LECTURE ASSIGNMENT

NAME: MAHER ESMAT SOHSAH

JUNE 8, 2021

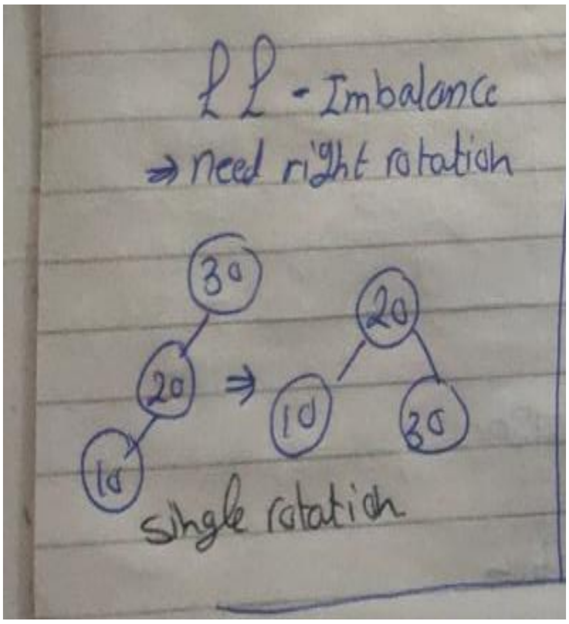
AVL TREE Insertion and Deletion

TYPES OF ROTATION: LL, RR, LR, RL

EX ON: 10, 20, 30

ROTATIONS:

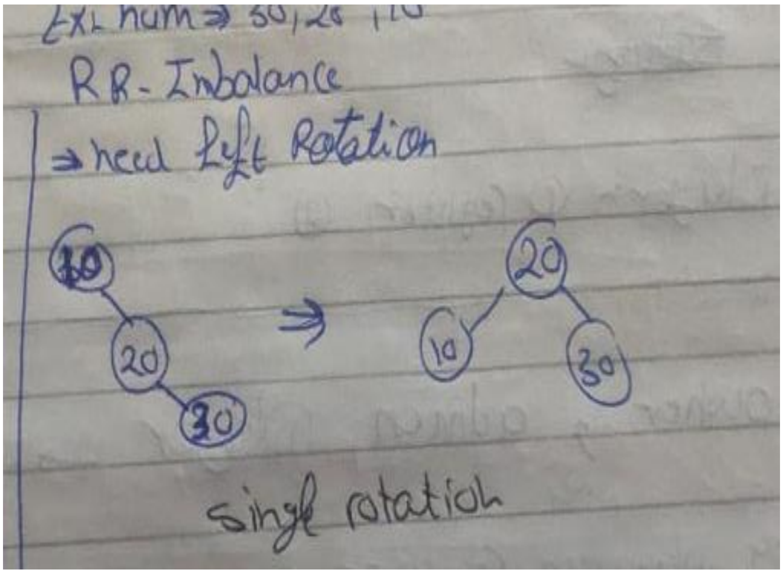
1. Right rotation.


LL - Imbalance

⇒ need right rotation

single rotation

2. Left rotation

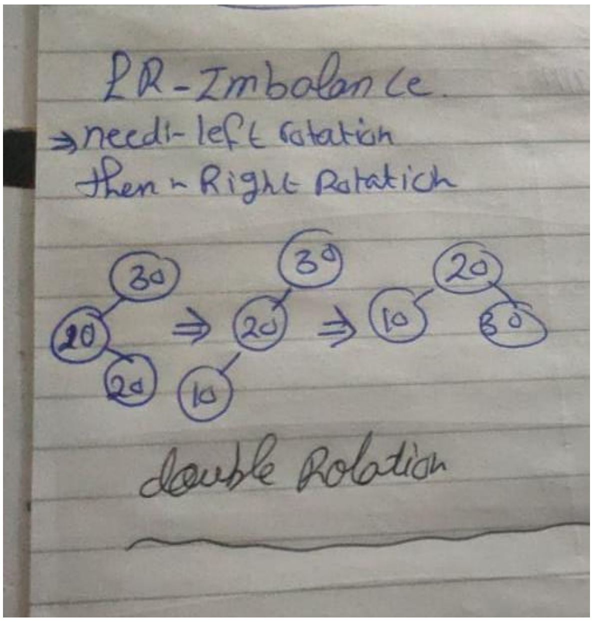

Ex: num ⇒ 30, 20, 10

RR - Imbalance

⇒ need left Rotation

single rotation

3. Left Right

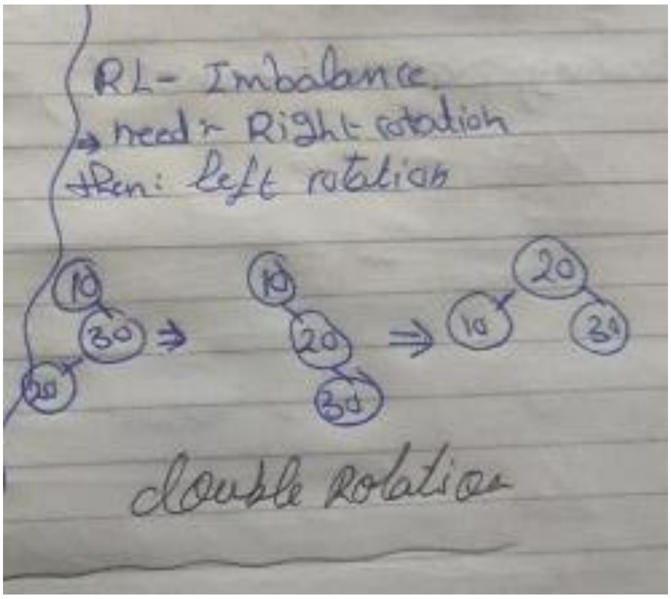

LR - Imbalance.

⇒ need:- left rotation

then :- Right Rotation

double Rotation

4. Right Left


RL - Imbalance.

⇒ need :- Right rotation

then: left rotation

double Rotation

Insertion Code

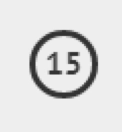
-
  ```
  Class TreeNode{
    Int value; TreeNode *left, *right;
    TreeNode()
    { value = 0; left = right = NULL;}
    TreeNode(int v)
    {value = v; Left = right = NULL;}
  ```

-
  ```
  Int getBalance(TreeNode *n)
    {
        1. If (n == NULL)
            1.1.   Return -1;
        2. Else
            2.1.   Return (height(n-> left) – height(n-> right));
     }
  ```

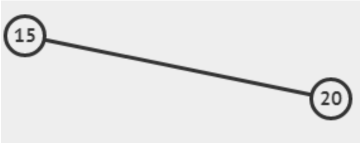
-
  ```
  Int height(TreeNode *r)
    {
        1. If (r == NULL)
            1.1.   Return -1;
        2. Else
            2.1.   Lheight = height(r -> left);
            2.2.   Rheight = height(r -> right);
            2.3.   If (Lheight > Rheight)
                2.3.1.       Return (Lheight + 1);
            2.4.   Else
                2.4.1.       Return (Rheight + 1);
      }
  ```

```
• TreeNode* rightRotate(TreeNode *y)
    {
      1. TreeNode *x = y -> left;
      2. TreeNode *temp = x -> right;
      3. X -> right = y;
      4. Y ->left = temp;
      5. Return x;
     }
• TreeNode* leftRotate(TreeNode *x)
    {
      1. TreeNode *y = x-> right;
      2. TreeNode *temp = y -> left;
      3. Y -> left = x;
      4. X -> right = temp;
      5. Return y;
     }
• TreeNode* insert (TreeNode *r, TreeNode *newnode)
{
    1. If (r == NULL)
          a. r = newnode;
          b. return r;
    2. if (newnode -> value smaller r-> value)
          a. r -> left = insert(r->left, newnode)
    3. Elseif(newnode -> value greater r-> value)
          a. r -> right = insert(r -> right, newnode)
    4. Else
          a. Print ("no duplicate values!! " )
          b. Return r;
    5. Num = getBalance(r);
    6. If (num > 1 && newnode -> value ≤ r->left -> value)// LL-rotation
          a. Return rightRotate(r);
    7. If (num < -1 && newnode -> value ≤ r-> left -> value)//RR-rotation
        a. Return leftRotate(r)
    8. If (num > 1 &&newnode -> value ≤ r->left -> value) //LR-rotation
        a. r -> left = leftRotation(r -> left)
        b. return rightRotation(r)
    9. if (num < -1 && newnode -> value ≤ r ->right ->value)//RL-rotation
        a. r ->right = rightRotation(r ->right);
        b. return leftRotation(r);
    10.     return r;
}
```

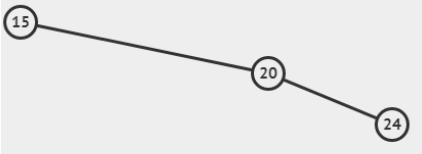
Insert 15:



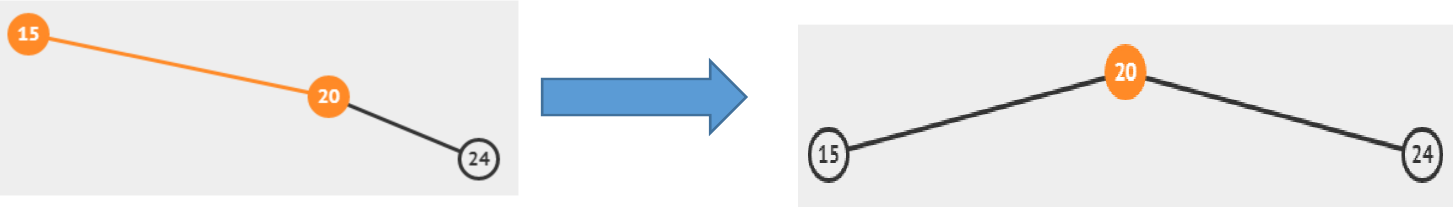
Insert 20:



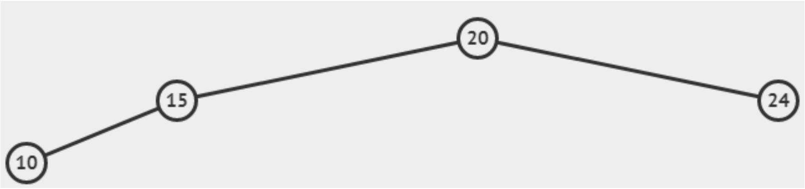
Insert 24:



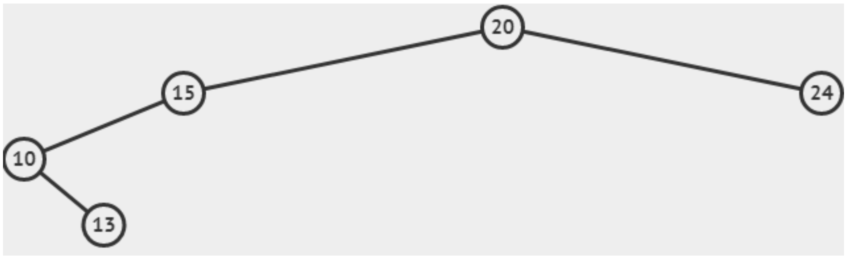
Need right rotation:



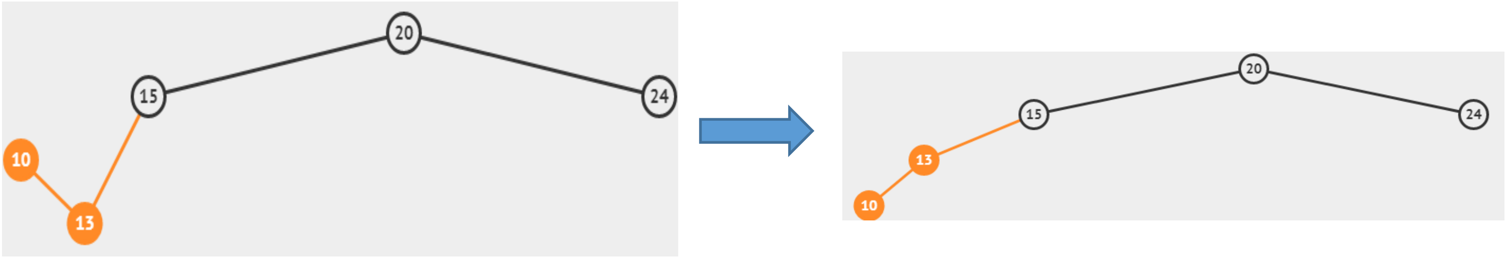
Insert 10:

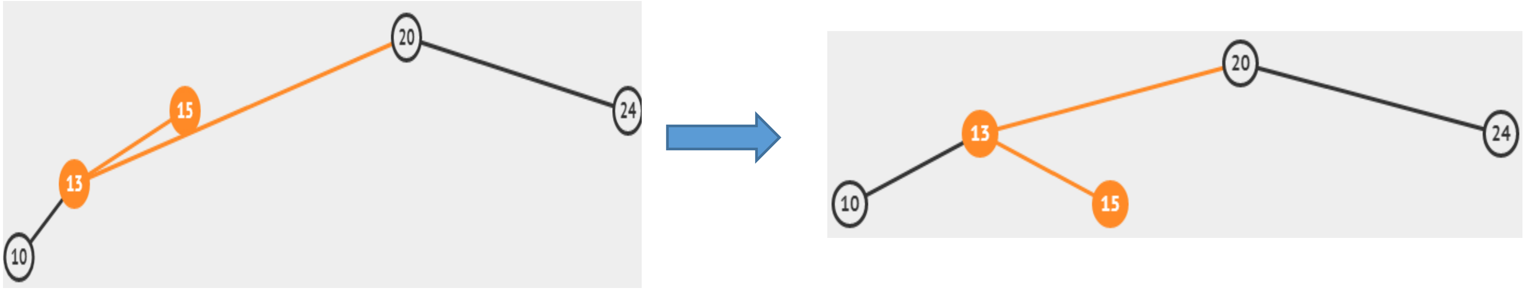


Insert 13:

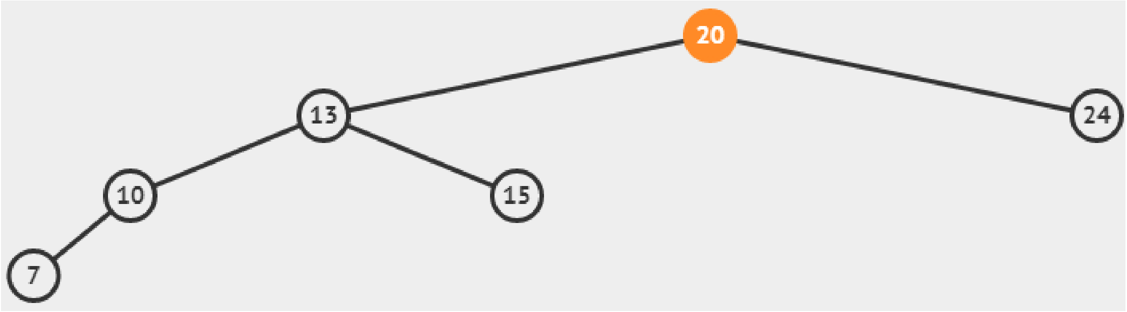


Need rotation:

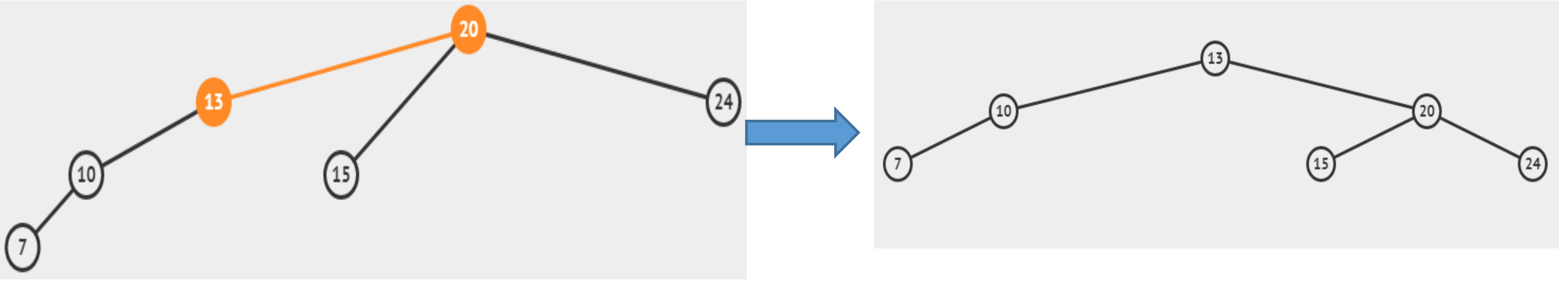




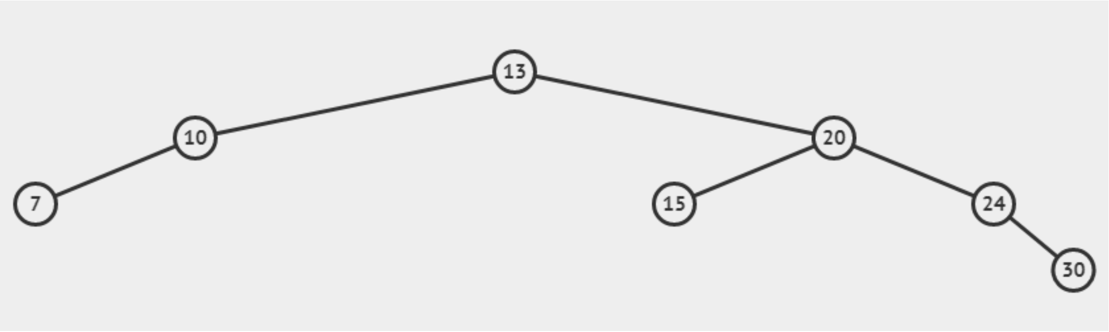
Insert 7:



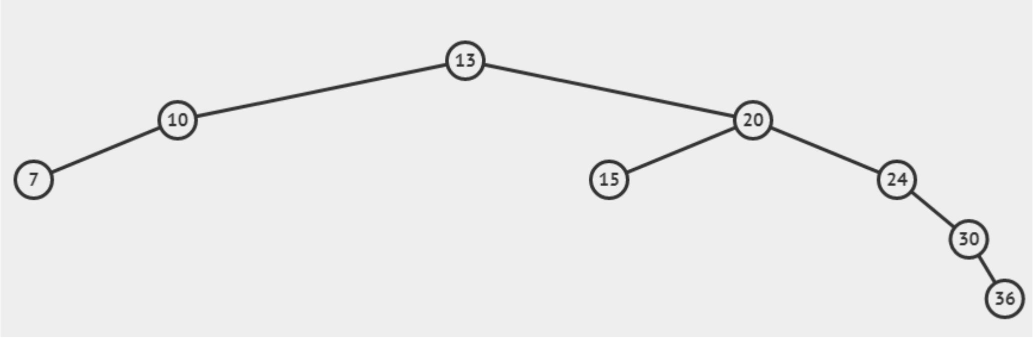
Need rotation:



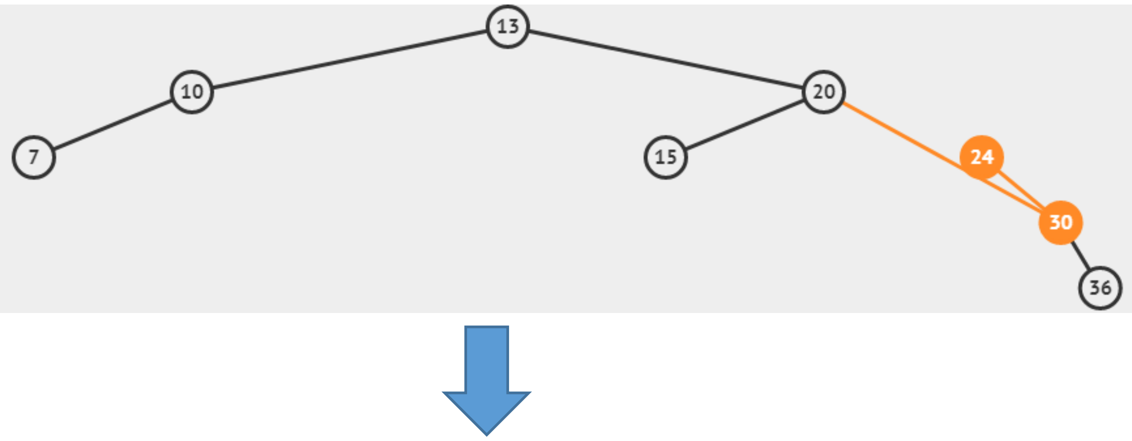
Insert 30:

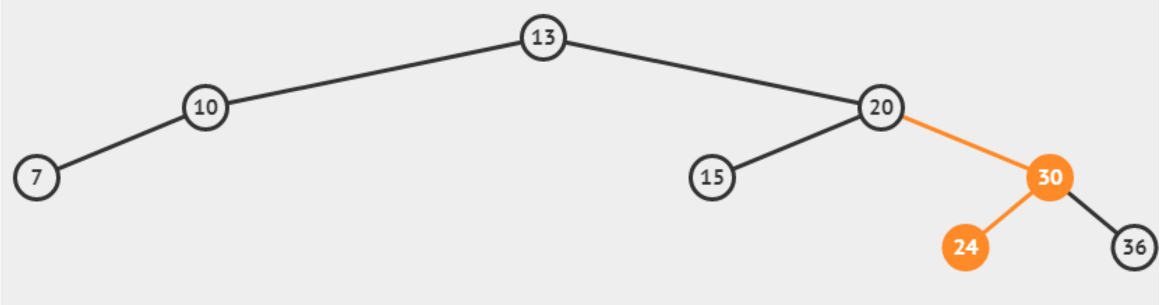


Insert 36:

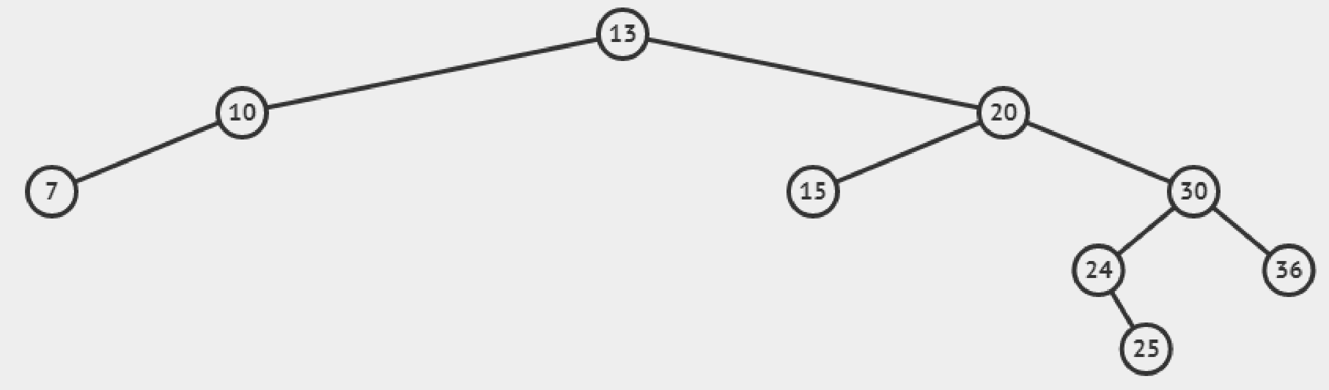


Need rotation:

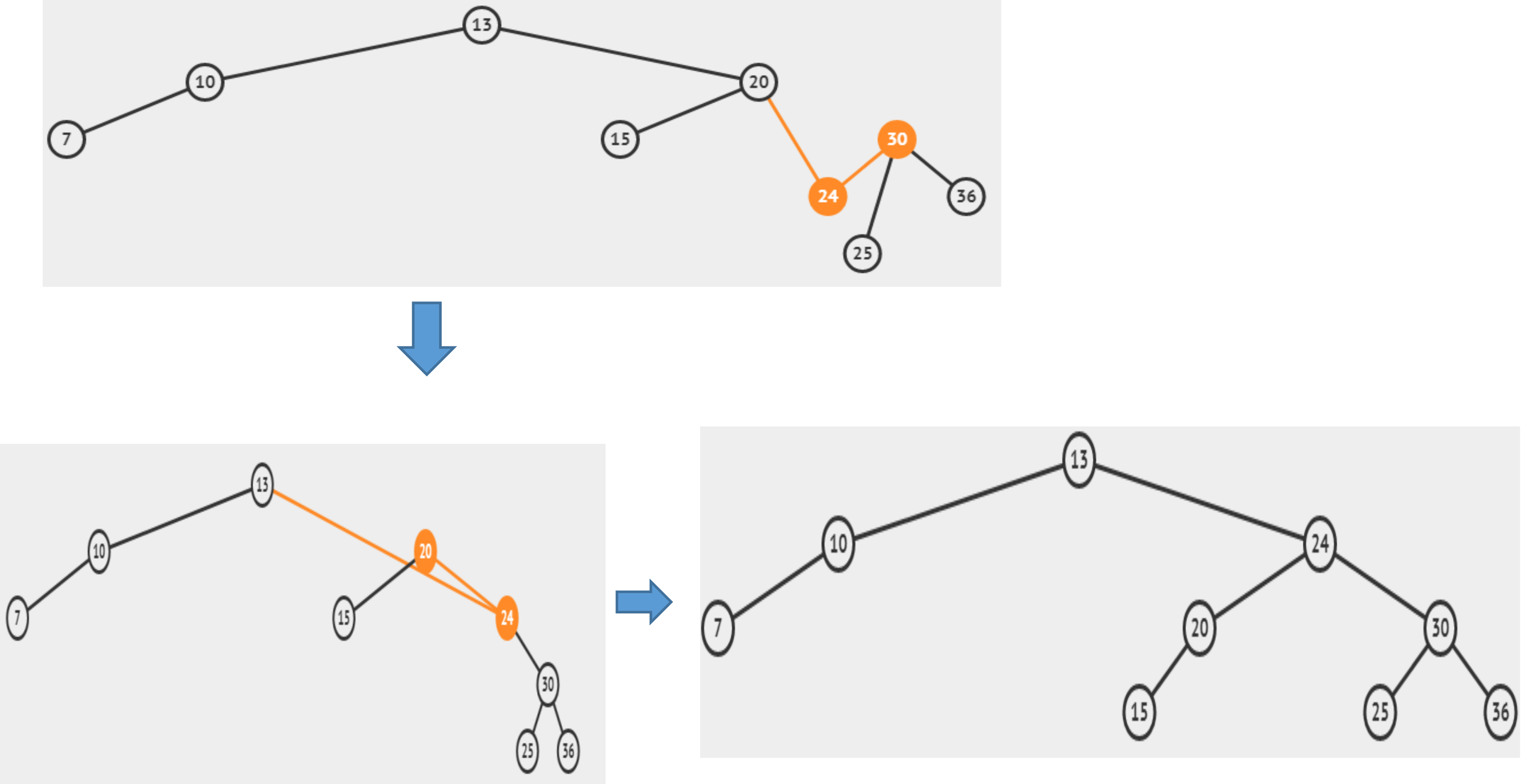




Insert 25:



Need rotation:



Deletion Code

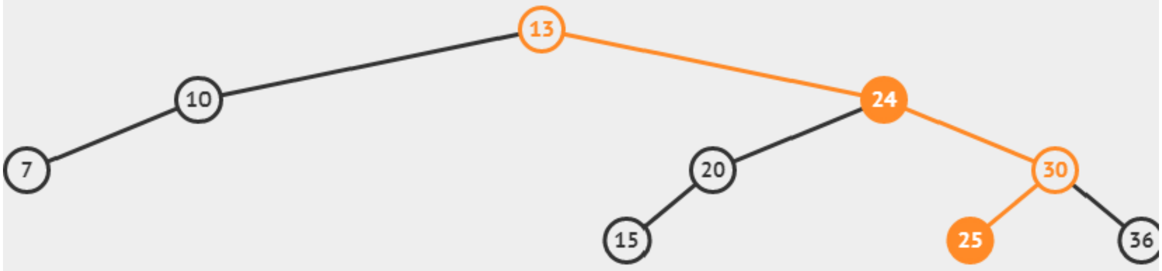
```
-   TreeNode* deleteNode(TreeNode* r, int v)
{
    1. IF (r == NULL)
          1.1.  Return r;
    2. Else IF (v ≤ r -> value)// if value smaller go left
          2.1. r -> left = deleteNode(r->left, v)
    3. Else IF(v ≥ r-> value) // value larger go right
          3.1. r -> right = (deleteNode(r -> right, v)
    4. Else
       4.1.  if (r -> left == NULL) // node with right child only
          4.1.1. temp = r-> right
          4.1.2.  delete r
          4.1.3. return temp
       4.2. Else IF(r -> right == NULL) // node with only left child
          4.2.1 temp = r -> right
          4.2.2. delete r
          4.2.3. return temp
       4.3. Else // node has 2 children
          4.3.1 temp = minval(r –> right)
          4.3.2. r-> value = temp -> value
          4.3.3. r-> right = deleteNode(r -> right, temp ->value)
    5. num = getbalance (r)
    6. IF num == 2 and getbalance (r -> left ) >= 0 // LL - imbalance
          6.1. return rightRotate(r)
    7. Else IF (num == 2 && getbalance (r -> left) == -1) // LR - imbalance
```

No. 6, 7, 8, 9 are condition on which we perform single or double rotation.

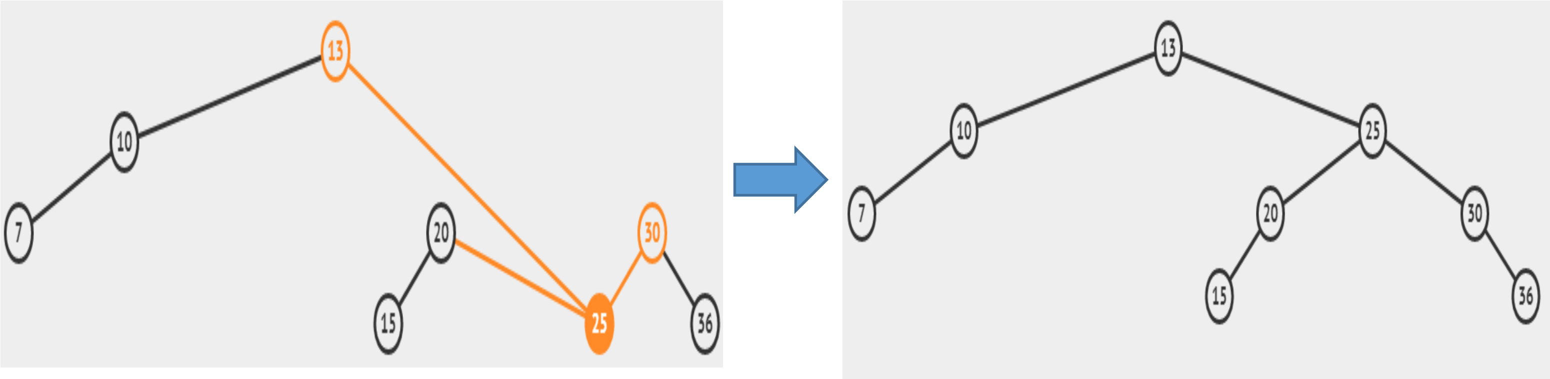
```
        7.1. r -> left = leftRotate(r -> left)
        7.2. return rightRotate(r)
    8. Else IF(num == -2 && getbalance (r -> right) <= 0) // RR imbalance
        8.1. return leftRotate(r)
    9. Else IF(num == -2 && getbalance (r-> right) == 1) // RL imbalance
        9.1. r -> right = rightRotate(r -> right)
        9.2. return leftRotate(r)
    10. return r
}
```

Deletion:

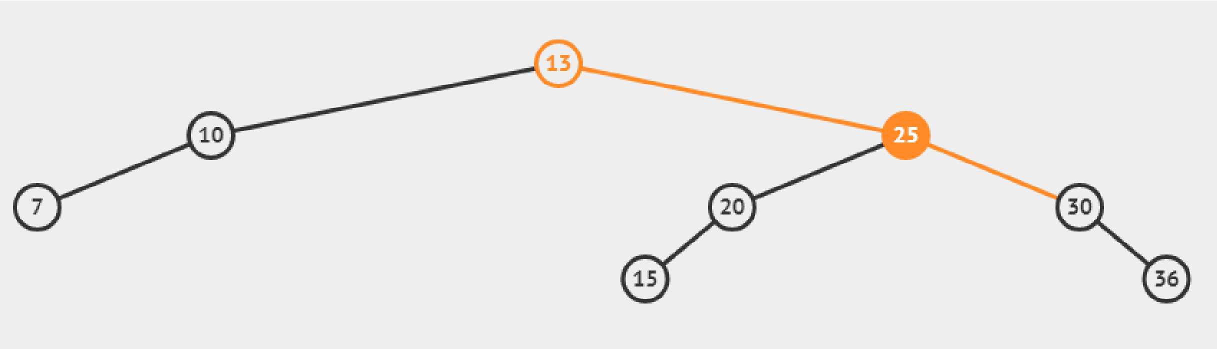
Delete 24:



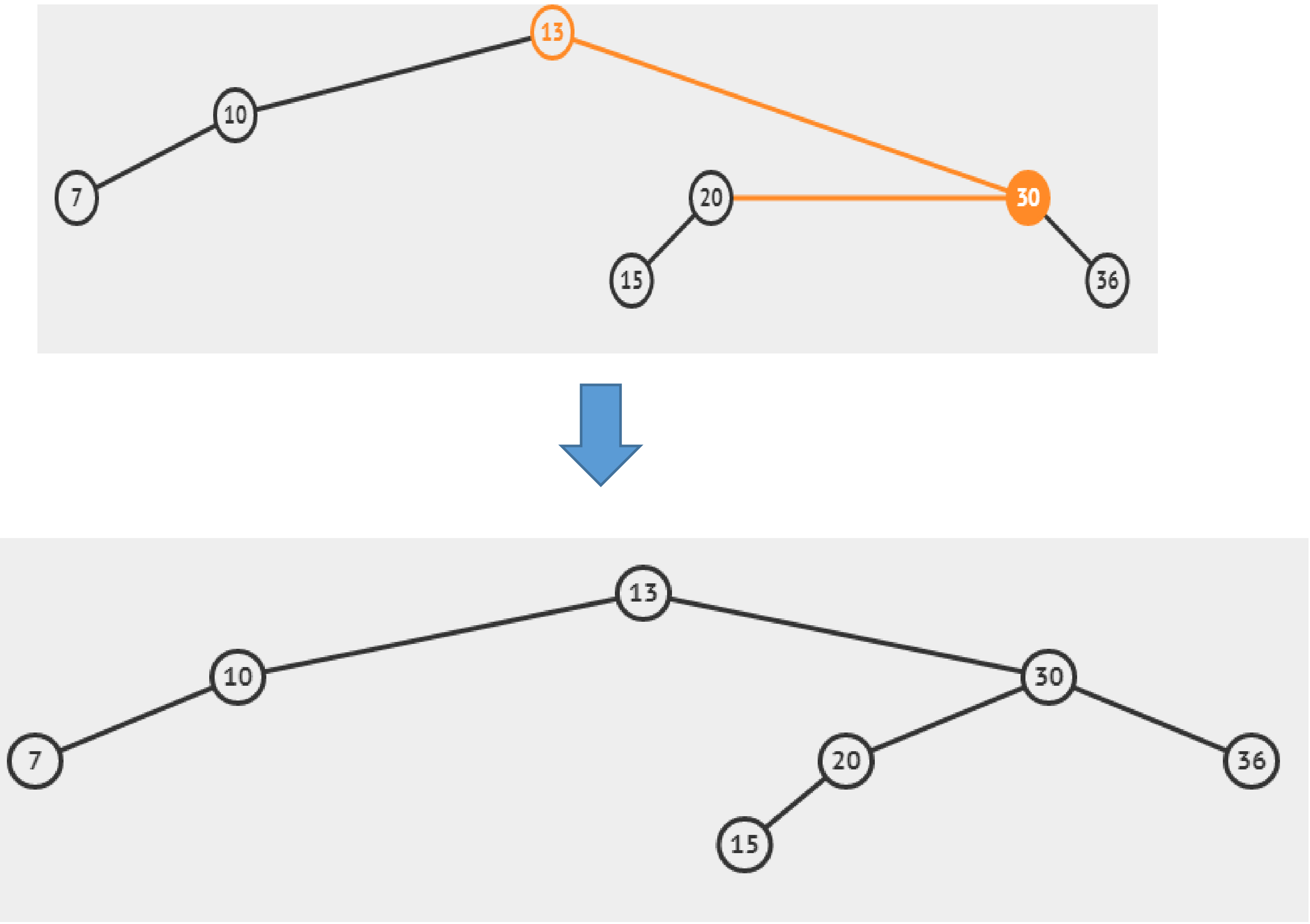
Need Rotation:



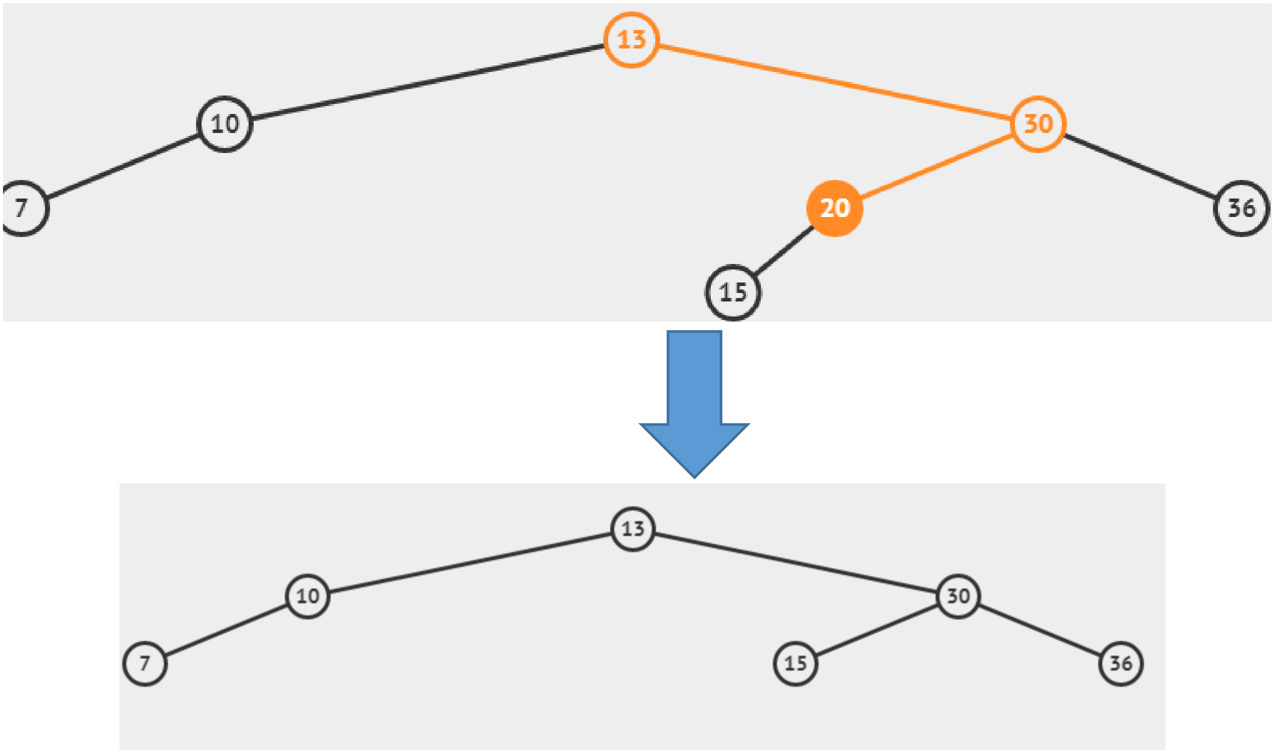
Delete 25:



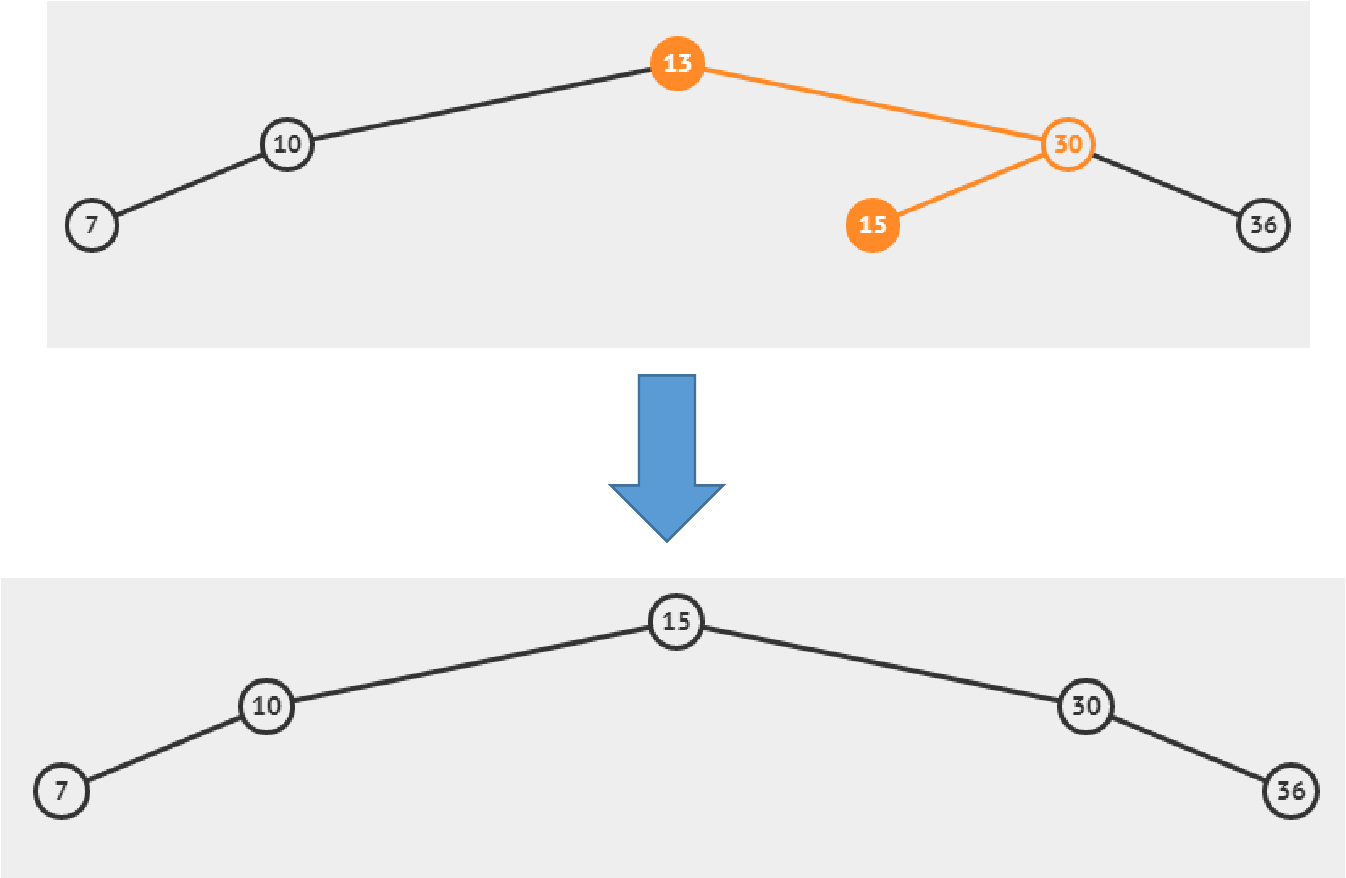
Need Rotation:


13
10
7
20
30
15
36
13
10
30
7
20
36
15

Delete 20:


13
10
30
7
20
36
15
13
10
30
7
15
36

Delete 13:



Deleting nodes: 10, 30 like deleting node 20.
Deleting node: 15 like deleting node 13.