AVL Tree

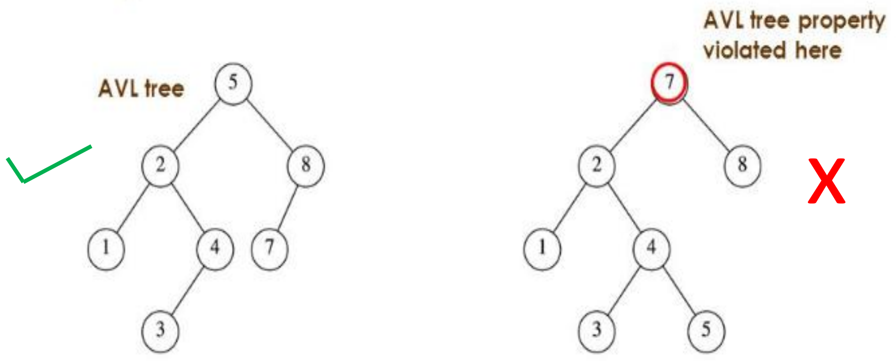
AVL tree is a binary search tree in which,

- the difference of heights of left and right sub-trees of any node is ***at most 1*** and
- the left and right sub trees are also AVL trees.

The technique of balancing the height of binary trees was developed by **A**delson, **V**elskii, and **L**andi and hence given the short form as **AVL** tree or Balanced Binary Tree.

Since AVL trees are height balance trees, operations like insertion and deletion have low time complexity.

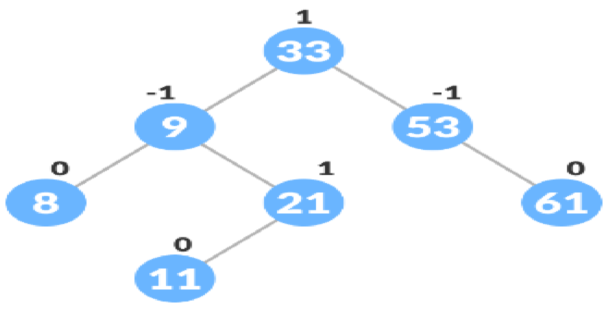
Example:



BalanceFactor `= height(left-sutree) - height(right-sutree)`

Height of a node= maximum number of edges to a leaf node
Height of an empty sub tree=-1
Height of one node=0

Example-1:



Balance factor of node 33= height of its left sub tree- height of its right sub tree= 2-1=1

Balance factor of node 9=0-1=-1

Balance factor of node 8=0, and so on

If balance factor of any node is:

1: the left sub-tree is one level higher than the right sub-tree.

0: the left sub-tree and right sub-tree contain equal height.

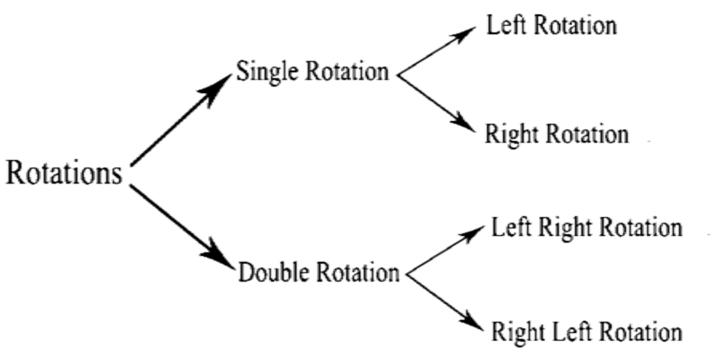
-1: the left sub-tree is one level lower than the right sub-tree.

AVL Tree Rotations:

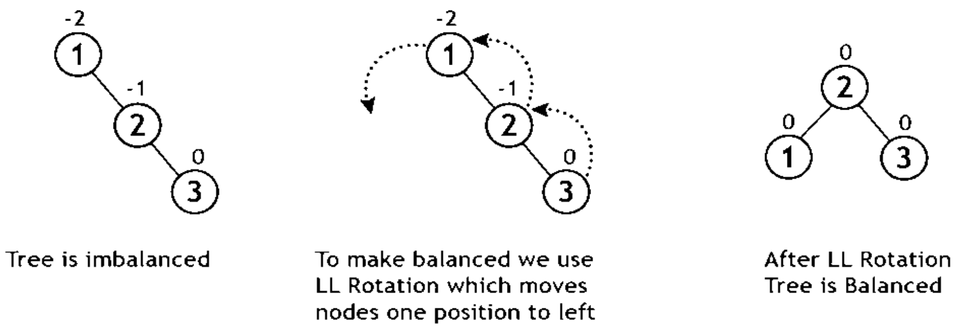
In AVL tree, after performing operations like ***insertion and deletion*** the ***balance factors*** of every node in the tree are needed to be checked.

If every node satisfies the balance factor condition then the operation is concluded.

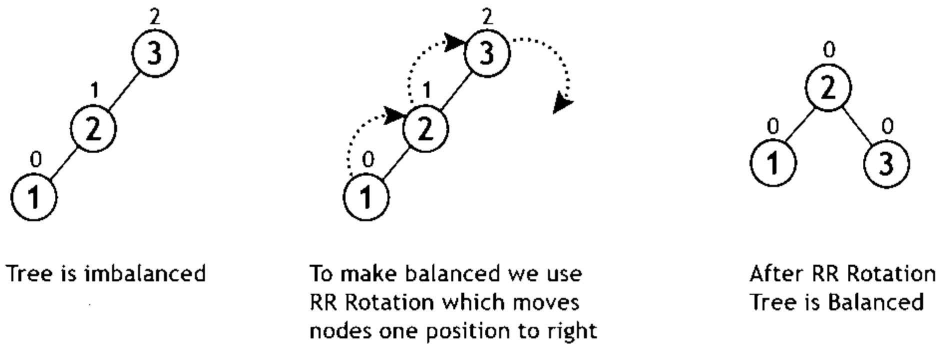
Otherwise whenever the tree becomes imbalanced due to any operation, rotation operations are applied to make the tree balanced.



1. **LL Rotation:** In LL Rotation, every node moves one position to left from the current position.
   Example: insert 1, 2, 3

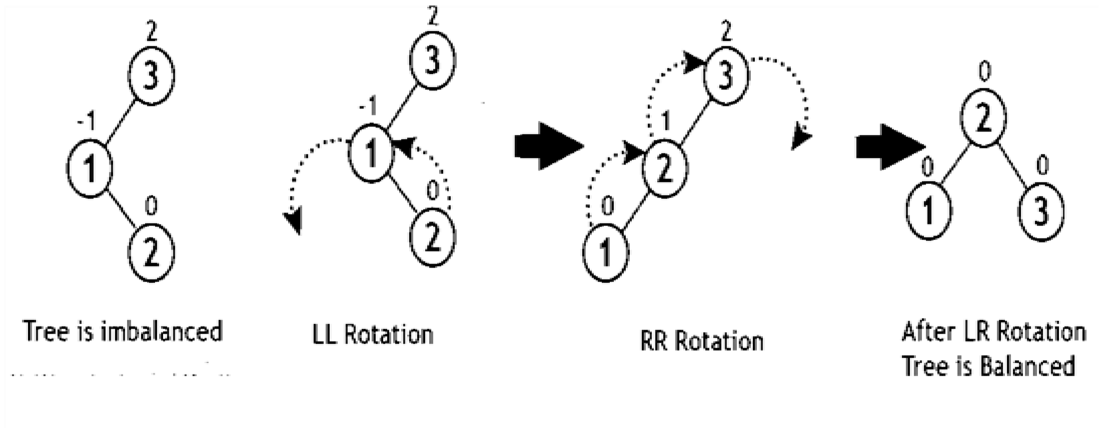

-2
1
-1
2
0
3
Tree is imbalanced
To make balanced we use LL Rotation which moves nodes one position to left
0
2
0
1
0
3
After LL Rotation Tree is Balanced

2. **RR Rotation:** In RR Rotation, every node moves one position to right from the current position.
   Example: insert 3, 2, 1

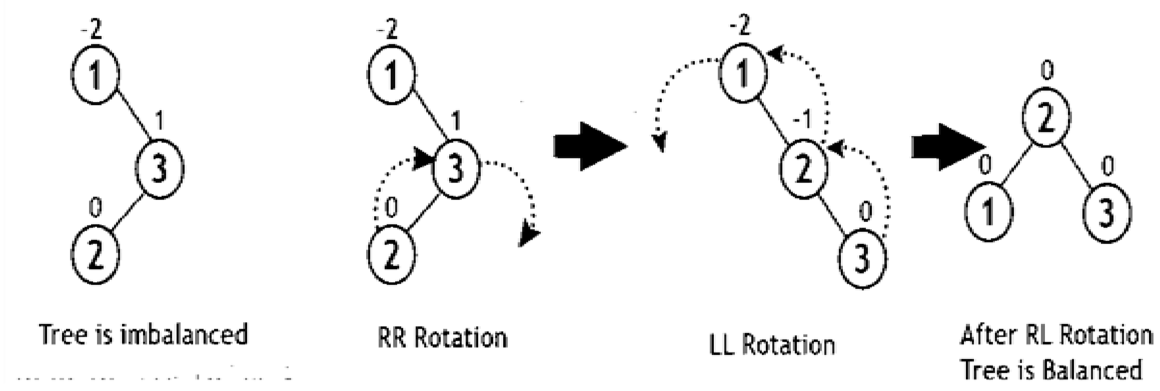

2
3
1
2
0
1
Tree is imbalanced
To make balanced we use RR Rotation which moves nodes one position to right
0
2
0
1
0
3
After RR Rotation Tree is Balanced

3. **LR Rotation:**
   The LR Rotation is a sequence of single left rotation followed by a single right rotation.
   In LR Rotation, at first, every node moves one position to the left and one position to right from the current position.

   Example**:** insert 3, 1, 2



4. **RL Rotations:** 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 and one position to left from the current position.
   Example: insert 1, 3, 2

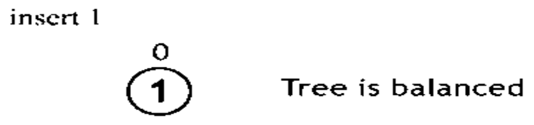


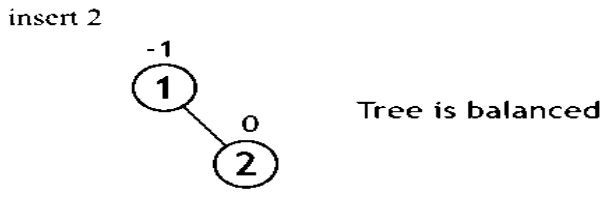
<u>Insertion in AVL tree:</u>

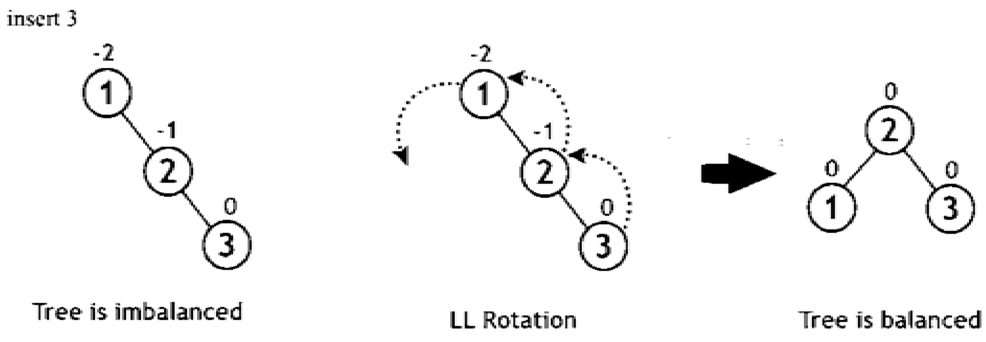
Steps:

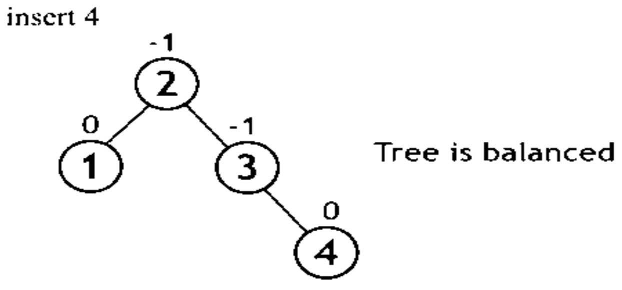
Step-1: Insert the new element into the tree using BST insertion concept.
Step-2: After insertion, check the Balance Factor of every node.
Step-3: If the Balance Factor of every node is 0 or 1 or -1 then go for next Operation.
Step-4: If the Balance Factor of any node is other than 0 or 1 or -1 then that tree is said to be imbalanced. Perform suitable Rotation to make it balanced and go for next operation.

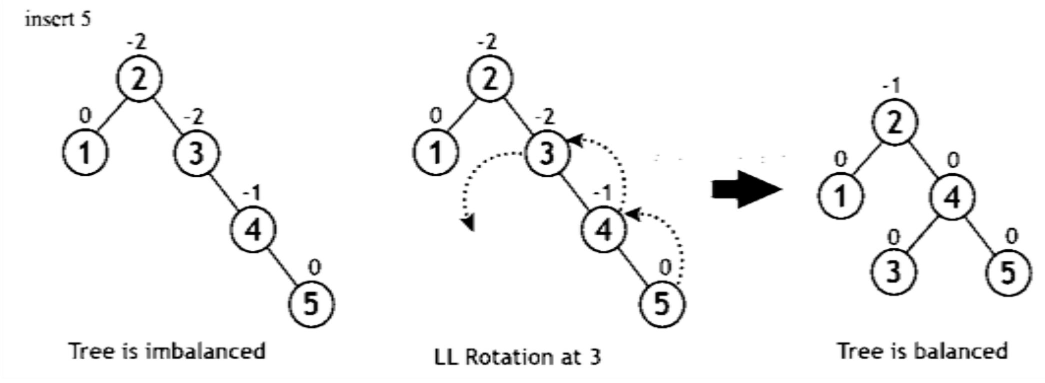
Example-1:
Construct an AVL tree by inserting 1, 2, 3, 4, 5, 6, 7, 8

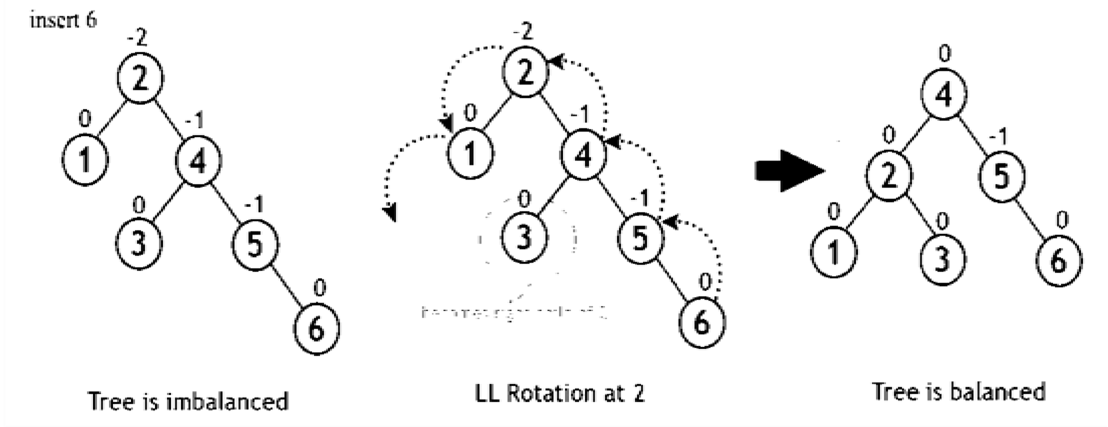


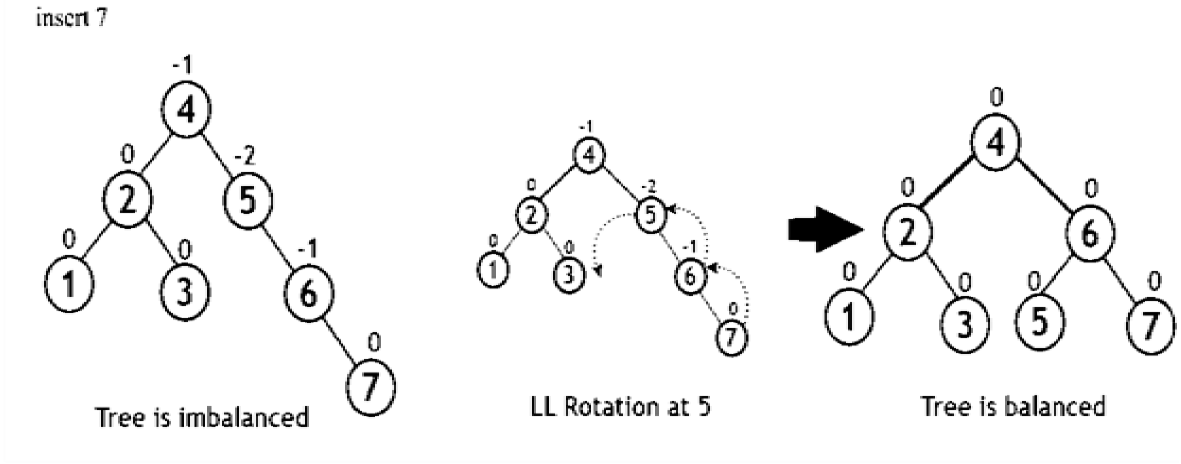


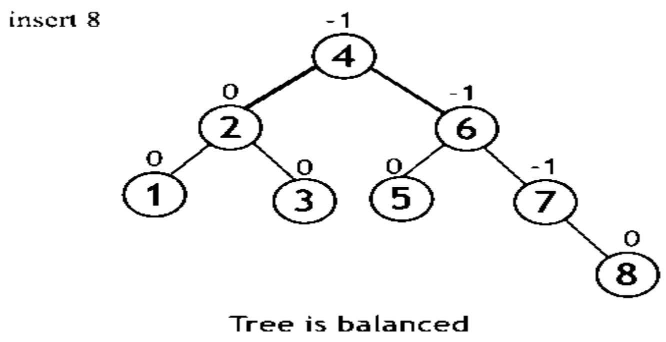






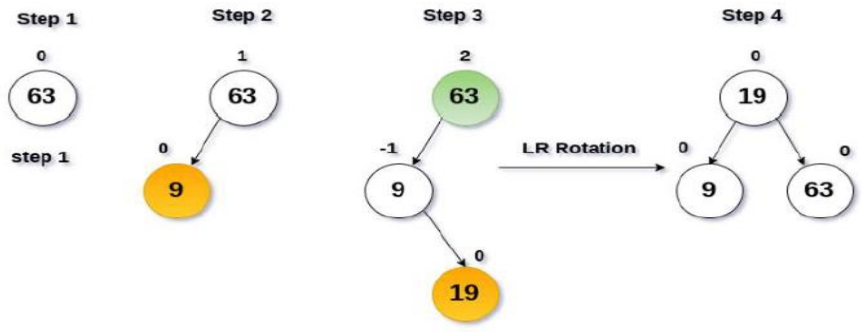


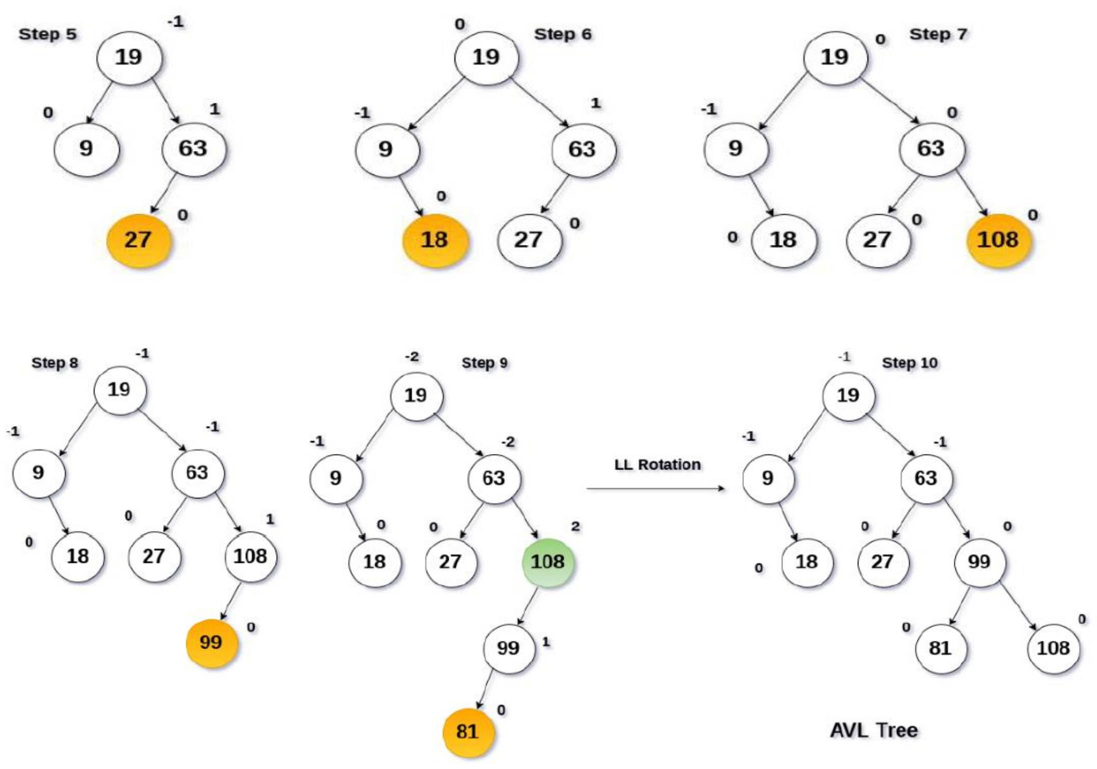




Example-2: Construct an AVL tree by inserting the following elements in the given order: 63, 9, 19, 27, 18, 108, 99, 81

Ans:

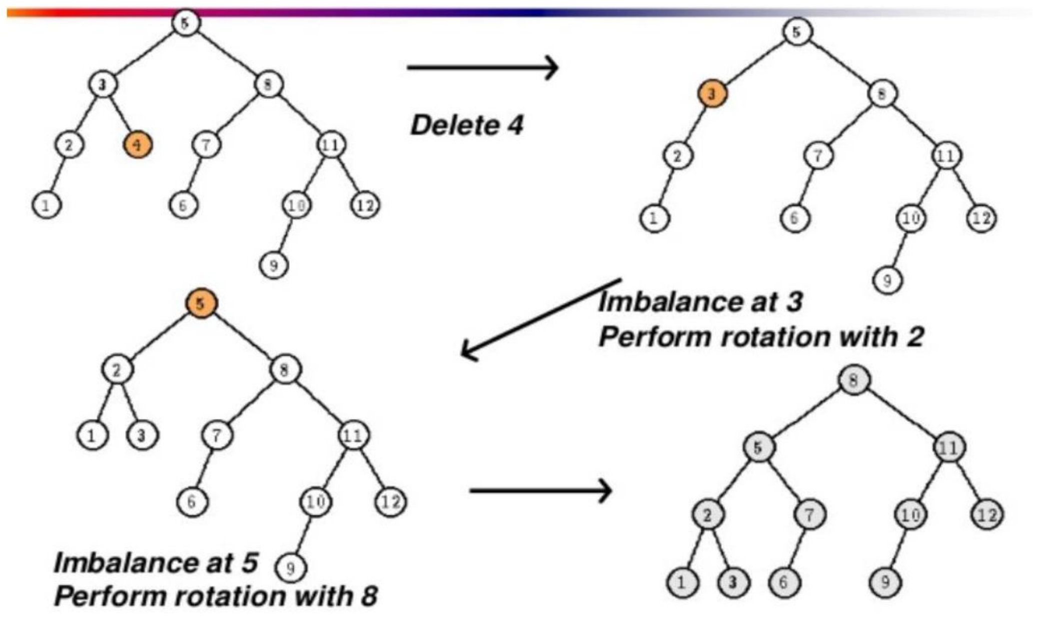




<u>Deletion Operation in AVL Tree:</u>

The deletion operation in AVL Tree is similar to deletion operation in BST. But after every deletion operation, the Balance Factor conditions are needed to be checked. If the tree is balanced after deletion go for next operation otherwise perform suitable rotation to make the tree Balanced.

Example:


Delete 4
Imbalance at 3
Perform rotation with 2
Imbalance at 5
Perform rotation with 8