2021

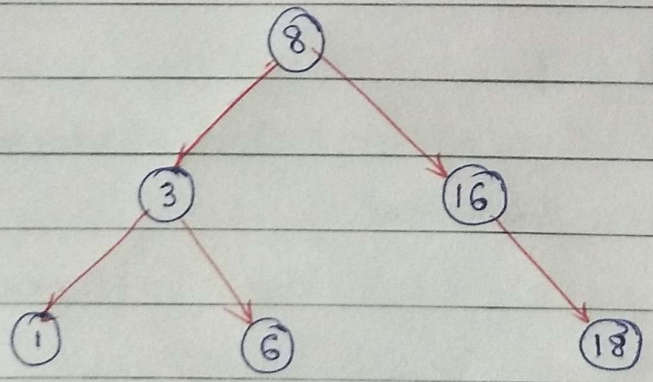
Module 2

Binary Search Tree

• BST is a node-based binary tree data structure has the following properties.

- The left subtree of a node contains only ~~one~~ nodes with keys lesser than the node's key.
- The right ~~stree~~ subtree of a node contains only nodes with keys ~~lesse~~ greater than the node's key.
- The left & right subtree each must also be a BST.

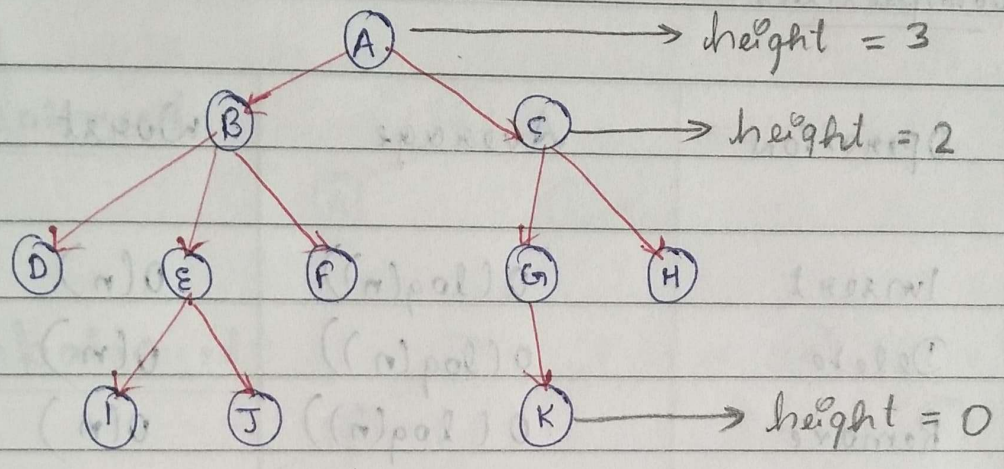
Eg:



Height of a Node & Tree

In a tree data structure, the total number of edges from leaf node to a particular node

in the longest path is called a height of that node. In a tree, height of the root node is said to be height of the tree. In a tree, height of a leaf node is '0'



Here height of tree is 3

- In any tree, height of node is total no. of edges from leaf to that node in longest path.
- In any tree, height of tree is the height of the root Node.

BALANCED BINARY SEARCH TREE

* A balanced Binary Search tree is a Self balanced BST.
* This type of tree will adjust itself in

order to maintain a low ~~weight~~ height allowing for faster operation such as insertion & deletion.

Complexities

Operation	Average	Worst
Insert	O(log(n))	O(n)
Delete	O(log(n))	O(n)
Remove	O(log(n))	O(n)
Search	O(log(n))	O(n)

- It is observed that BST's worst case performance is closest to linear search algorithms, that is O(n)
- In real time data, we cannot predict data pattern & their frequencies.
- So, a need arises to balance out the existing BST.

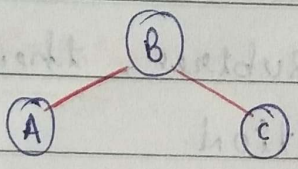
* Named after their inventor Adelson Velski & Landis, AVL ~~trees~~ trees are height balancing

Page No :
Date :

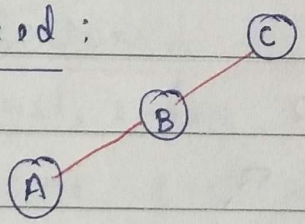
BST.

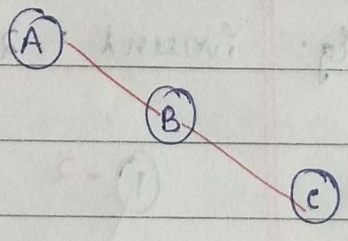
* AVL trees checks the height of the left & right sub-trees & assures that the difference is not more than 1.
* This difference is called the **Balance Factor**.

Balanced :



Unbalanced :





In the second tree, the left subtree of C has height 2 & the right subtree has height 0, so the difference is 2 in the third tree, the right subtree of A has height 2 & left is missing, so it is 0 & the difference is 2 again. AVL tree permits difference to be only 1.

Balance Factor = height(left-tree) - height(right)

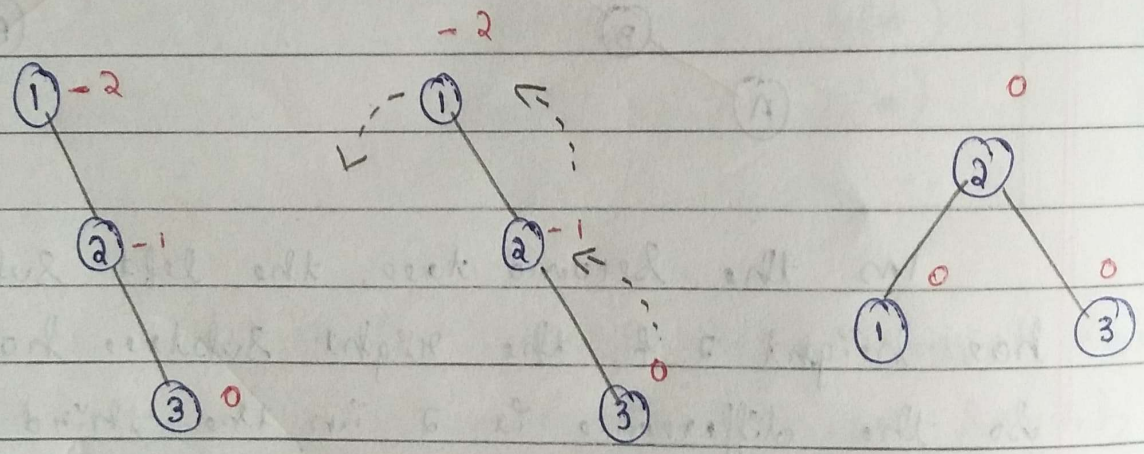
If the difference in the height of left & right sub tree is more than 1, the tree is

balanced using some rotation techniques.

1) Single Left Rotation (LL Rotation)

If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation.

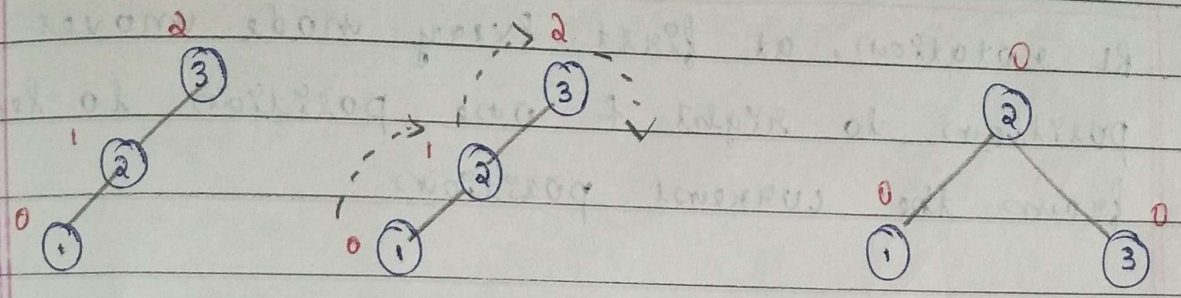
: insert 1, 2 and 3



2) Single Right Rotation (RR Rotation)

AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.

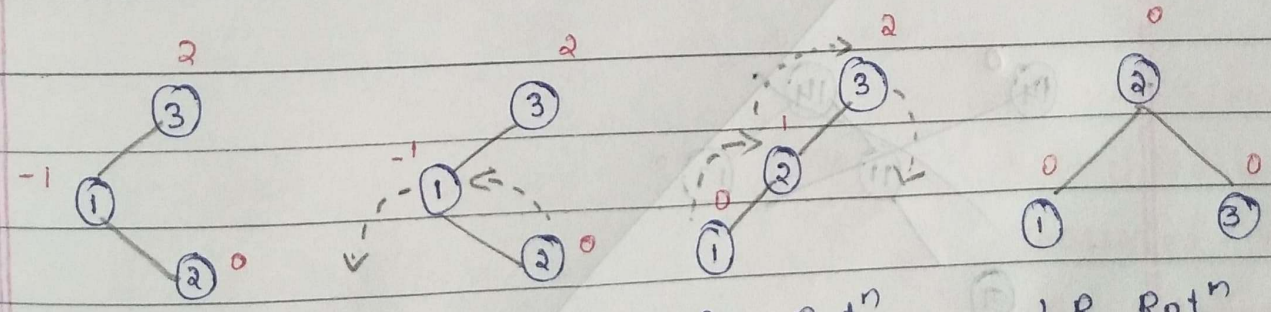
Eg: insert 3, 2 and 1



3) Left Right Rotation

A left-right rotation is a combination of left rotation followed by right rotation. In LR rotation, at first, every node moves one position to the left & one position to right from the current position. To ~~understand LR Rotation~~,

Eg: insert 3, 1 & 2



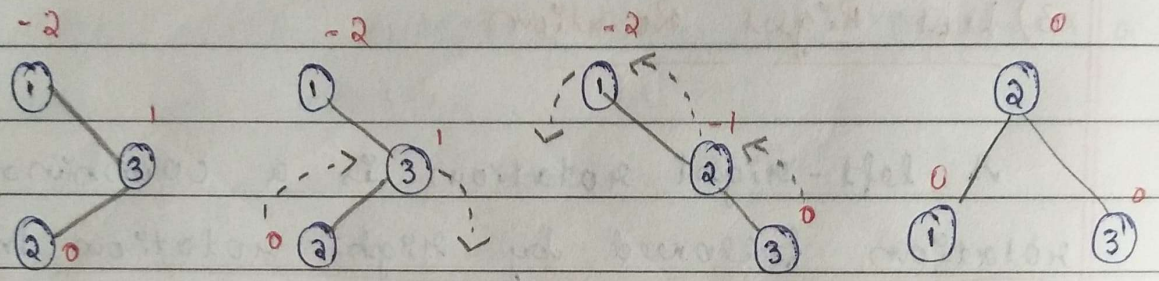
LL Rot^n RR Rot^n LR Rot^n

4) Right Left Rotation

The RL Rotation is sequence of Single right

rotation followed by Single left Rotation. In RL rotation, at first every node moves one position to right & one position to left from the current position.

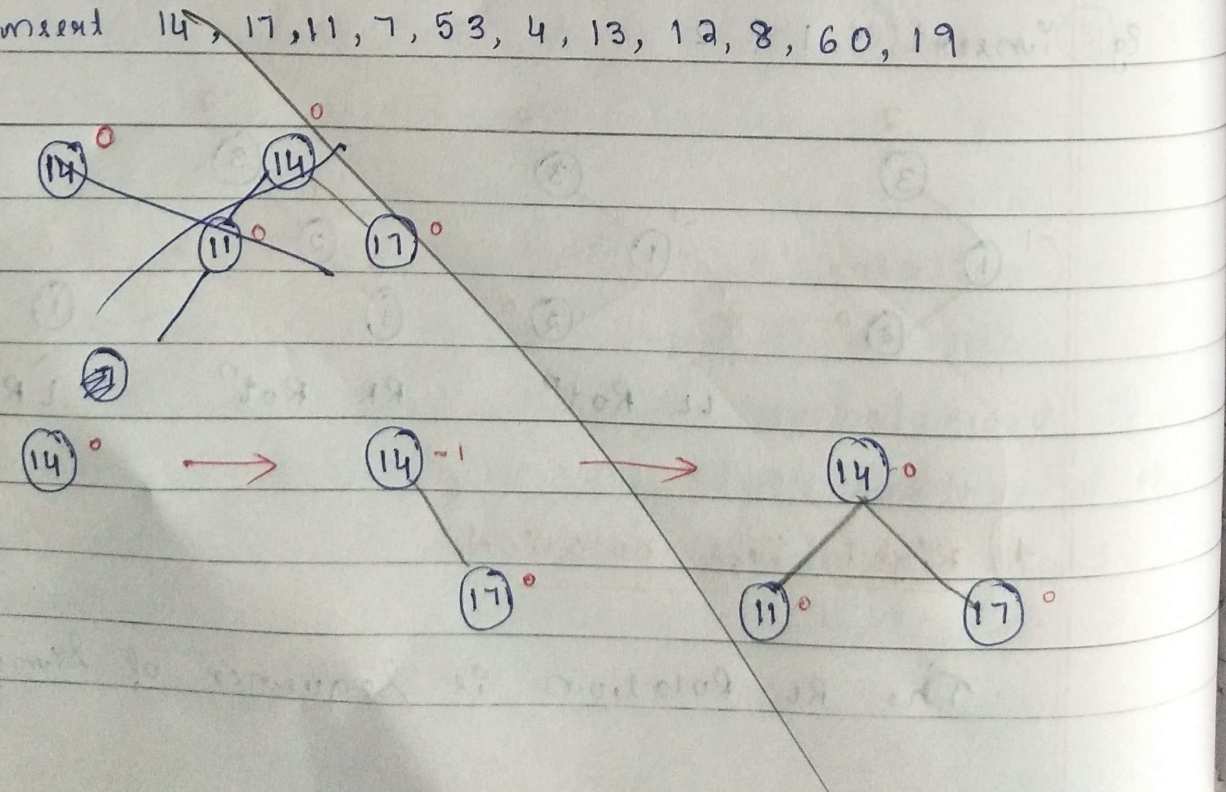
Eg: insert 1, 3 and 2



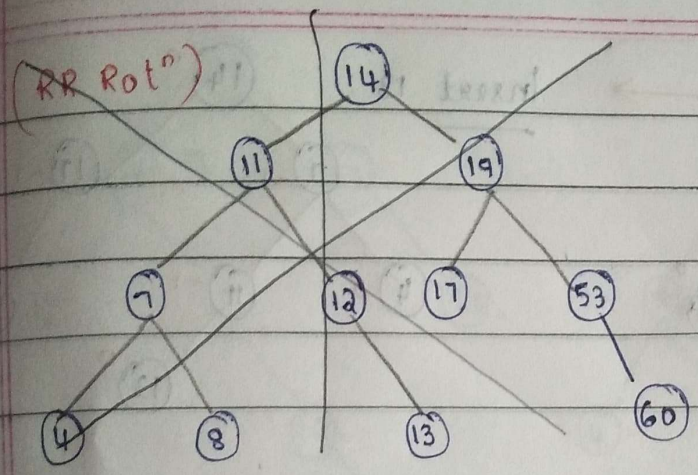
RR Rotation LL Rotation After RL

12021

Q. Insert 14, 17, 11, 7, 53, 4, 13, 12, 8, 60, 19



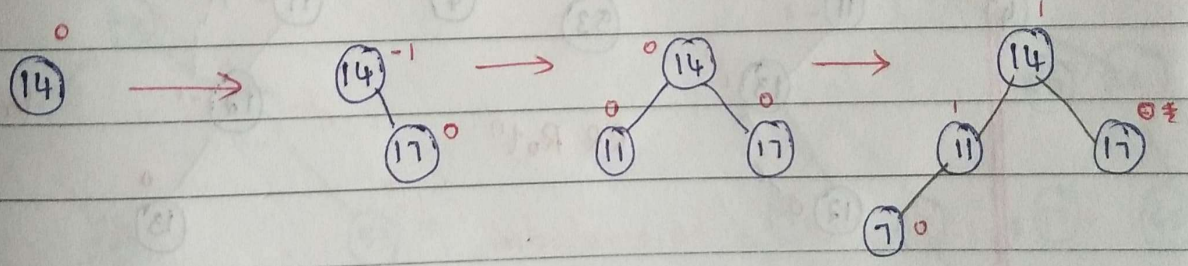
VL Simulator



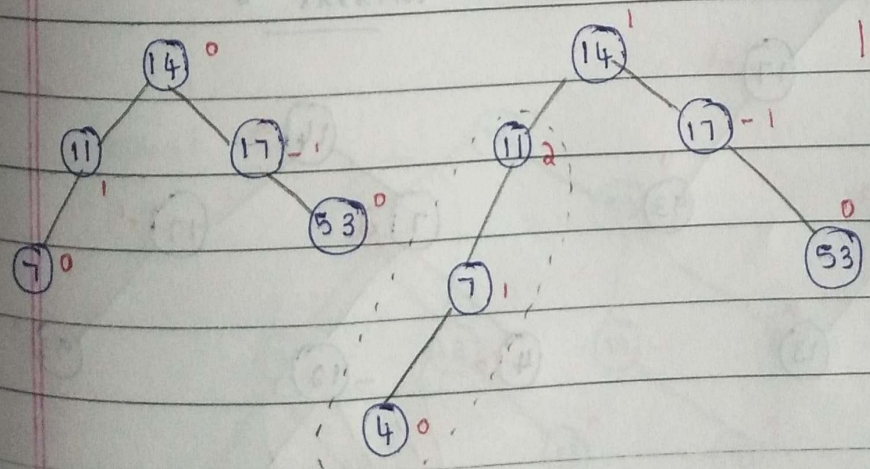
6/1/2021

Q) Insert 14, 17, 11, 7, 53, 4, 13, 12, 8, 60, 19

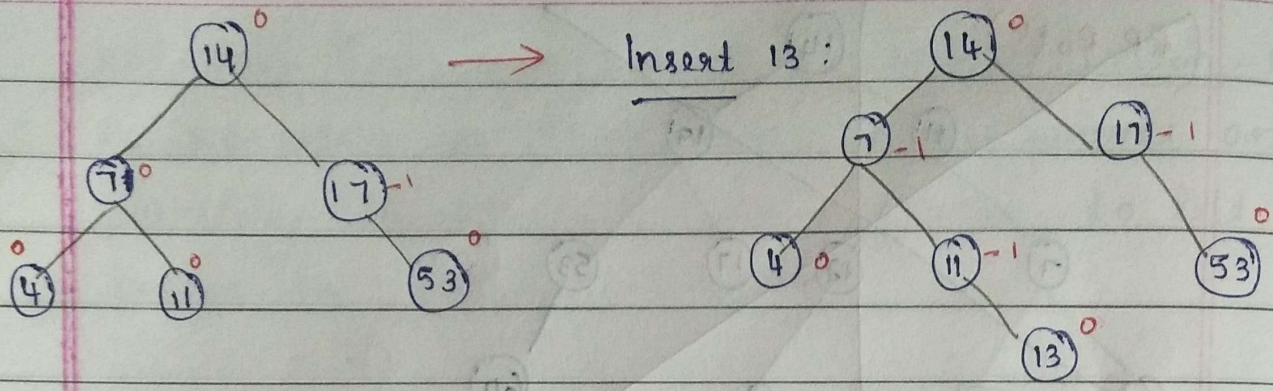
Insert 14 → Insert 17 → Insert 11 → Insert 7



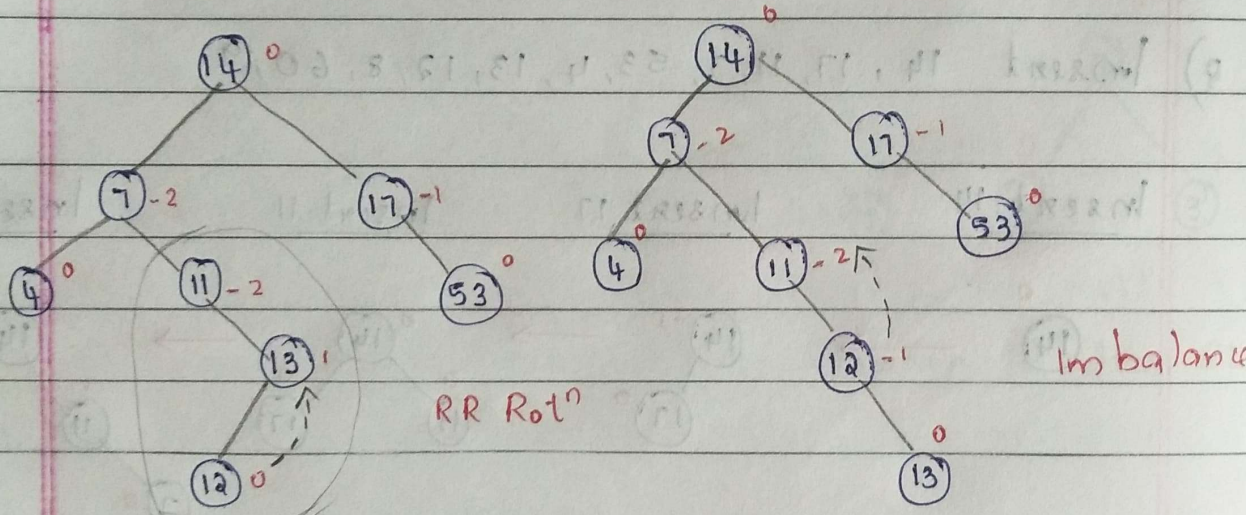
Insert 53 → Insert 4

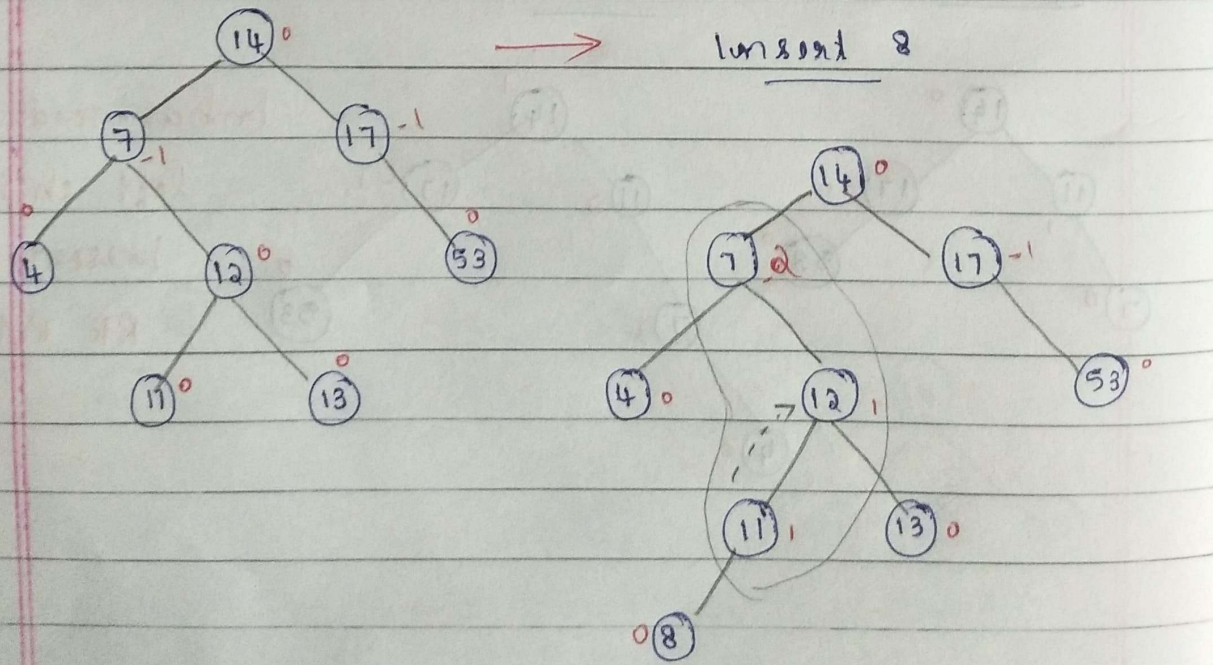


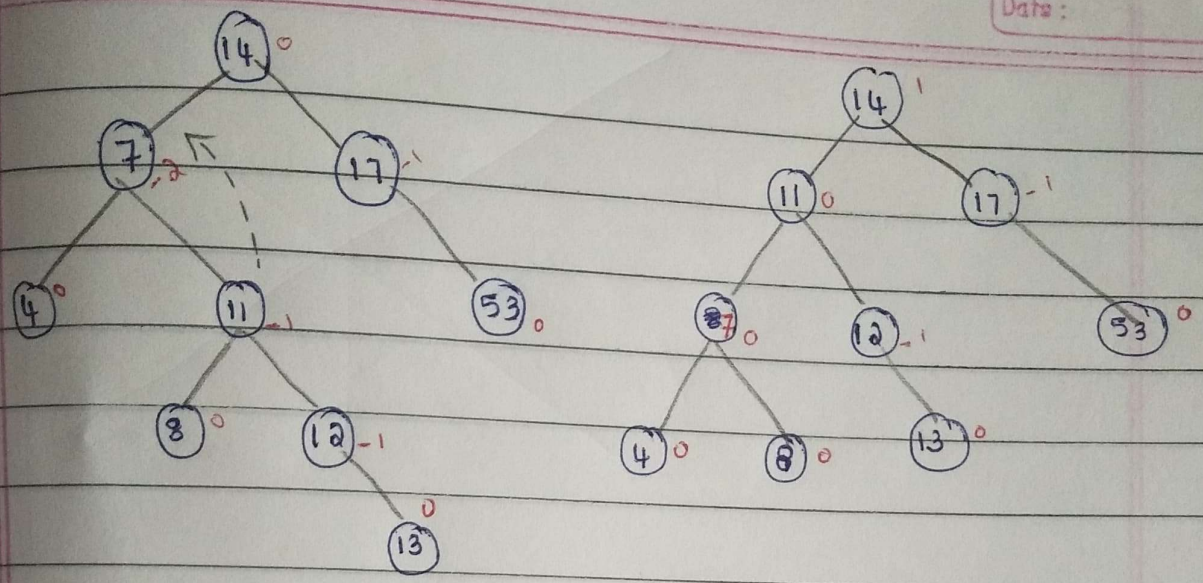
Imbalanced when left child is Inserted. so RR Rotation


14 0
7 0
17 -1
4 0
11 0
53 0
Insert 13 :
14 0
7 -1
17 -1
4 0
11 -1
53 0
13 0

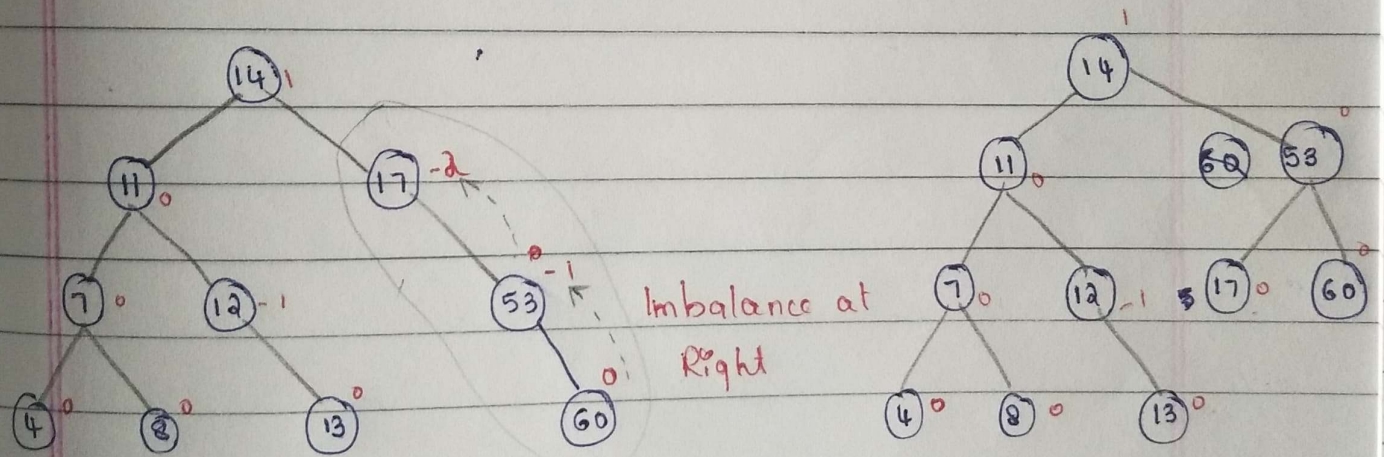
Insert 12


14 0
7 -2
17 -1
4 0
11 -2
53 0
13 -1
12 0
RR Rotn
14 0
7 -2
17 -1
4 0
11 -2
53 0
12 -1
13 0
Imbalance


14 0
7 -1
17 -1
4 0
12 0
53 0
11 0
13 0
Insert 8
14 0
7 -2
17 -1
4 0
12 1
53 0
11 1
13 0
8 0



Insert 60



Insert 19