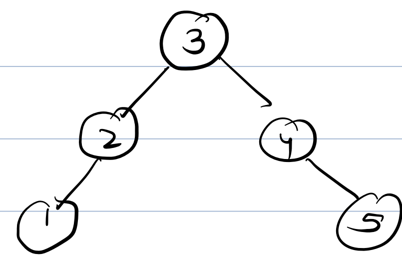
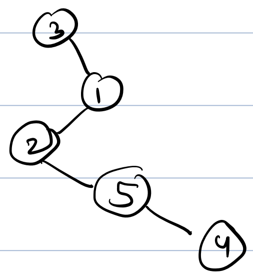
Binary Search Trees (BST)

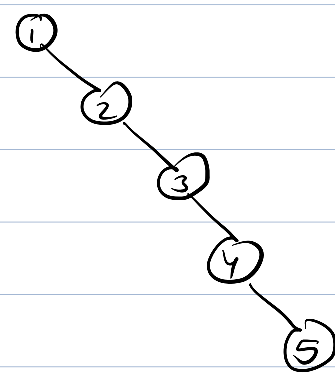
Search :



BT



Search O(N)



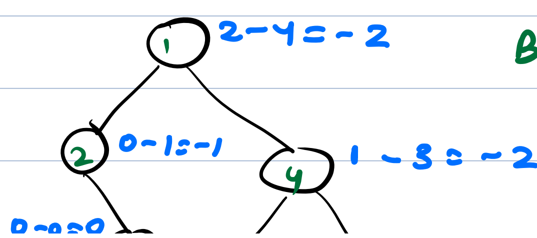
Search : O(H)

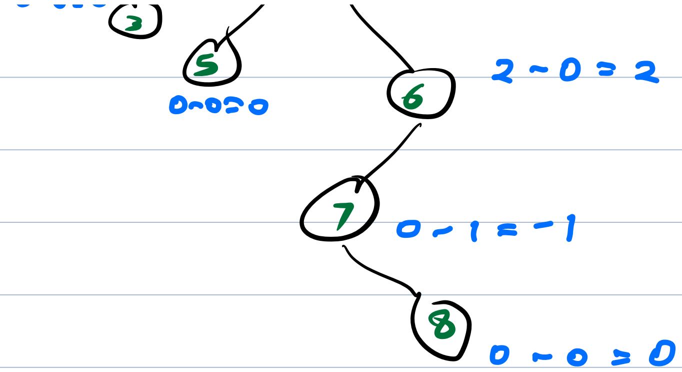
AVL Tree

It is a self balancing tree. It is always maintain the balance factor of the nodes.

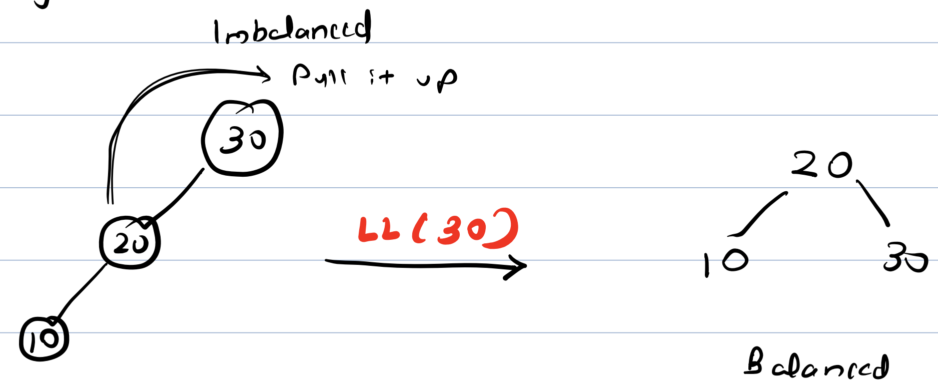
$|BF| \leq 1$

$BF_{\text{of a Node}} = H_L - H_R$

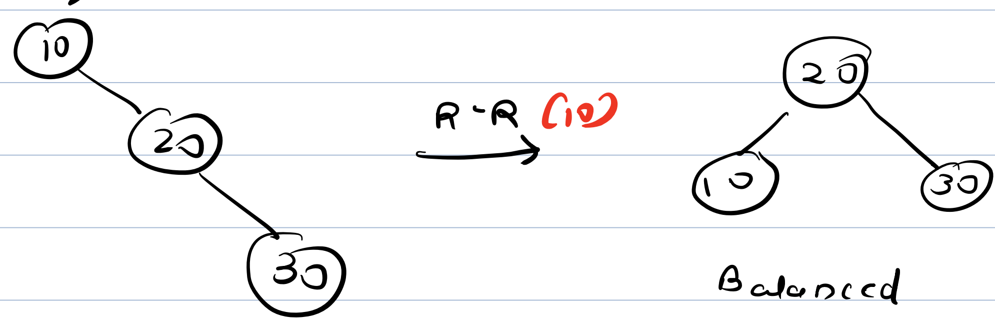




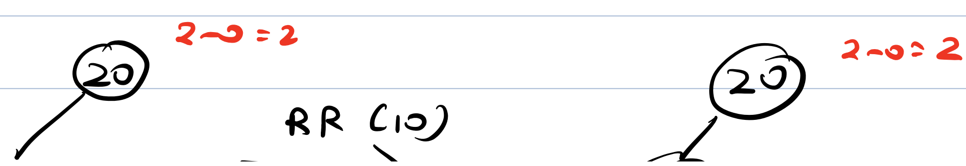
1. Left-Left Rotation

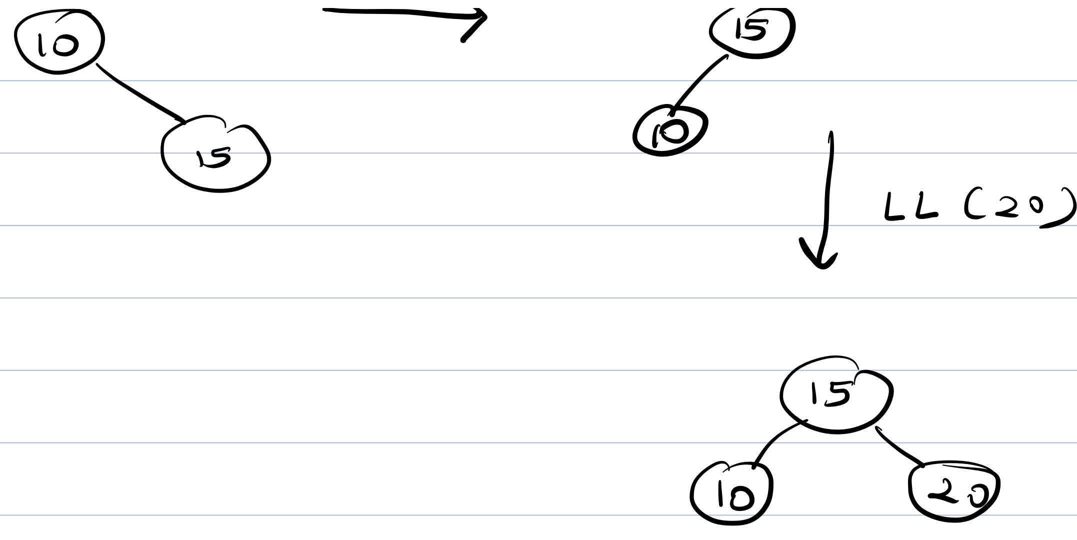


2- Right-Right Rotation



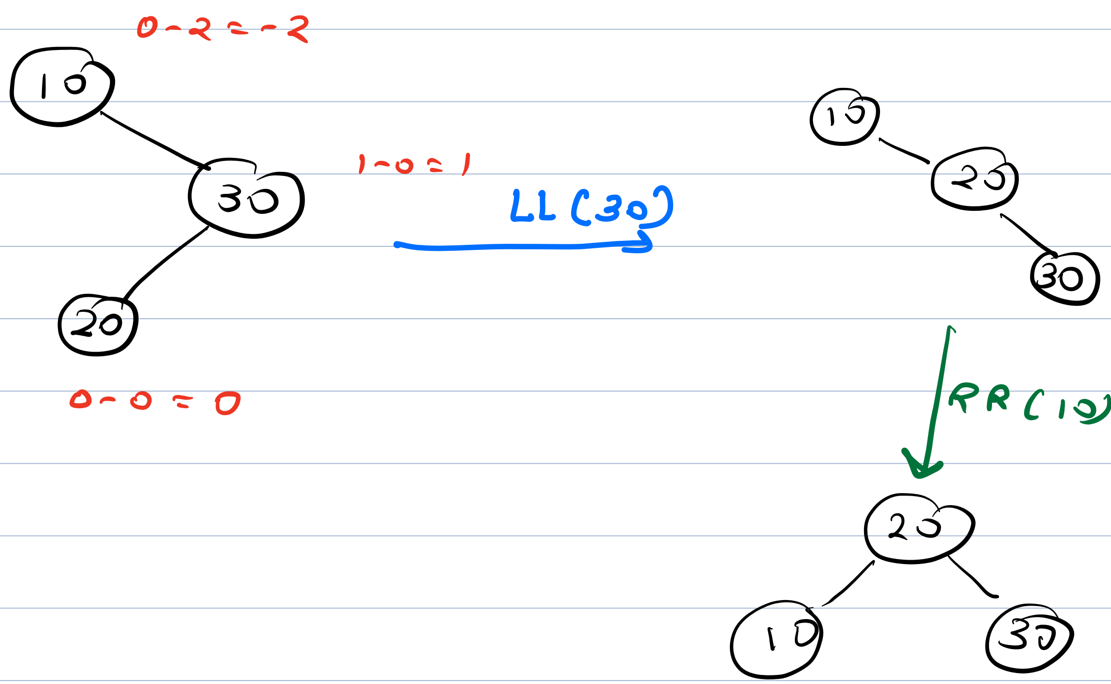
3. Left Right Rotation





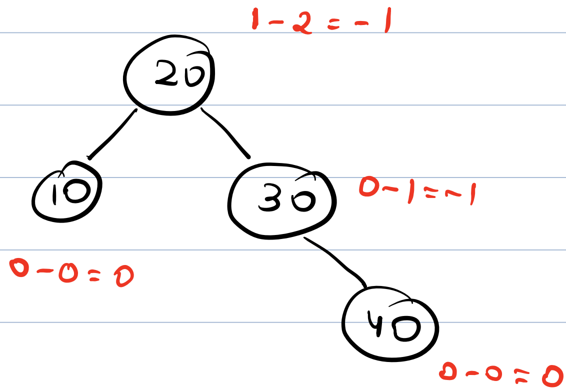
Balanced

4. Right left Rotation



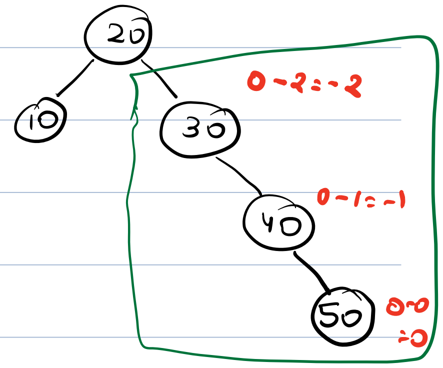
Balanced

Insertion in AVL Tree

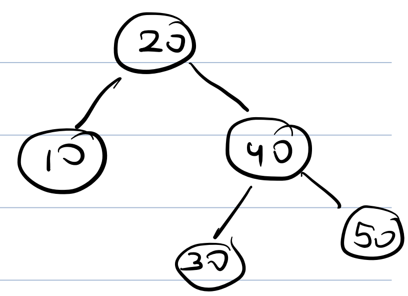


Balanced

add 50 →



↓ RR



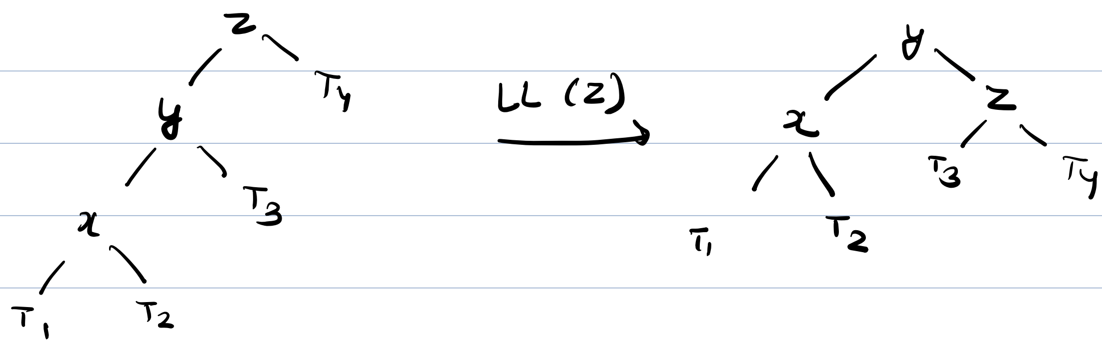
Balanced BST
(AVL Tree)

z is inbalanced node.

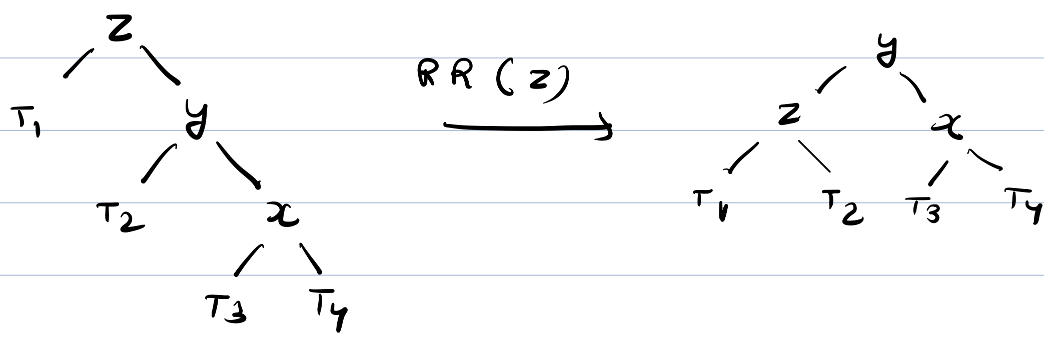
y child of z, in which we inserted new element

x grand child, in which we inserted.

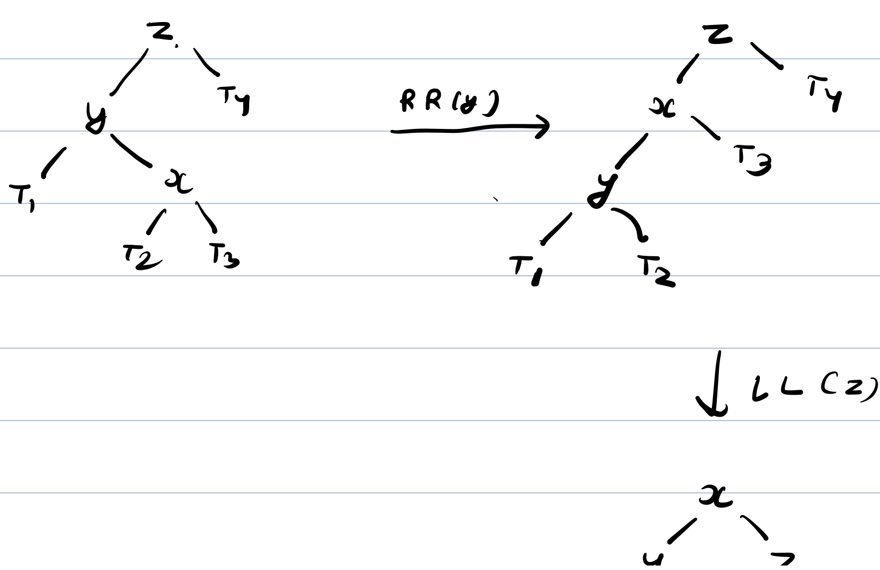
1. Left-Left

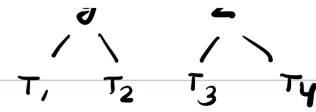


2. Right - Right Rotation.

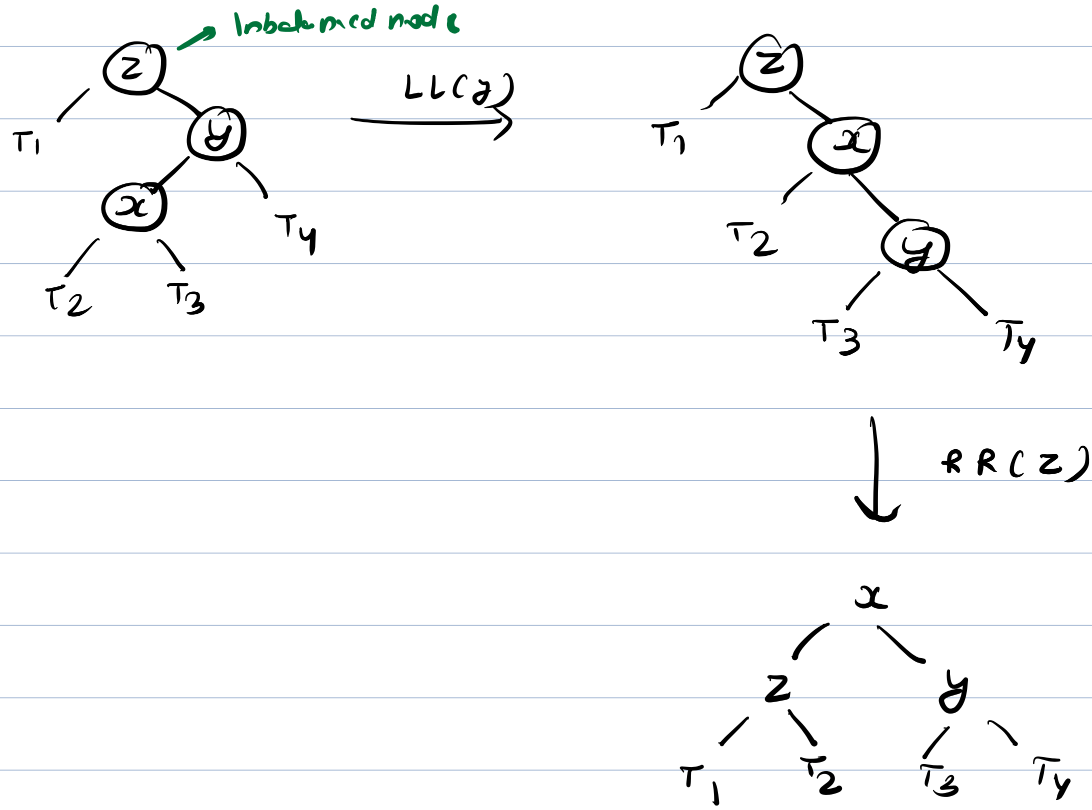


3. Left - Right Rotation





4. Right left Rotation



Balanced BST

Insertion

1. Insert the node using BST insertion logic
2. Track back the path where you just inserted the node
3. Check balance factors, if its value is inside -1 to 1, then perform rotation of this node.

z imbalanced Node

y child of z, where insertion

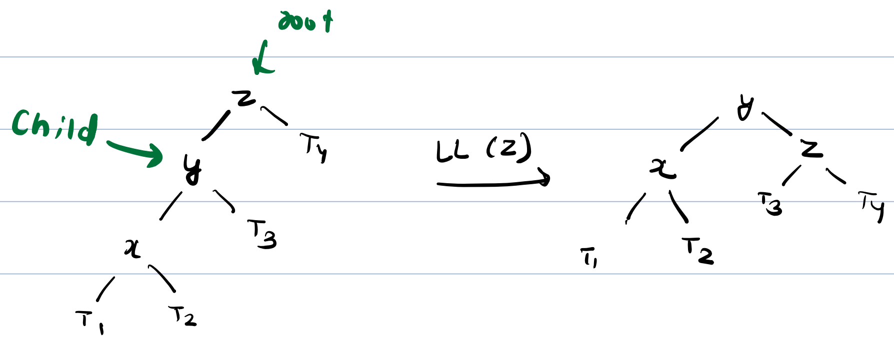
x child of y, where insertion

Node
```
{
    int data, height
    Node * left
    node * Right
}
```

```
int get_balance_factor( Node * x )
{
    return x -> left -> height - x -> right -> height
}
```
If NULL = 0 (for x -> left), If NULL = 0 (for x -> right)

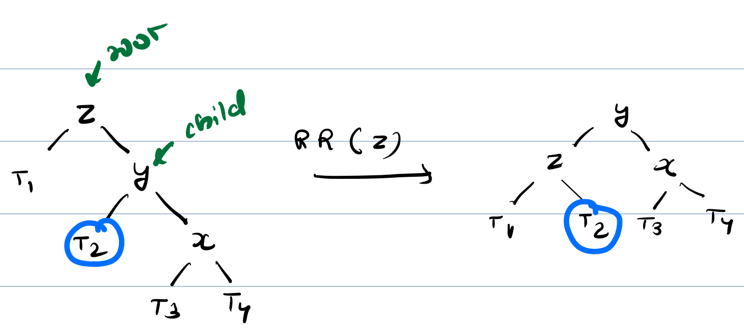
```
Node * LL Rotation ( Node * rt )
{
    Node child = root -> left
    root -> left = child -> right
    Child -> right = root
    root -> height = max( root -> left -> height, root -> right -> height)
    Child -> height = max( child -> left -> height, child -> right -> height)

    return Child
}
```




```
Node * RRRotation ( Node * root )
{
    Node child = root -> right
    root -> right = child -> left
    child -> left = root
    root->height = max( root->left->height, root->right->height)
    child->height = max( child->left->height, child->right->height)

    return child.
```




```
Node * Insert ( Node * root, int data )
{
```

```
{
    if( root == NULL )
    {
    }
    if ( root -> data < data )
    {   root -> right = insert( root -> right, data)
    else
        root -> left = insert ( root -> left, data )

    int b = get_bal_factor (root)
}
if (b > 1 )
{
    // LL or LR
    int c = get_bal_factor ( root -> left )
    if ( c > 0 )
    {   root = LL ( root )
    else
    {   root -> left = RR ( root -> left )
        root = LL ( root )
}
else if (b < -1 )
{
    RL or RR
    int c = get_bal_factor ( root -> right )
    if ( c > 0 )
    {   root -> right = LL ( root -> right )        // RL
        root = RR ( root )
```

else
{ root = RR(root) //RR

return root
}

Deletion :

1. Delete node as it is BST
2. Track back the path where you just inserted the node
3. Check balance factors, If its value is inside -1 to 1, then perform rotation of this node.

Searching time in AVL Tree : O(logN)
Insertion : O(logN)
Deletion O(logN)