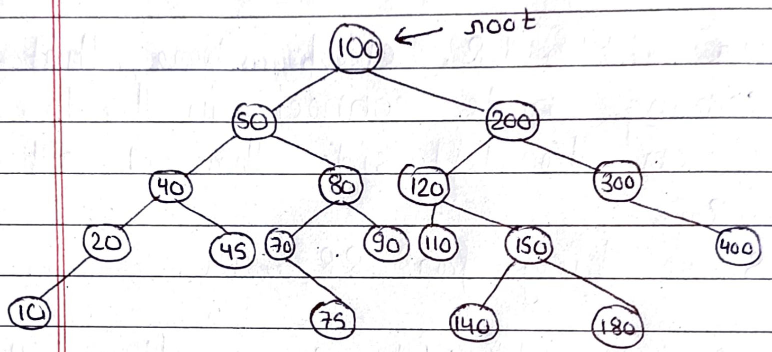
Date Page

Day - 174

AVL Tree - 2

* AVL Tree Deletion:

⇒



⇒ Suppose, we want to delete 10, then we can simply delete it & return null.

⇒ And when we are returning back, we will again check every node for Balance.

⇒ Case 1: Leaf node

⇒ simply delete it.

⇒ Case 2: Single child

→ Left child

Delete that node & returns its left child.

→ Right child

Delete that node & returns its right child

⇒ Case 3: Both child exists

Date ____________ Page ________

⇒ Right side → minimum
⇒ So, here, first, we find that node.
⇒ After that we will find min. element that will on the right subtree of that node.
⇒ Now, we will change the data of that node with min. node.
⇒ After that delete that min. node.
⇒ And check again for balance.

⇒ So, suppose, unbalancing occurs in any node.
⇒ Always, remember, when we were inserting node that time unbalancing occurs in the inserting side.
⇒ But at the time of deletion, unbalancing occurs on the other side.

⇒ We will find the side of unbalancing by previous logic.
⇒ After that we have to find in it that it will (LL or LR) or (RR or RL).
⇒ For that, we go to first child of that unbalancing node. subtract left & right height.
⇒ If result → -ve that means right side.
If result → +ve that means left side.
If result → 0 then we can do any

notation but we will prefer LL or RR because RL or LR are done in two notations.

⇒ So, it helps in reducing some time.

⇒ Time Complexity of deletion is $O(\log n)$.