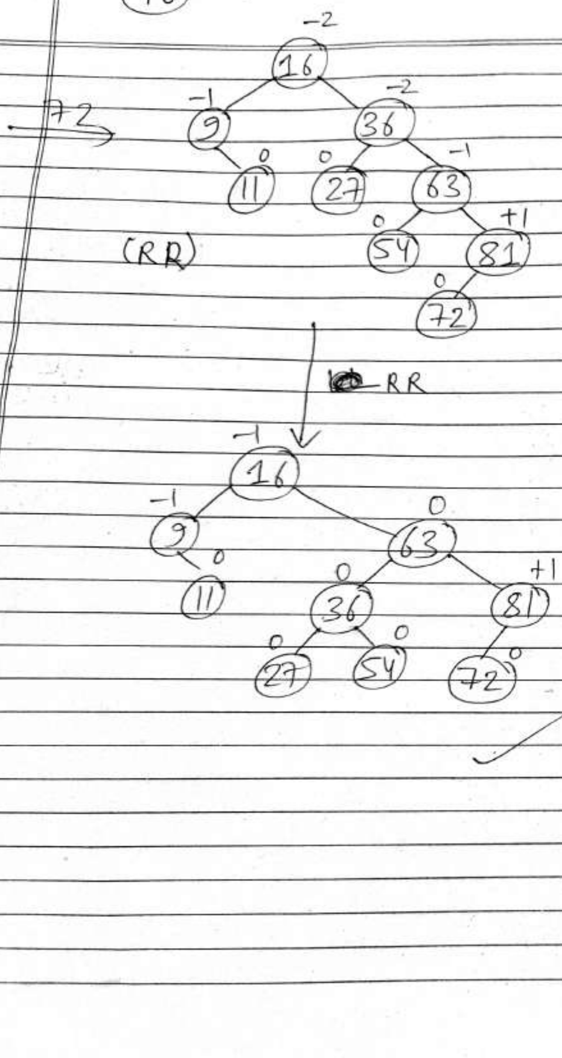
Questions: Construct AVL trees.

(1) 16, 27, 9, 11, 36, 54, 81, 63, 72





16

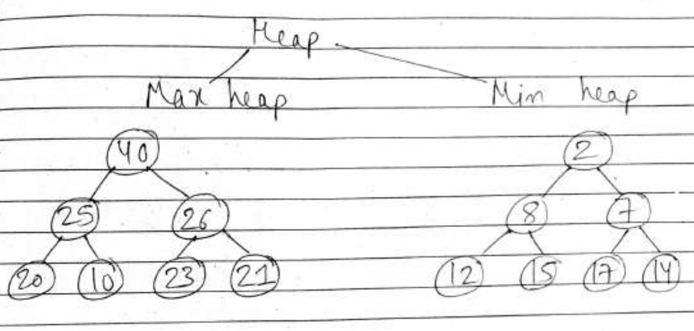


17

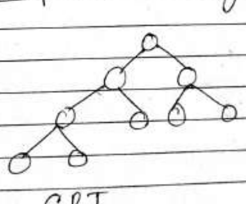
*) Heap

→ It is a complete Binary Tree. (left possible)

→ The value at node N is greater than or equal to the value at each of the children of N (max heap)

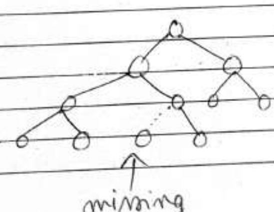


Complete Binary Tree: When new node is added, add to the left most first.

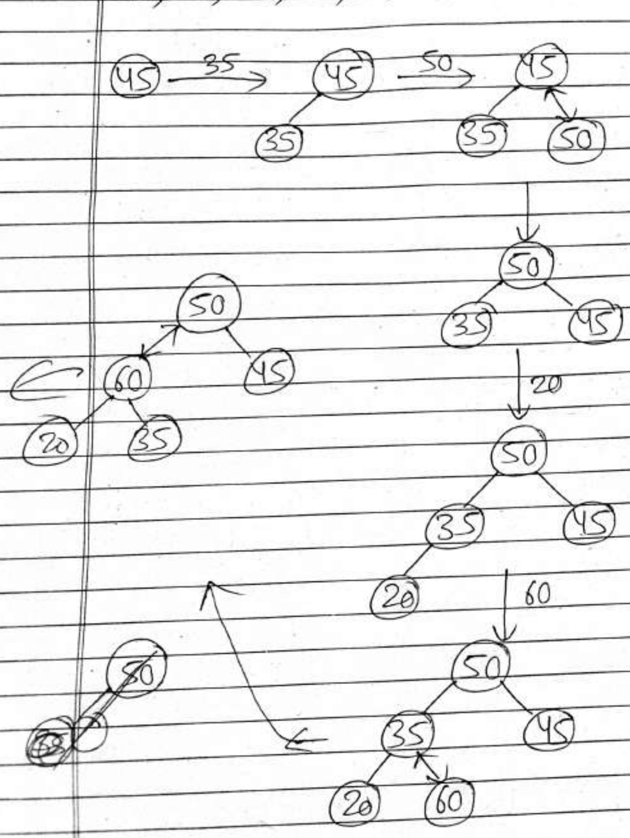


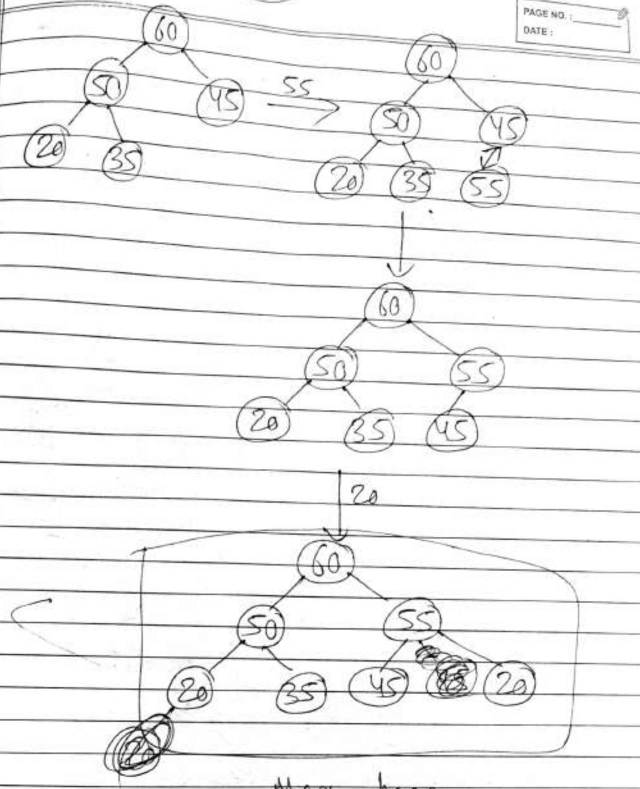
CBT

Not Complete Binary Tree:



*) Insertion in Heap: (Left most priority)
(Max heap)
45, 35, 50, 20, 60, 55, 20

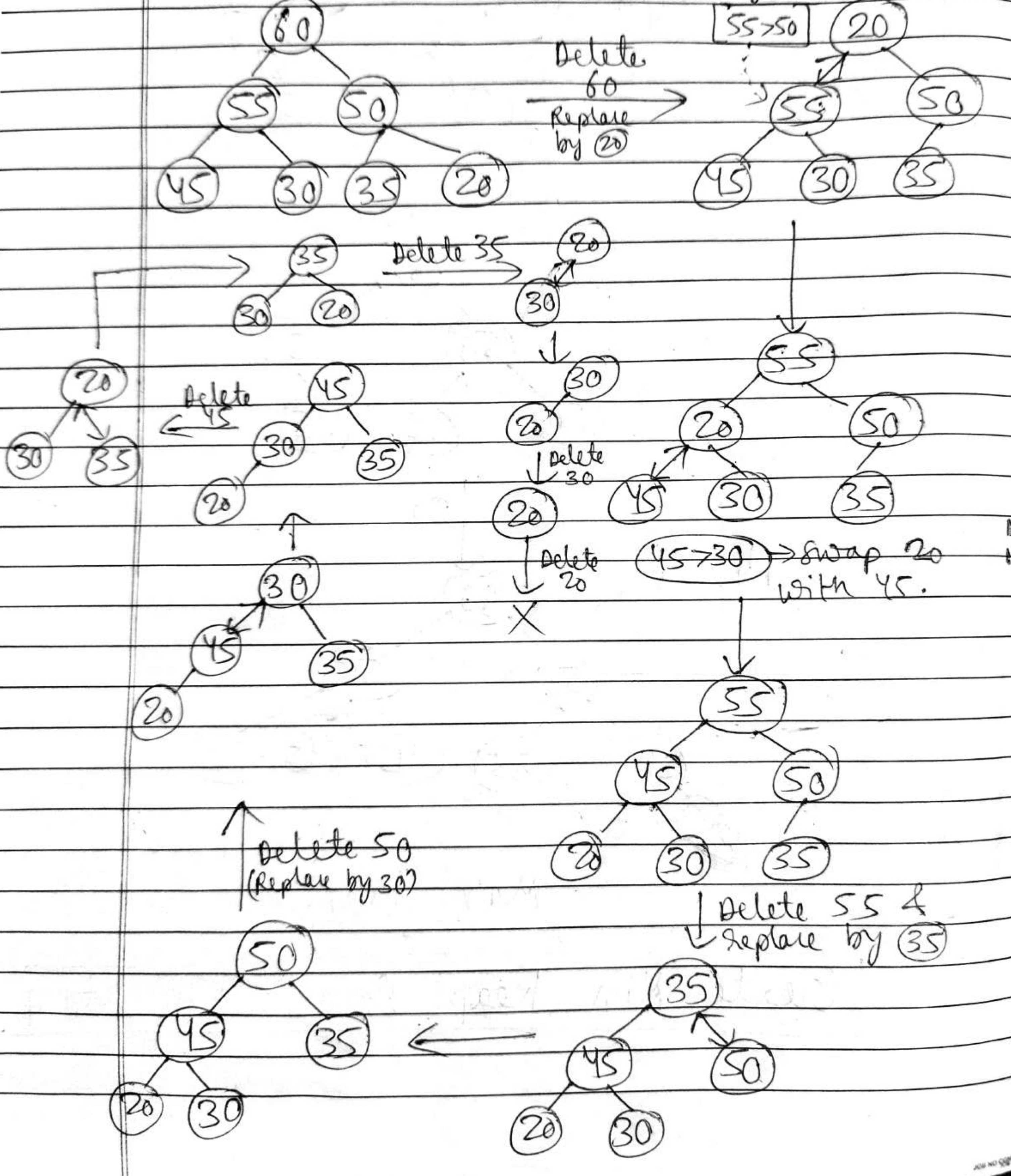




Max heap.

$\boxed{\text{Create Min heap from above data}}$

*) Deletion in heap: (Delete the topmost & replace by last element)



2080/9/4

Date
Page

v) B-Tree:

→ It is a self-balanced search tree in which every node contains multiple keys (values) & has more than two children.

→ all nodes (leaf) are at same level.
[M-order]
→ all nodes except root node have at least [m/2 - 1] keys & max. (m-1) keys. ⇒ (m/2 → ceiling value)

→ no node contains more than (m-1) values.

→ keys are arranged in defined order within the nodes. (ascending order)

*) Construct B-Tree:

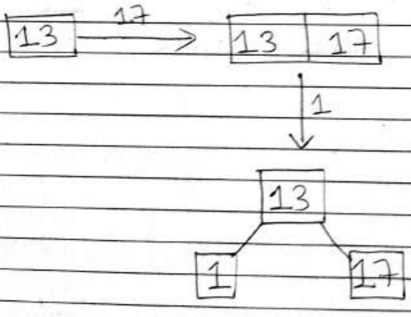
13, 17, 1, 14, 16, 23, 24

Order (m) = 3

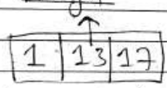
Sol^n,

min. keys (values) = m/2 − 1
= 3/2 − 1
= 2 − 1
= 1

max. keys = m − 1
= 3 − 1
= 2



Rough



(Since, leaf nodes should be at same level.)