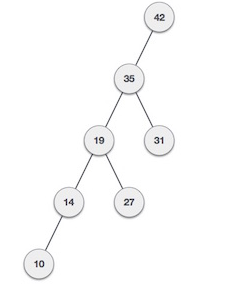
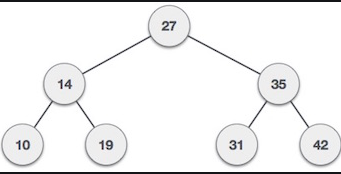
# AVL Tree:

# Why AVL tree?

Avoid dis-balanced tree to improve the performance of different operation. By using rotation.

Example:

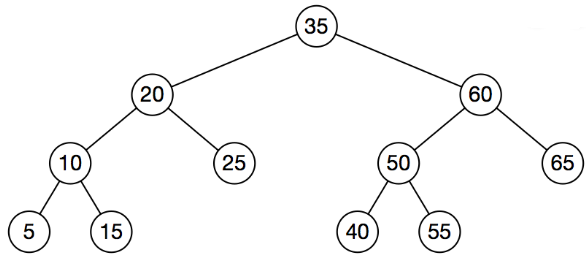




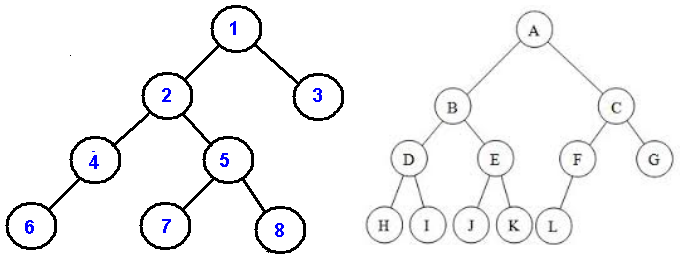
# What is AVL Tree?

AVL tree is a balanced Binary search tree follows all the property of Binary search tree.

AVL tree makes sure that the height of the siblings or Subtree will not differ more than one.



Few more Examples:



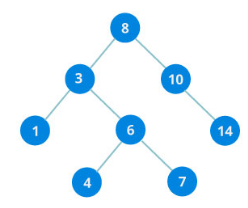
# Creation and Searching of AVL Tree

## Creation

CreateAVL()

Create root with null

**Time Complexity: O(1); Space Complexity: O(1)**



## Search in BST:

SearchAVL(root, value)

If Root = null

Return Error

Else if root = value

Print root

Else if root < value

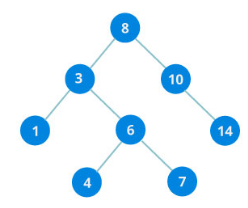
SearchAVL(root.left, value)

Else if root > value

SearchAVL(root.right , value)

**Time Complexity: O( log n); Space Complexity: O(log n)**

# Traverse AVL tree:



Logic: Pre, post, In Order ; Also Level order.

Preorder(root)

If root=null

Return error

Else

Print root

Preorder(root.left)

Preorder(root.right)

LevelOrder(root)

Create enqueuer(Q)

Enqueue (root)

While queue is not empty

Enqueuer() # Child of first element

Dequeuer() and print

**Time Complexity: O(n); Space Complexity: O(n)**

# Insert in AVL tree using LL

All the insertion properties will be similar to the Binary search tree.

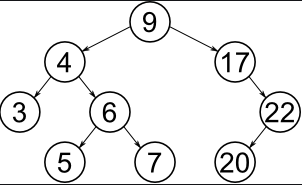
**Condition:** When the height violation happens.

**Case 1**: Rotation is not required

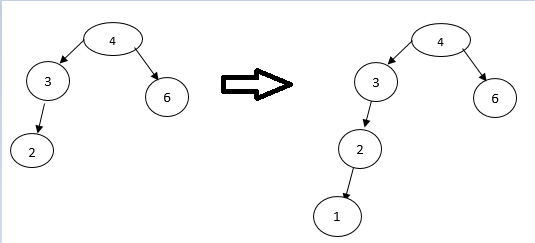
**Case 2:** Rotation is required (LL,LR,RR,RL)

## Types of rotation condition:

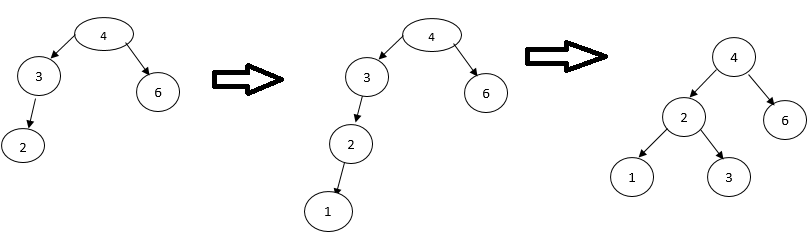
* Left left condition
* Left right condition
* Right Left condition
* Right right condition



## Left- Left condition



For a Left – left condition, we need to perform a right rotation.



# Left-Left Algorithm:

RightRotate(disbalance)

NRoot= disbalance.left

Nroot.left = Disbalance.left.left

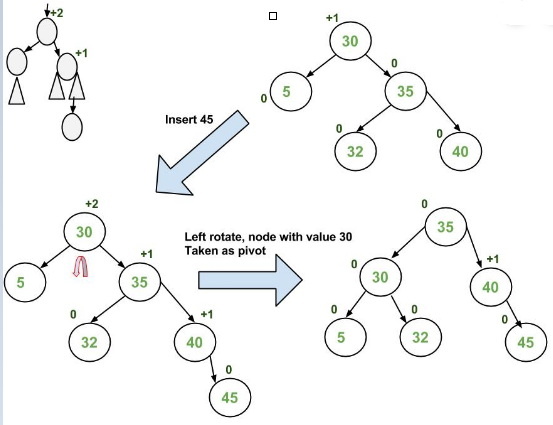
Nroot.right= disbalance

Disbalance.height= calculateHeight(disbalance)

Nroot.height = calculateHeight(Nroot)

**Time Complexity : O (1) Space Complexity : O (1)**

# Right-Right Rotation:



Leftrotate (disbalance)

Nroot=disbalance.right

Disbalance.right= disbalance.right.left

Nroot.left= disbalance

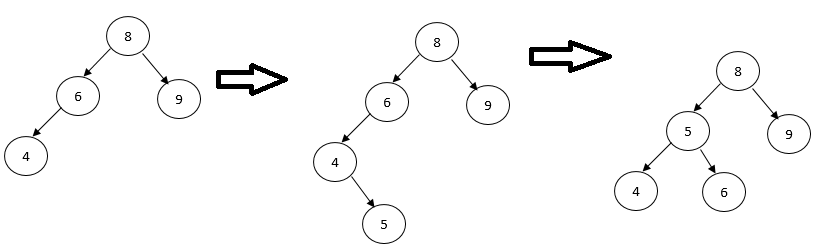
Disbalance.height = CalculateHeight (disbalance)

Nroot.height = CalculateHeight(Nroot)

Return Nroot

**Time Complexity: O (1) Space Complexity : O (1)**

# Left-Right Rotation



Leftrotate (disbalance.left)

Nroot=disbalance.left.right

Disbalance.right= disbalance.left.right.left

Nroot.left= disbalance.left

Disbalance.left.height = CalculateHeight (disbalance.left)

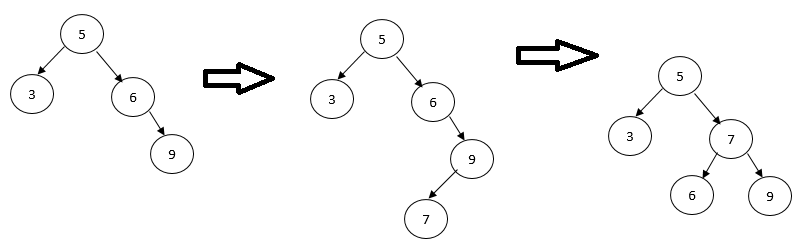
Nroot.height = CalculateHeight(Nroot)

Return Nroot

Right Rotate(disbalance)

**Time Complexity: O (1) Space Complexity : O (1)** .

# Right-Left Rotation



RightRotate (disbalance.right)

NRoot= disbalance.right.left

NRoot.left =disbalance.right

Nroot.right = Disbalance.right

Disbalance.height= calculateHeight(disbalance)

Nroot.height = calculateHeight(Nroot)

Return Nroot

# Delete using LL, LR, RL, RR

Deletion can be made in 3 cases:

1. When free node does not exist
2. When rotation is not required
3. When rotation is required

