Date _____ Page _____

Day - 173

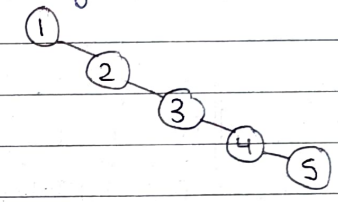
AVL Tree

* **AVL Tree:** tree.

⇉ First, we have to understand, why we need AVL

⇉ In worst case, BST T.C. for various operations are —

Insertion: O(n)
Search: O(n)
Deletion: O(n)



⇉ Creation time:
Array in advance → nlogn
Runtime → n^2

⇉ So, in AVL tree, we can do all three operations in O(log n) T.C.

⇉ AVL tree don't grow only on one side.

⇉ If any side is growing only then it is self balanced that side.

⇉ So, AVL tree is called as self balance Binary Search Tree.

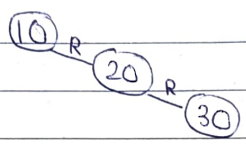
⇉ Every node should have balance between
$-1 \leq$ balance ≤ 1

Date _____ Page _____

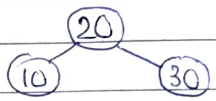
⇒ Suppose, we have three elements —
10 20 30

⇒

In BST



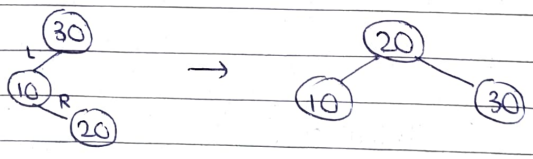
In AVL



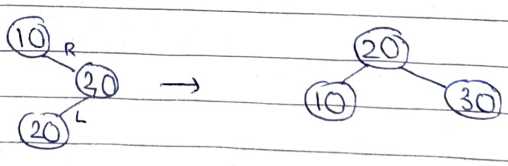
⇒ So, if we have three nodes then the permutation will be —

10	20	30	→	we have to balanced these AVL Trees.
30	20	10		
30	10	20		
10	30	20		
20	30	10	→	Balanced AVL Tree.
20	10	30		

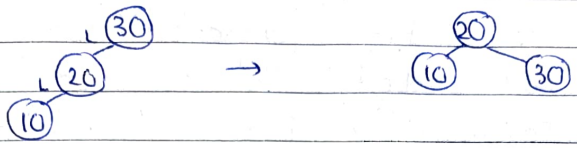
⇒



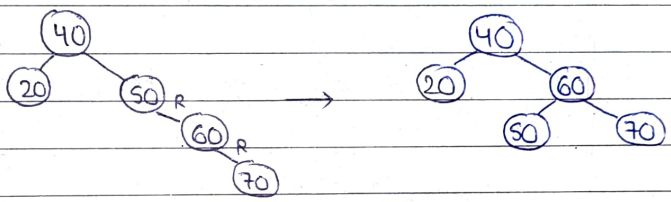
⇒



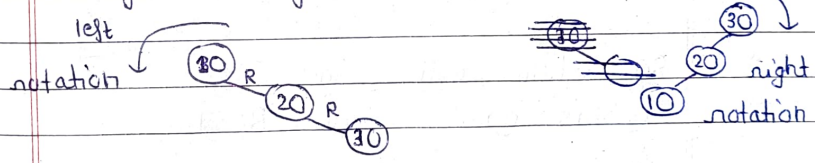
Date _____ Page _____

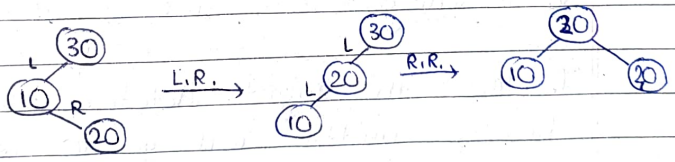
⇒ 

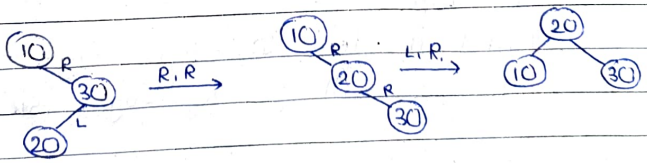
⇒ We will only have these above cases.

⇒ 

balanced ⇒ We have two rotations —
left & right rotation.



⇒ 

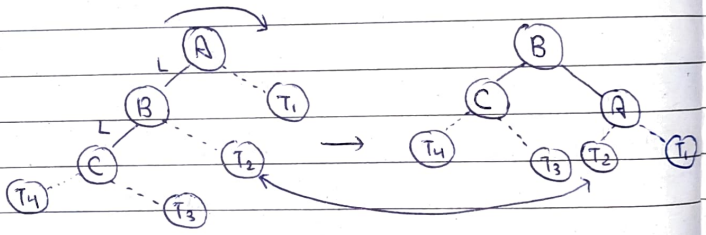


⇒ L.R. → Left Rotation
R.R. → Right Rotation

Date _____ Page _____

1. LL Case: Right Rotation (Top)
2. RR Case: Left Rotation (Top)
3. LR Case: Left Rotation (Middle)
   Right Rotation (Top)
4. RL Case: Right Rotation (Middle)
   Left Rotation (Top)

⇒ There is a edge case remains here —



⇒ So, where will T_2 go?

⇒ T_2 is greater than B but less than A.

⇒ So, it will go to left side of A.

⇒ Therefore, in right rotation, right child of middle will go to left side of top.

⇒ And in left rotation, left child of middle will go to right side of top.

⇒ So, how will be know that it is which case?

Date _____ Page _____

⇒ For this, if Balance Factor (BF) = left height - right height —

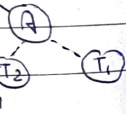
⇒ BF > 1 → Left case (LL or LR)

⇒ BF < -1 → Right Case (RR or RL)

⇒ For LL & LR, check where that coming node connects in the tree.

⇒ If on the left side then LL otherwise LR.

⇒ Same thing for RR & RL.



⇒ Calculating height at every time will increase the T.C.

⇒ So, we will create a height variable in node class.

⇒ So, when we return, we have to update the height.

⇒ Time Complexity

Creation → O(log n) * n

⇒ O(n log n)

Deletion → O(log n)

Searching → O(log n)