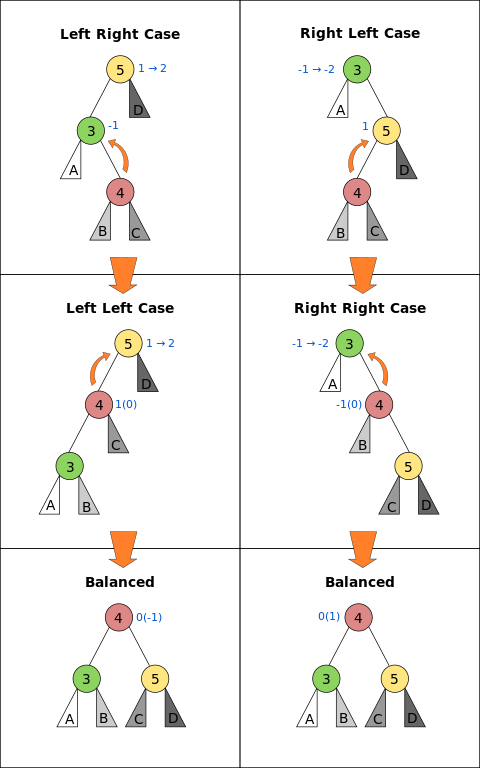
**AVL Tree Algorithm**

AVL tree is widely known as **self-balancing binary search tree**. It is named after its creator **(Georgy Adelson-Velsky and Landis’ tree).** In AVL Tree, the heights of child subtrees at any node differ by at most 1. At anytime if height difference becomes greater than 1 then tree balancing is done to restore its property.

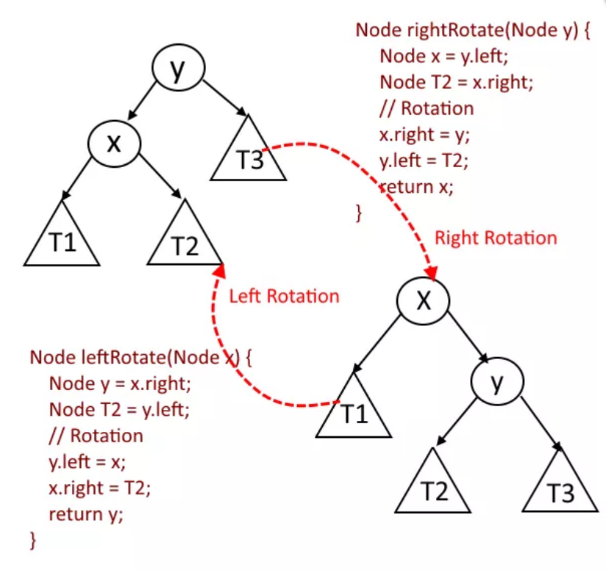
Search, Insertion and deletion, all operations takes O(logn) time since the tree is balanced.

**Why AVL Tree is better than normal Binary Search Tree:**

Average time complexity in binary search tree for any operation takes O(logn) time but there are times when your tree is skewed. In those cases the operations on them takes O(n) time but in AVL Tree, since it is always balanced, it always takes O(logn) time.

**How Tree balanced itself:**

There are two basic operations, using which tree balanced itself.

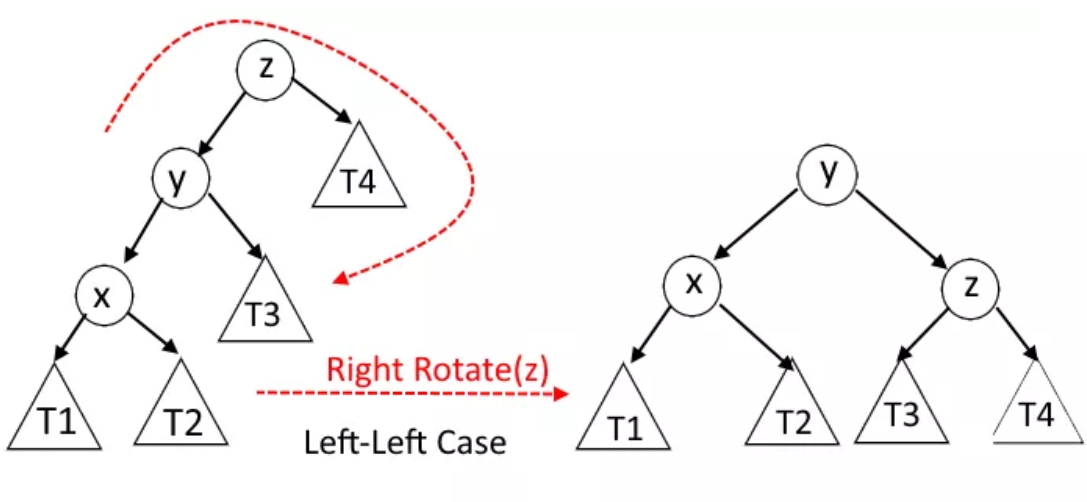
* Left Rotation
* Right Rotation

**How These Rotation works to balance the tree:**

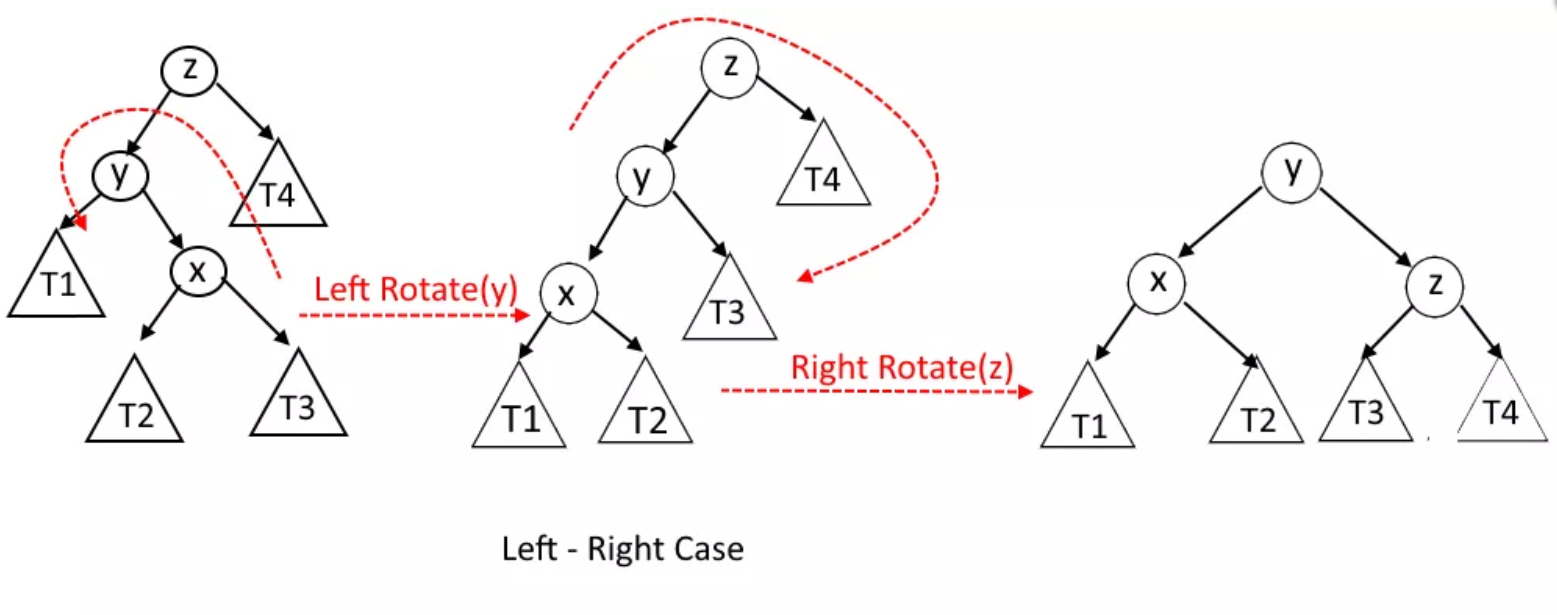
* Let ‘A’ be the new node added to the tree.
* Once ‘A’ is added, start travelling up from ‘A’ to root and find the unbalanced node, balance it and again keep traveling up.
* Say you have found the node z which is unbalanced.
* Node y is the child of z and x be the grandchild of z.

**Then there will be four possibilities:**

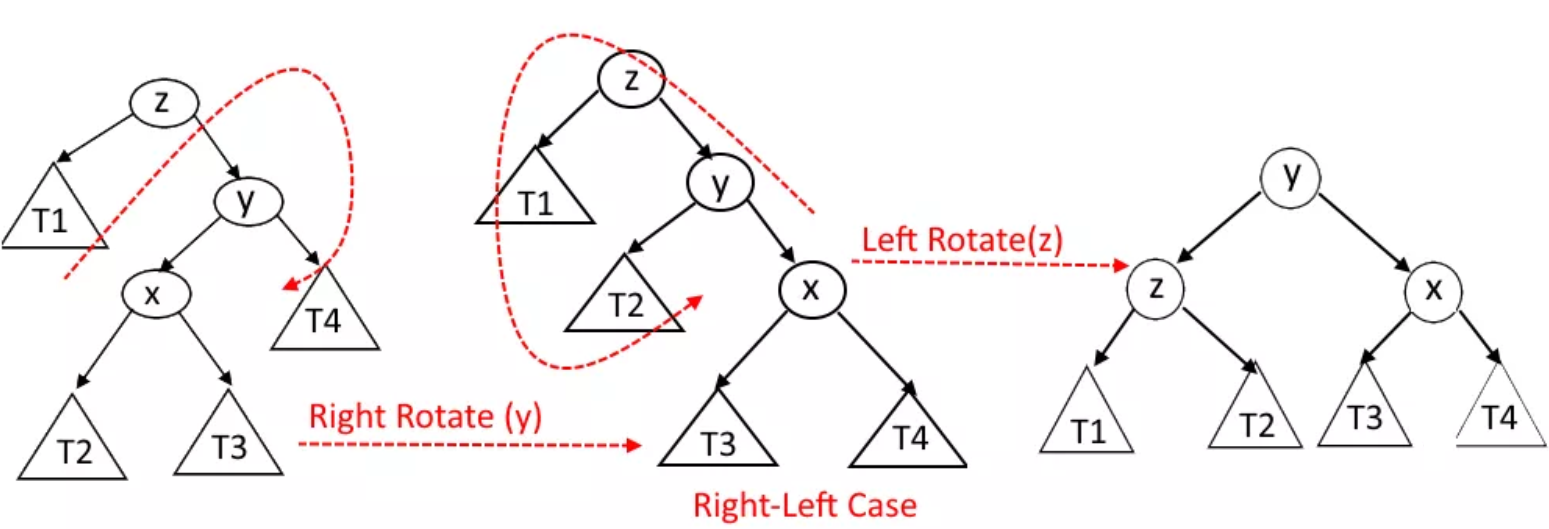
1. **Left-Left Case:** – x is the left child of y and y is the left child of z.
2. **Left-Right Case:** – x is the right child of y and y is the left child of z.
3. **Right-Left Case:** – x is the left child of y and y is the right child of z.
4. **Right-Right Case :** – x is the right child of y and y is the right child of z

**Left-Left Case :** x is the left child of y and y is the left child of z.

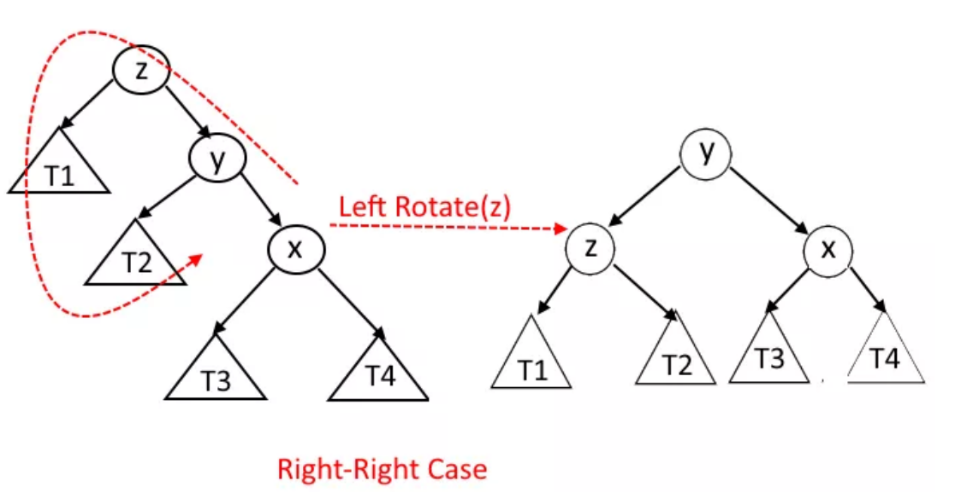
**Left-Right Case :**  x is the right child of y and y is the left child of z.



**Right-Left Case :** x is the left child of y and y is the right child of z.

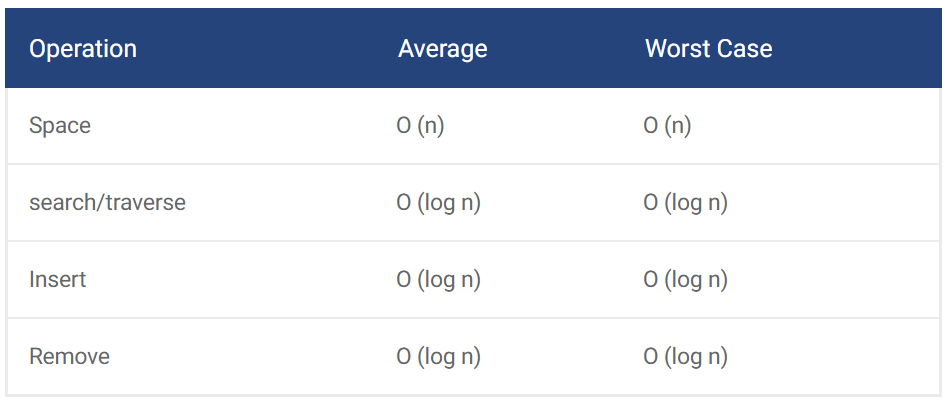


**Right-Right Case :** x is the right child of y and y is the right child of z.



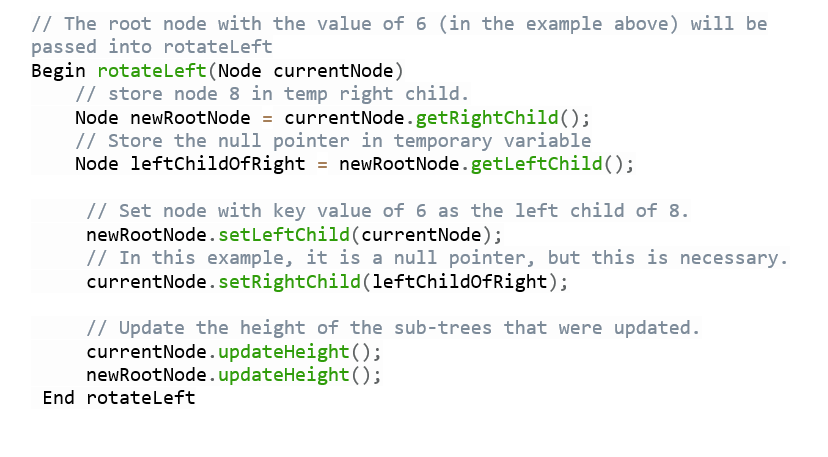
**Insertion Operation:**

1. Insert the new Node using recursion so while back tracking you will all the parents nodes to check whether they are still balanced or not.
2. Every node has a field called height with default value as 1.
3. When new node is added, its parent’s node height get increased by 1.
4. So as mentioned in step 1, every ancestors height will get updated while back tracking to the root.
5. At every node the balance factor will also be checked. balance factor = (height of left Subtree – height of right Subtree).
6. If balance factor =1 means tree is balanced at that node.
7. If balance factor >1 means tree is not balanced at that node, left height is more that the right height so that means we need rotation. (Either Left-Left Case or Left-Right Case).
8. Say the current node which we are checking is X and If new node is less than the X.left then it will be Left-Left case, and if new node is greater than the X.left then it will be Left-Right case. see the pictures above.
9. If balance factor <-1 means tree is not balanced at that node, right height is more that the left height so that means we need rotation. (Either Right-Right Case or Right- Left Case)
10. Say the current node which we are checking is X and If new node is less than the X.right then it will be Right-Right case, and if new node is greater than the X.right then it will be Right-Left case. see the pictures above.

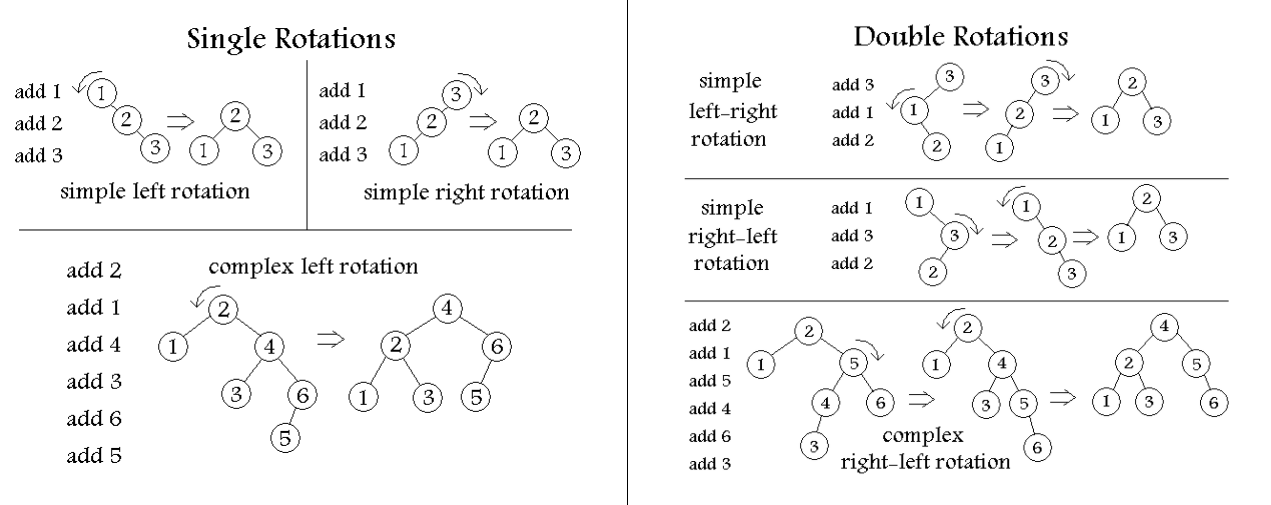
**Performance Ratings:**

Because it is self-balancing, the performance of the AVL tree is generally more consistent than that of a BST. As you can see, the worst case performance for the major operations are O (log n). Therefore, the AVL tree is a very consistent data structure, which performs well, especially when used to look up data. However, since it has to check and the rotation operations are performed frequently, in contrast to the red black tree, its performance suffers when inserting/removing data on a regular basis. For your information, the red black tree is also another self-balancing binary search tree.

**Pseudo Code:**



**Summary:**

****

