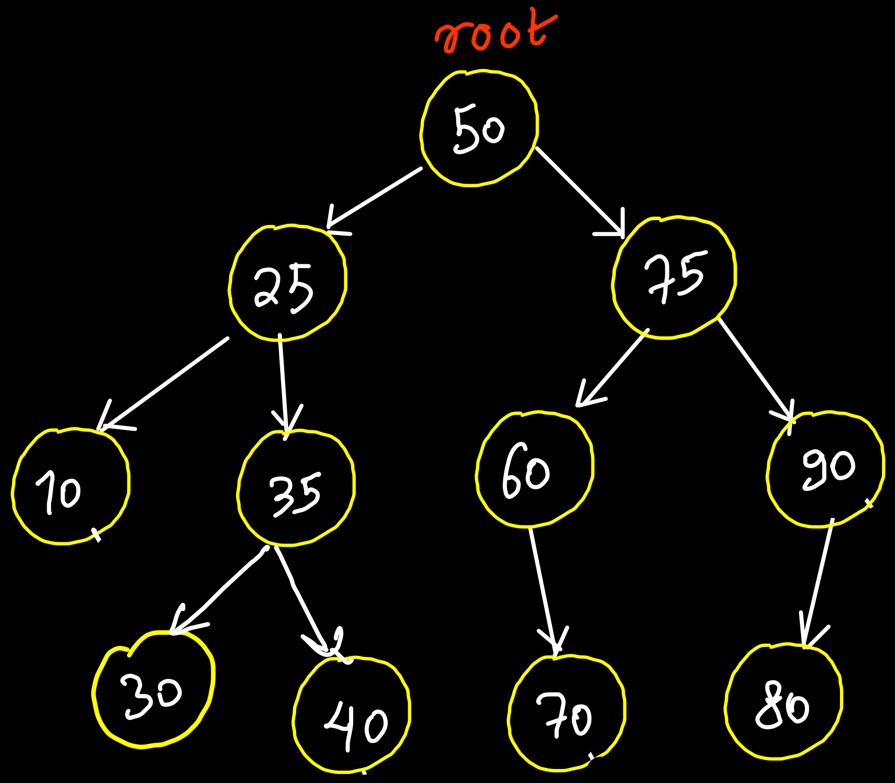
Binary Search Tree (BST)



1. left < root < right

2. inorder → sorted

insertn

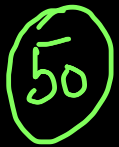
avg case → O(log n) balanced bst

worst case → O(n) skewed bst

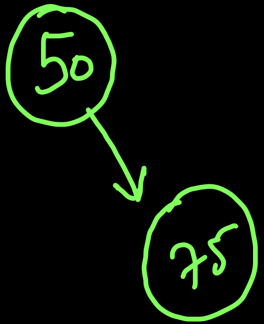
normal bst (n)

Insert in BST

① insert 50

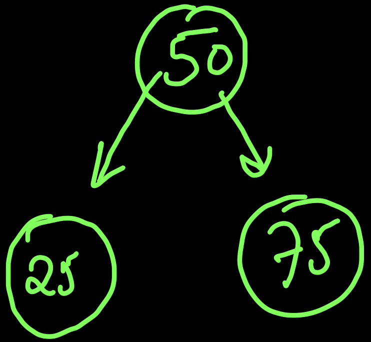


② insert 75

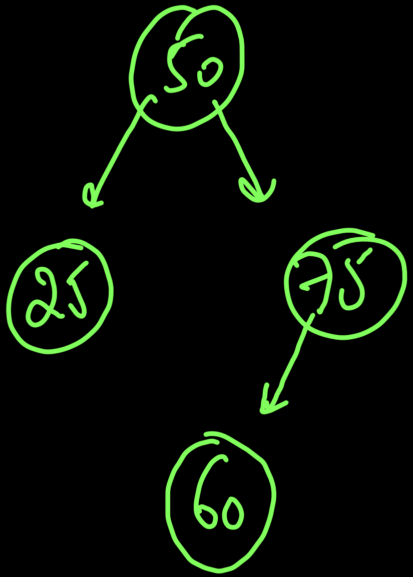


insertion order → balanced

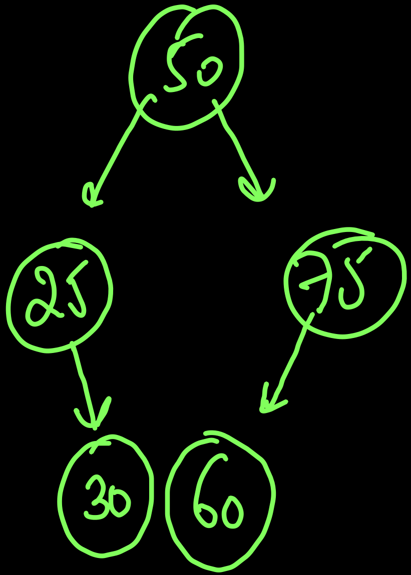
③ insert 25



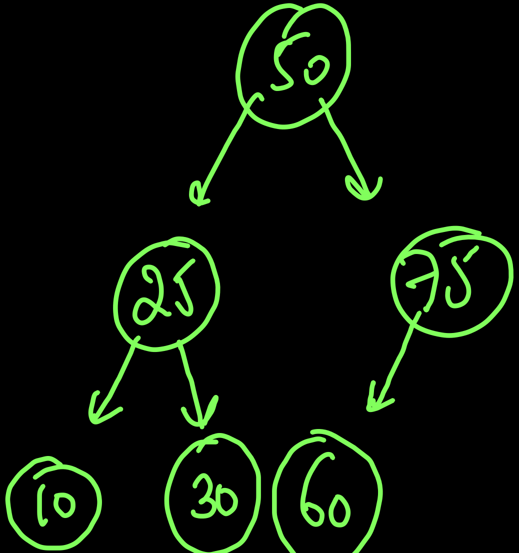
④ insert 60



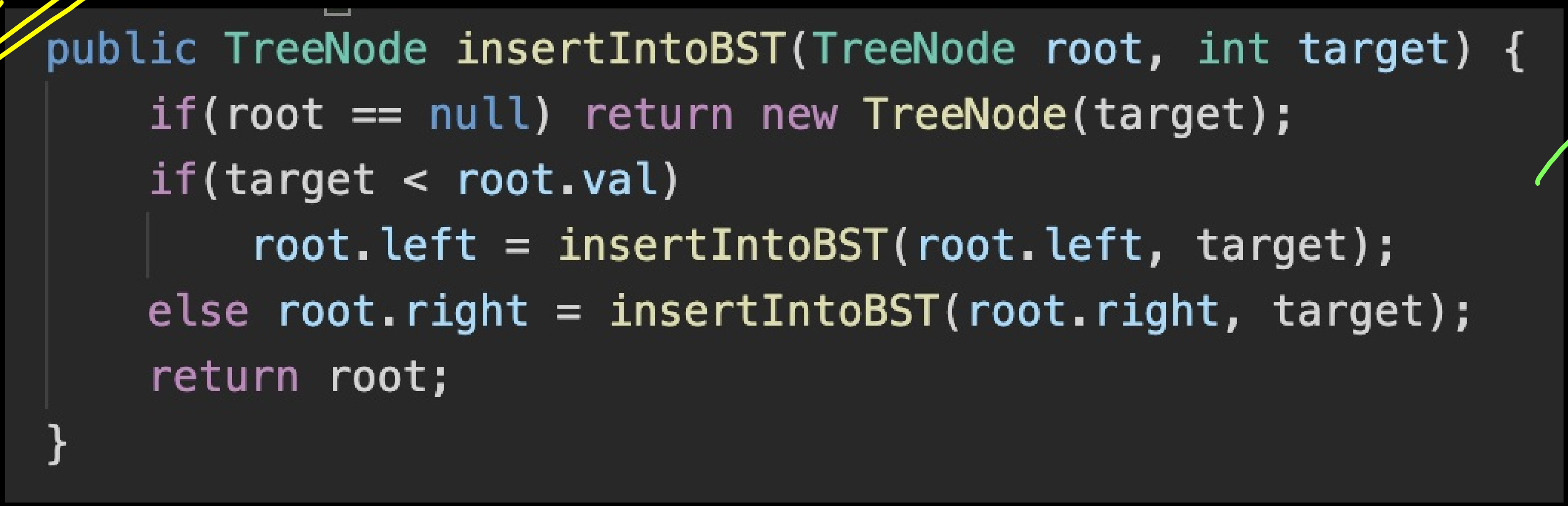
⑤ insert 30



⑥ insert 10



lc 701



```
public TreeNode insertIntoBST(TreeNode root, int target) {
    if(root == null) return new TreeNode(target);
    if(target < root.val)
        root.left = insertIntoBST(root.left, target);
    else root.right = insertIntoBST(root.right, target);
    return root;
}
```

Normal BST

(insertion)

avg case → O(logn)

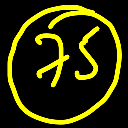
worst case → O(n)

balanced bst → for every node (balancing factor)

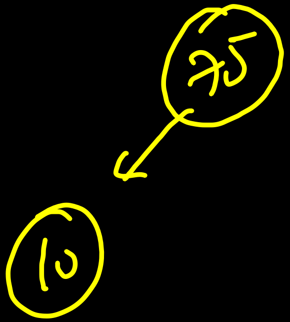
|left height - right height| ≤ 1

Insert into BST (skewed tree)

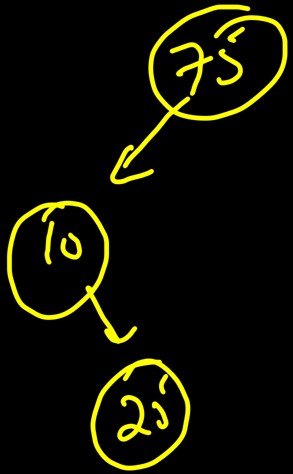
1) insert 75



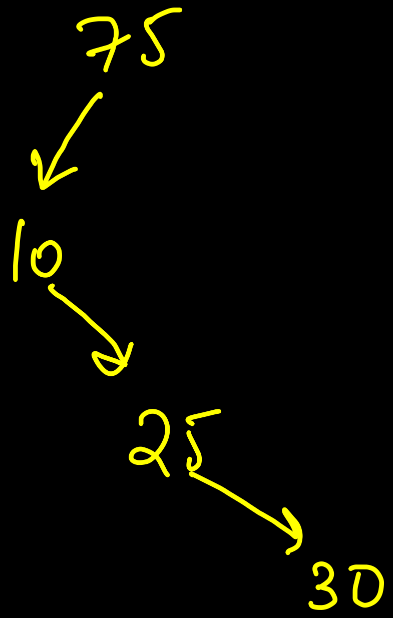
2) insert 10



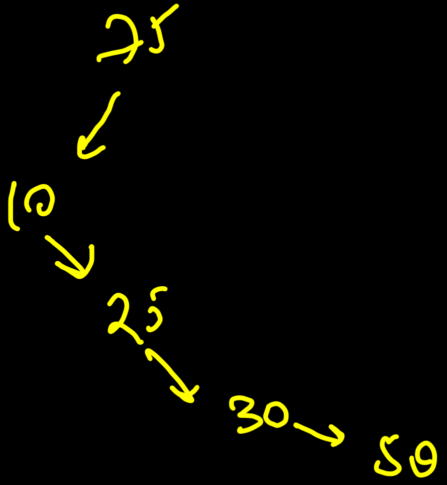
3 insert 25



4) insert 30



5) insert 50

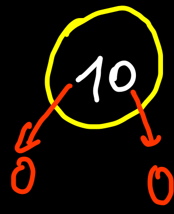


worst case
↳ O(n)

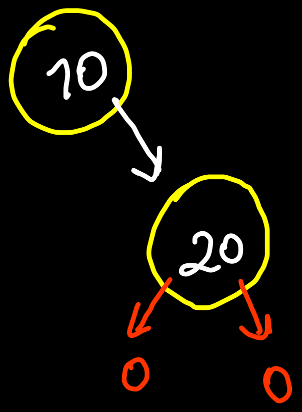
search
↳ O(n)

# AVL Tree → Self Balancing BST → TreeSet{K}
↘ Treemap{K,V}

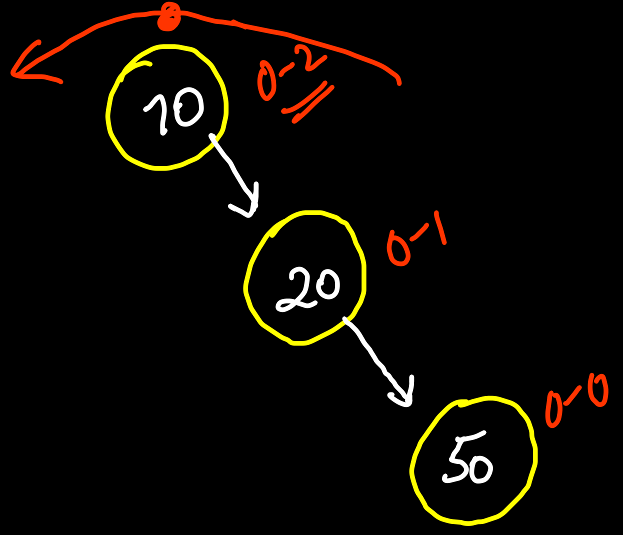
1) insert 10

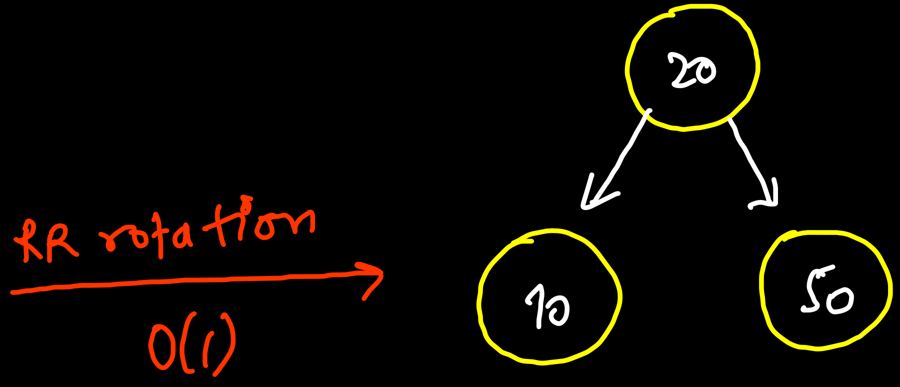


2) insert 20

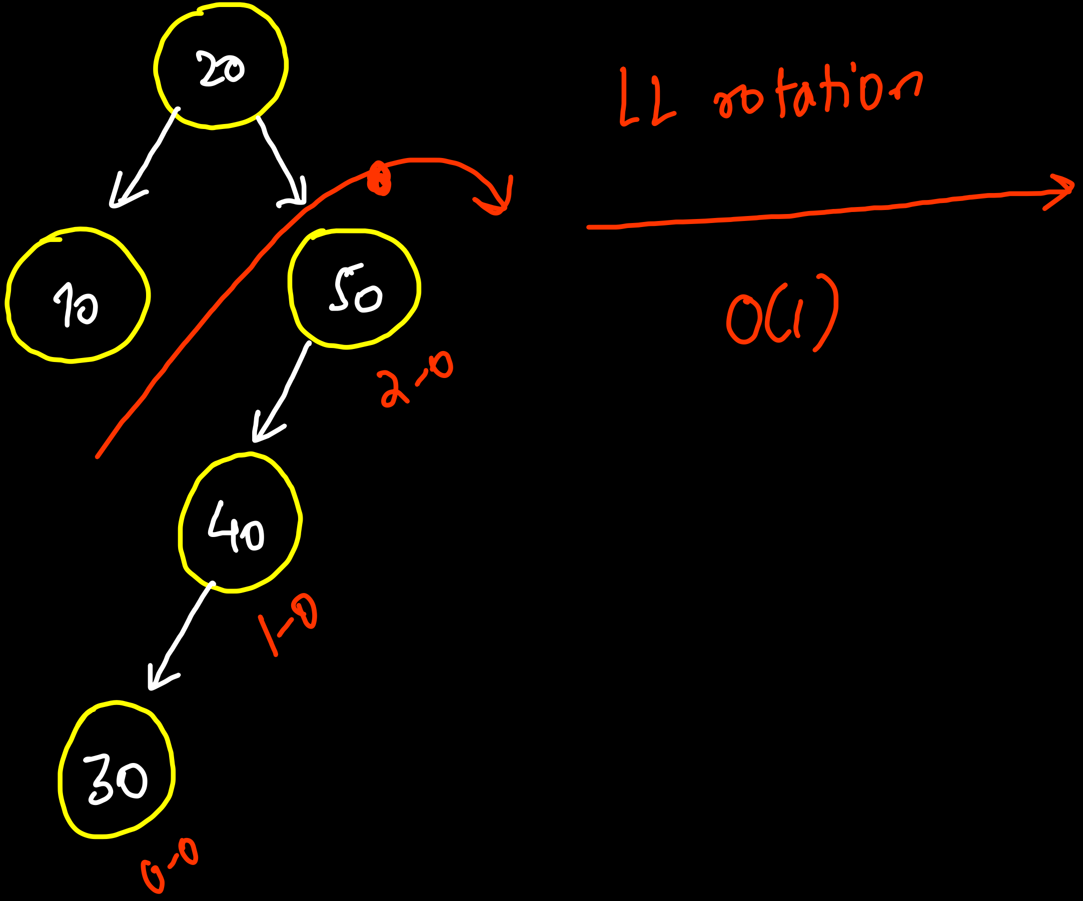


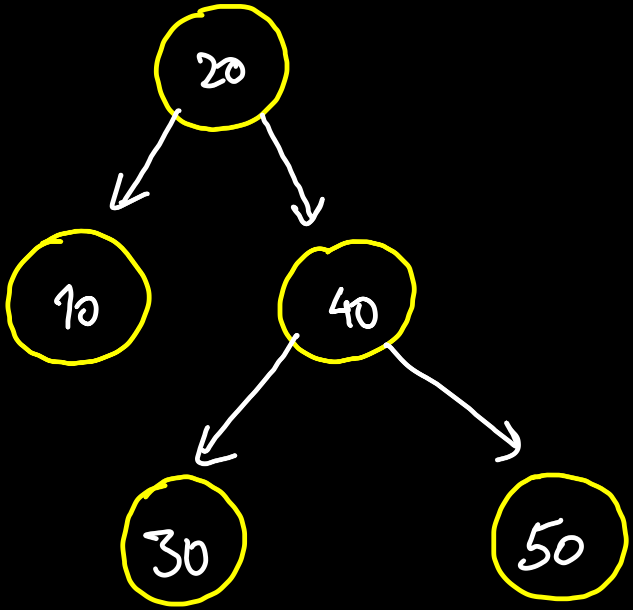
3) insert 50



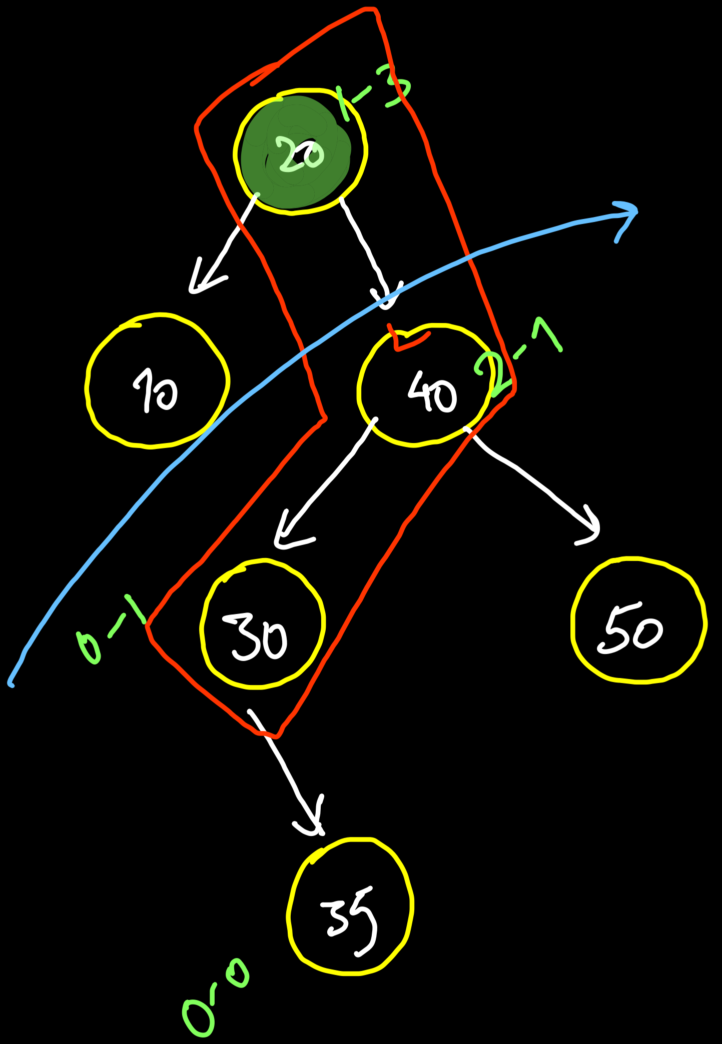


4) insert 40, 30



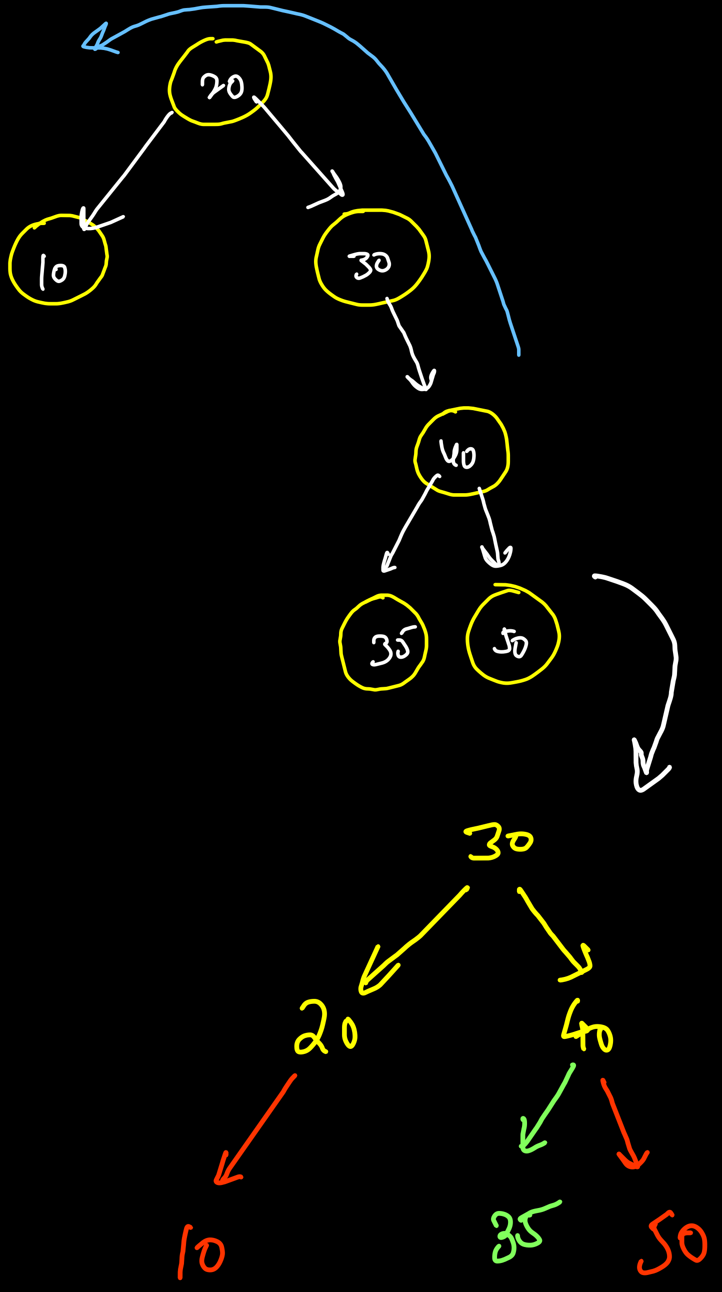


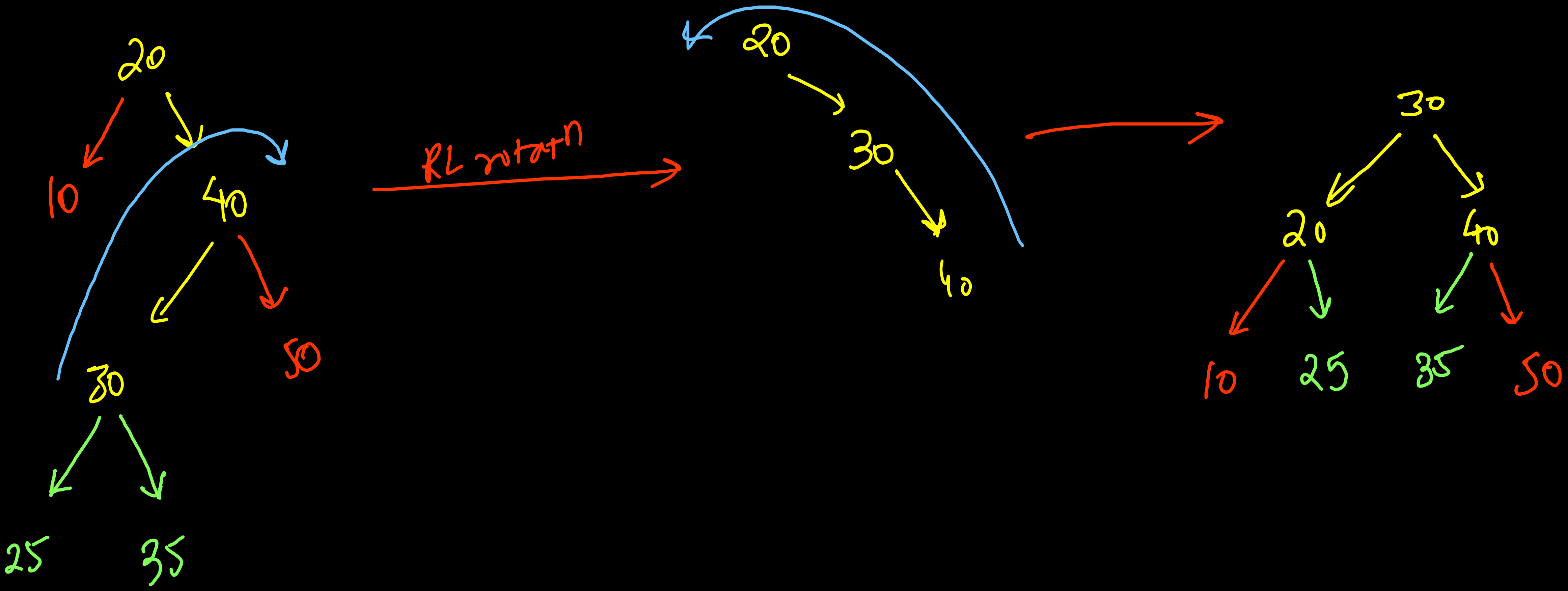
5) insert 35



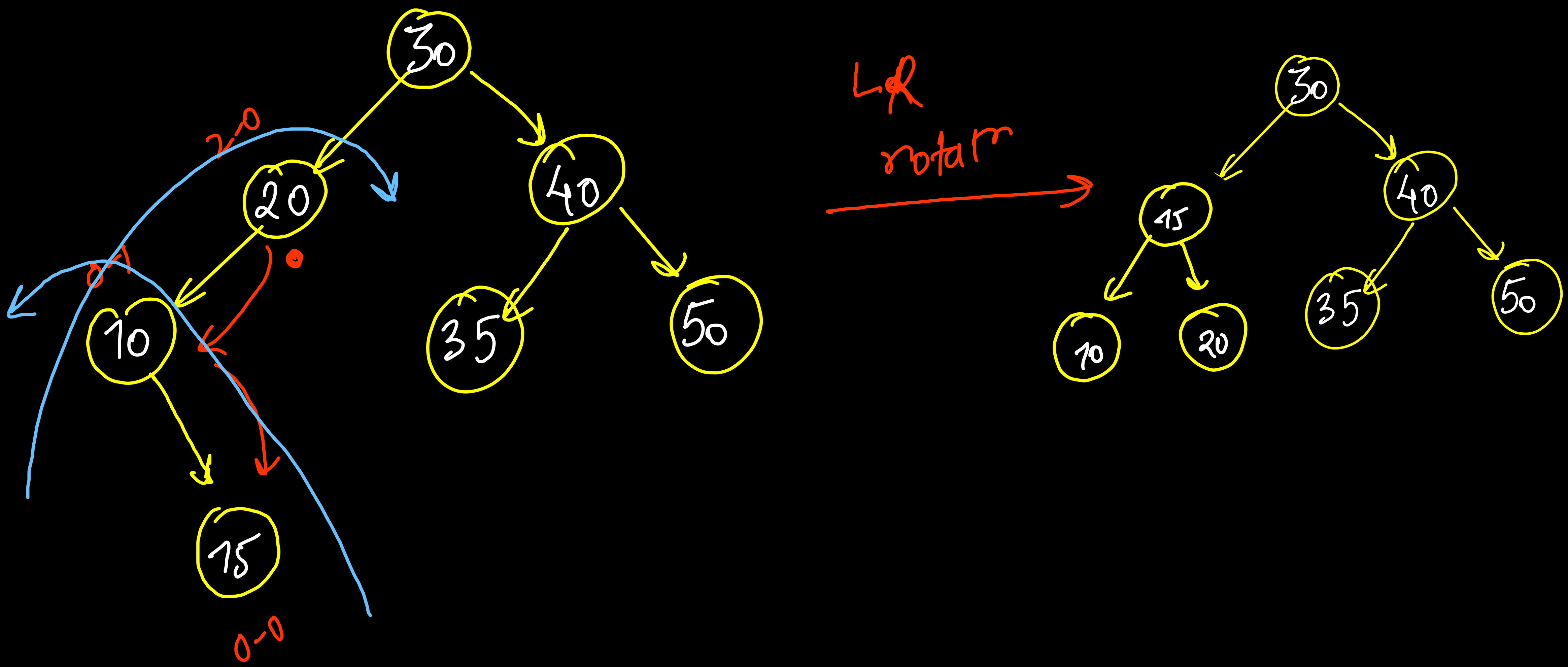
RL rotatn

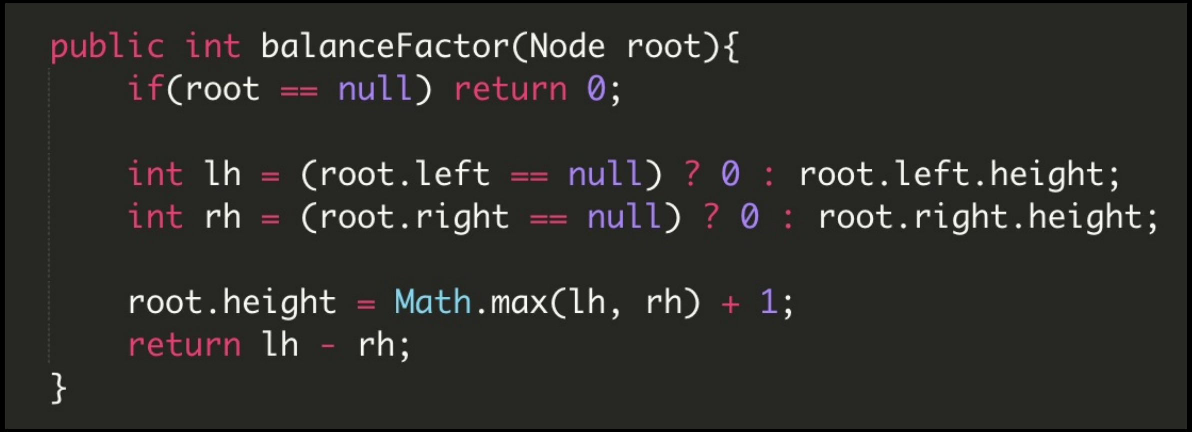
O(1)



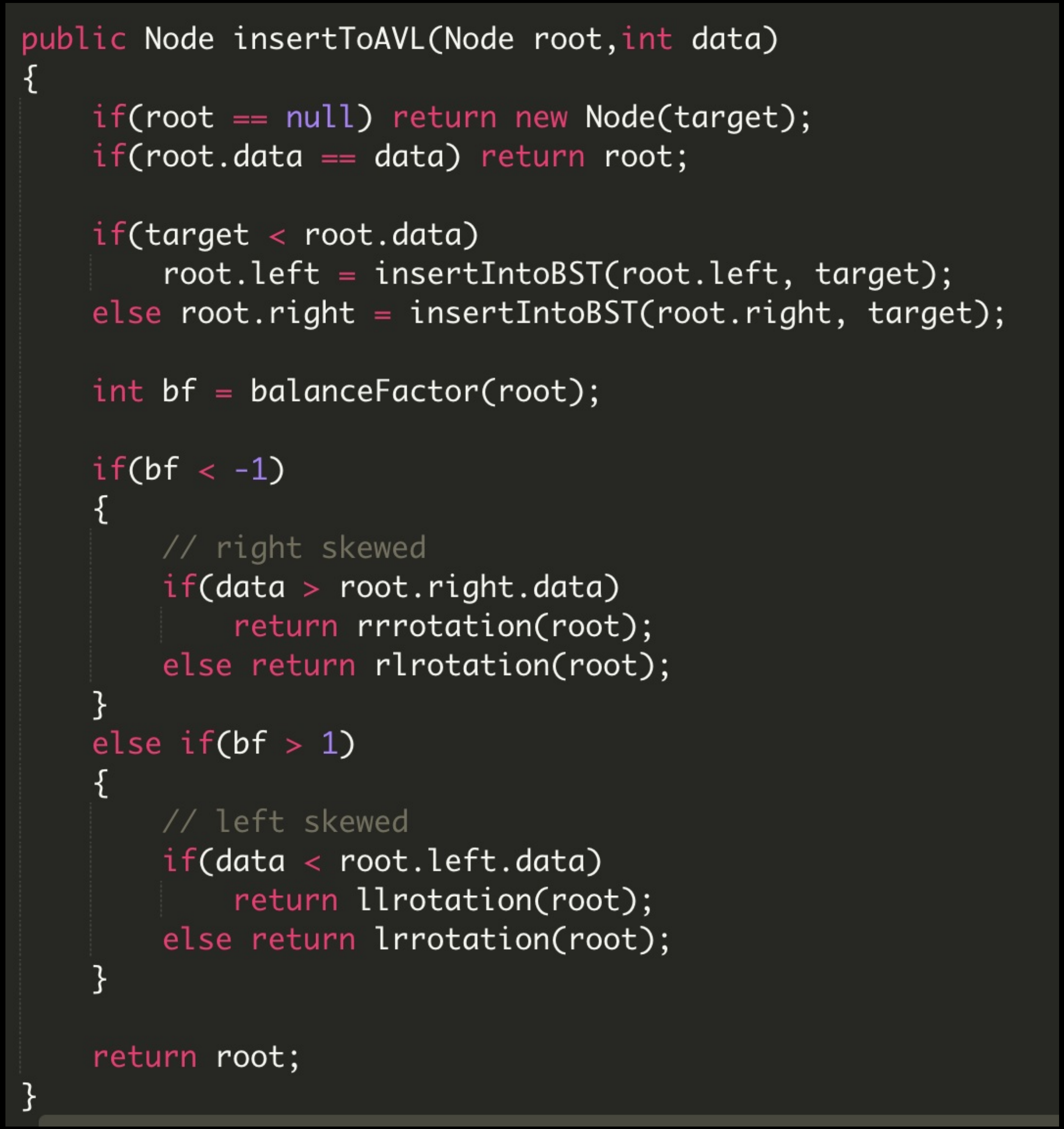

20
10
40
30
50
25
35
RL rotatn
20
30
40
30
20
40
10
25
35
50

6) insert 15

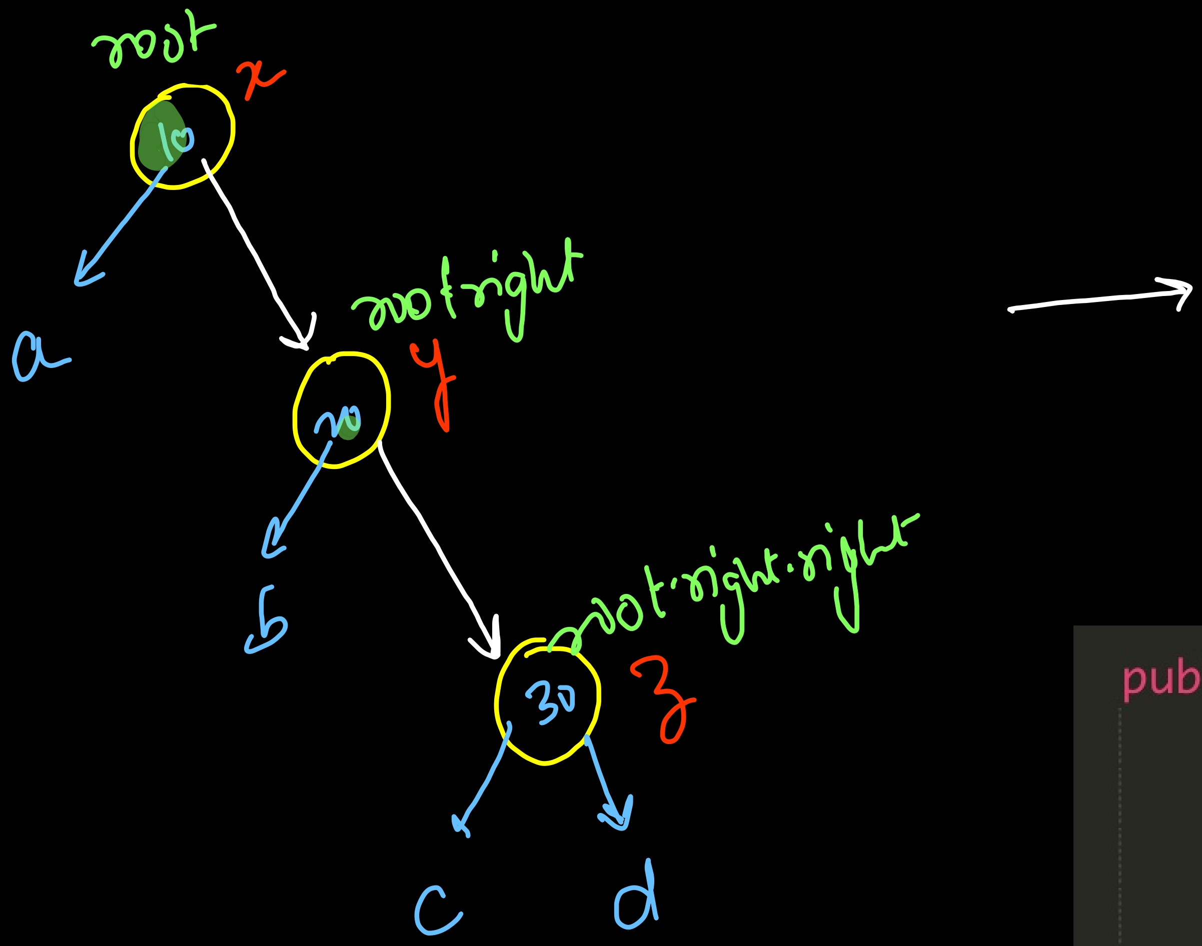


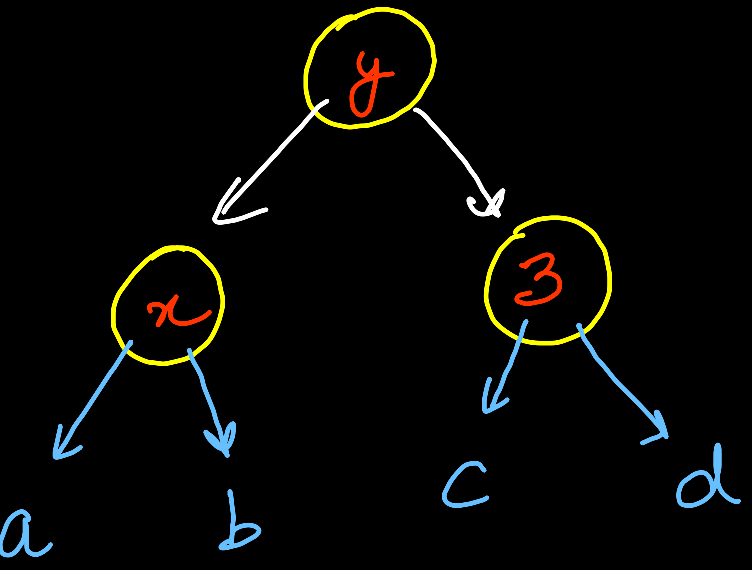


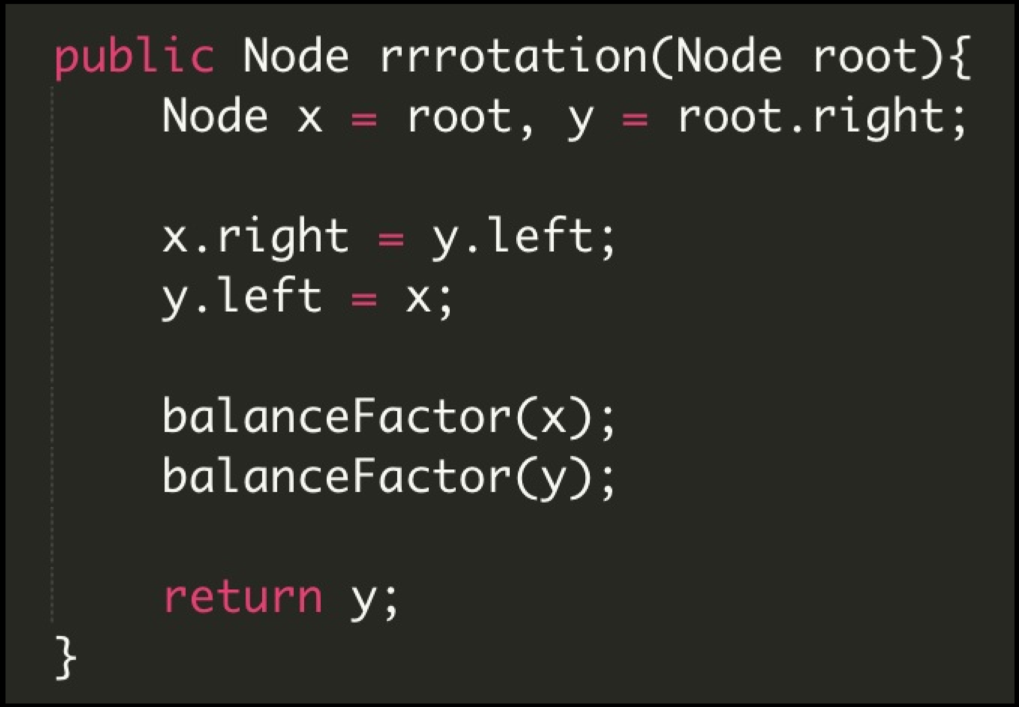
```java
public int balanceFactor(Node root){
    if(root == null) return 0;

    int lh = (root.left == null) ? 0 : root.left.height;
    int rh = (root.right == null) ? 0 : root.right.height;

    root.height = Math.max(lh, rh) + 1;
    return lh - rh;
}
```



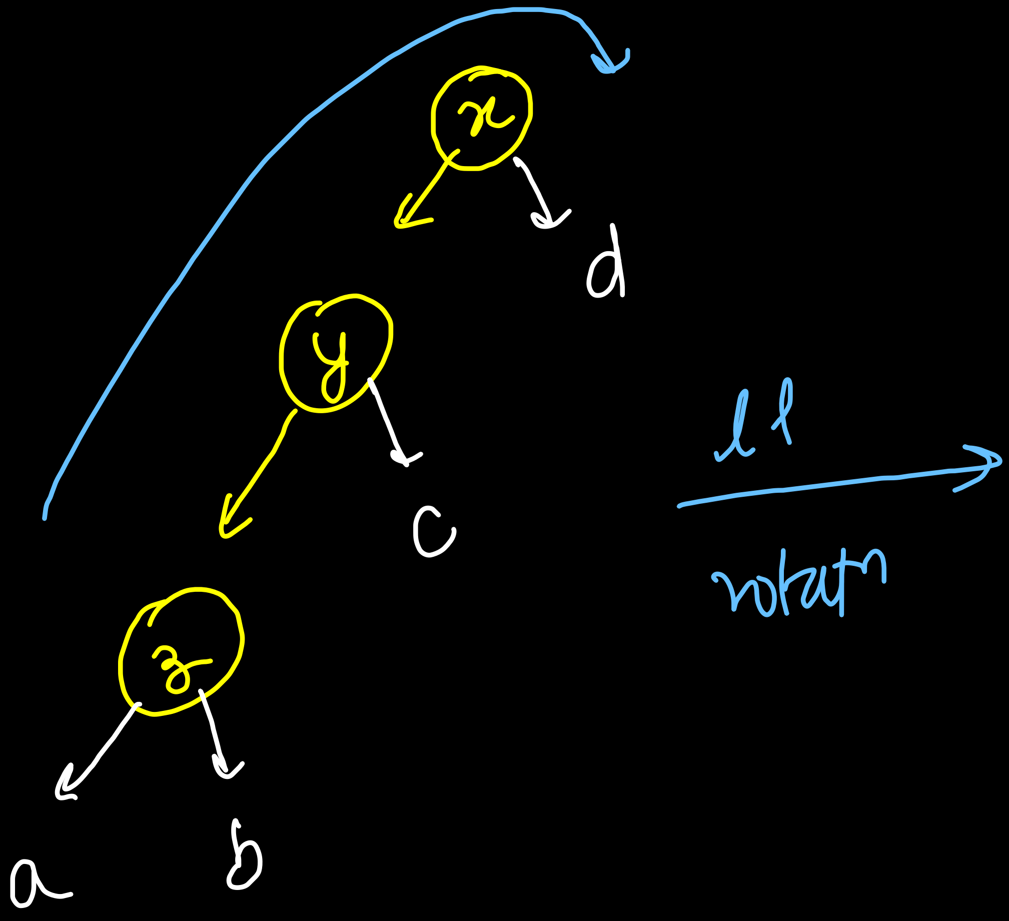
```java
public Node insertToAVL(Node root,int data)
{
    if(root == null) return new Node(target);
    if(root.data == data) return root;

    if(target < root.data)
        root.left = insertIntoBST(root.left, target);
    else root.right = insertIntoBST(root.right, target);

    int bf = balanceFactor(root);

    if(bf < -1)
    {
        // right skewed
        if(data > root.right.data)
            return rrrotation(root);
        else return rlrotation(root);
    }
    else if(bf > 1)
    {
        // left skewed
        if(data < root.left.data)
            return llrotation(root);
        else return lrrotation(root);
    }

    return root;
}
```

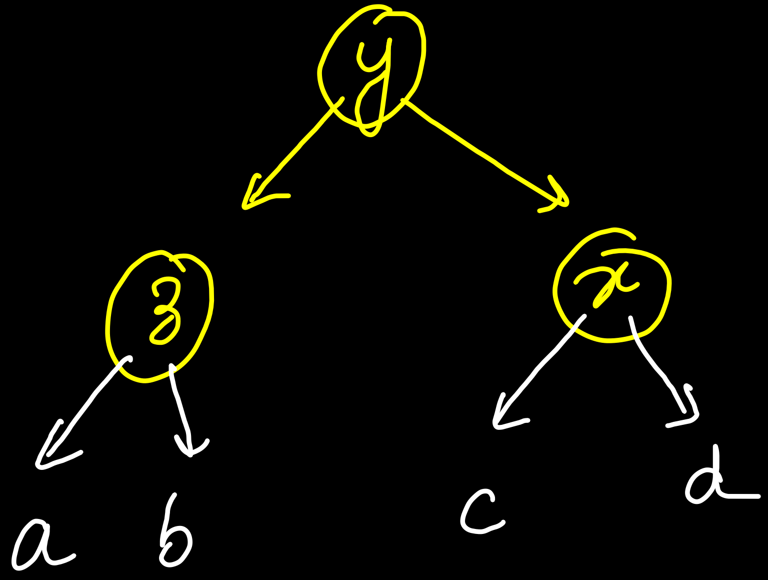


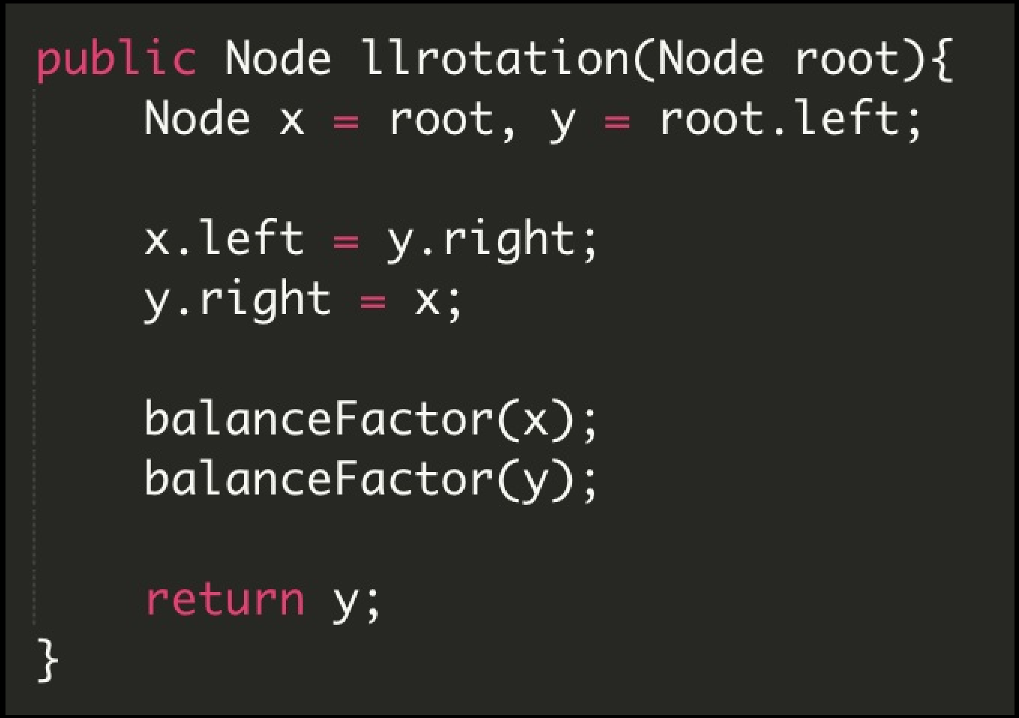




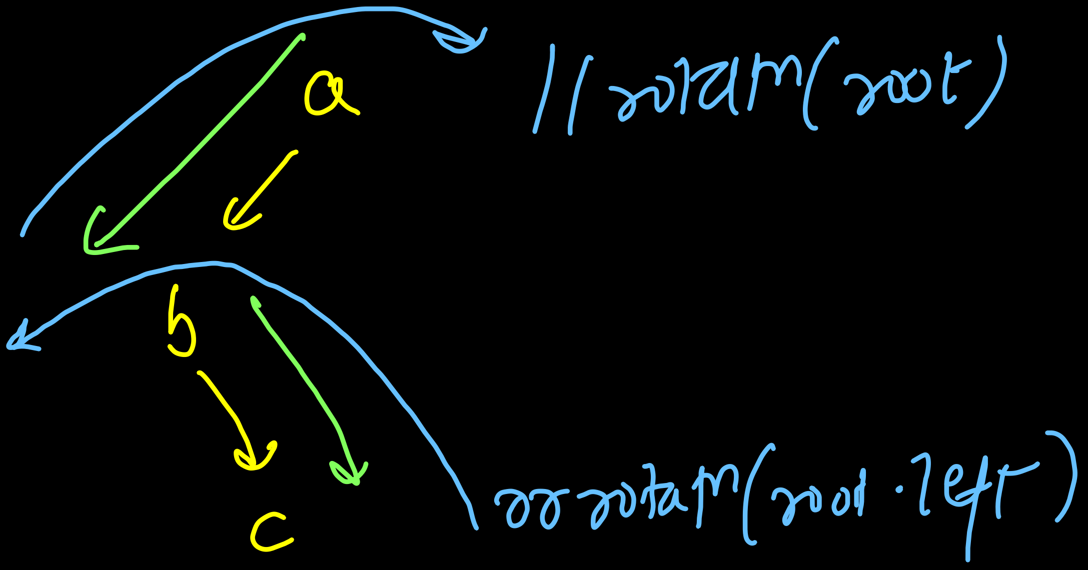
```java
public Node rrrotation(Node root){
    Node x = root, y = root.right;

    x.right = y.left;
    y.left = x;

    balanceFactor(x);
    balanceFactor(y);

    return y;
}
```

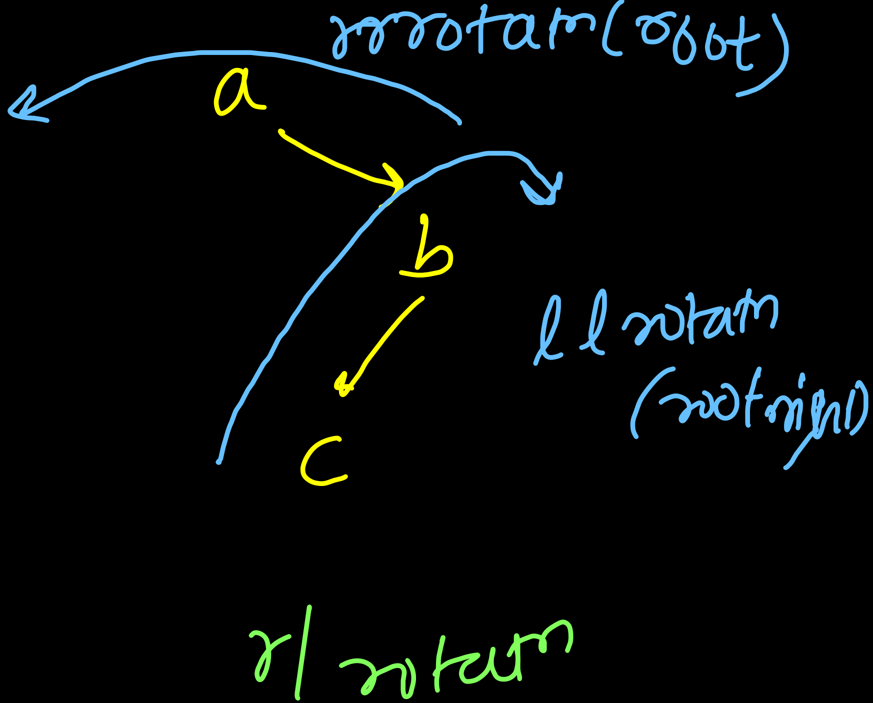


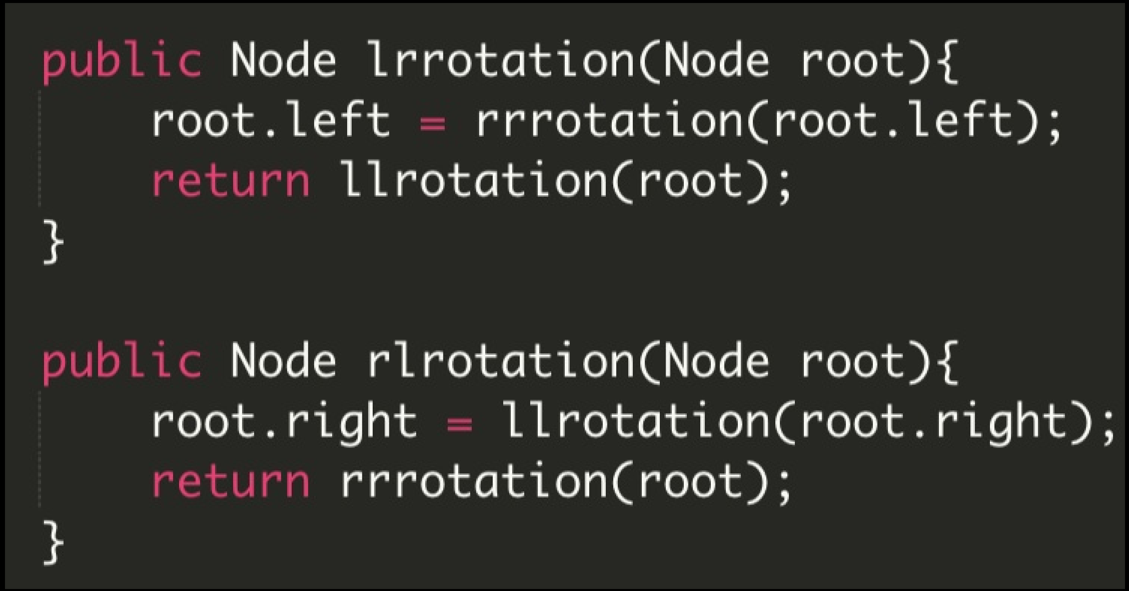




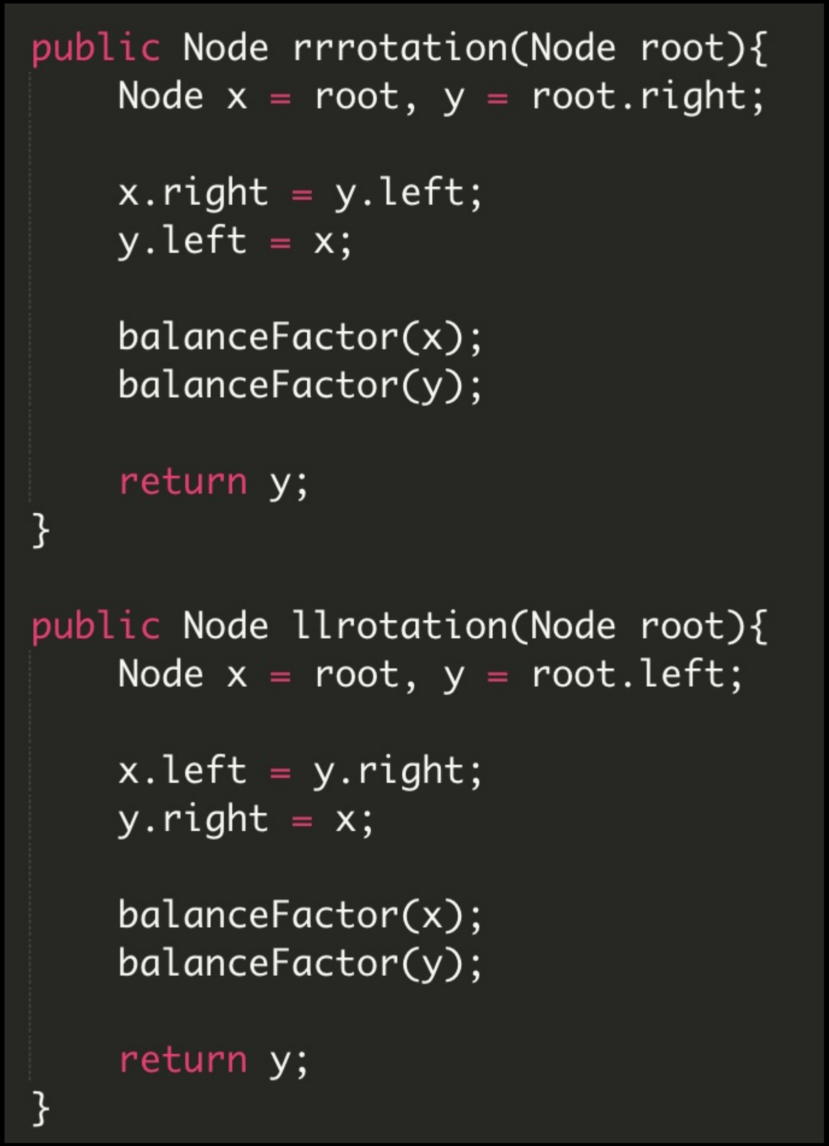
```
public Node llrotation(Node root){
    Node x = root, y = root.left;

    x.left = y.right;
    y.right = x;

    balanceFactor(x);
    balanceFactor(y);

    return y;
}
```



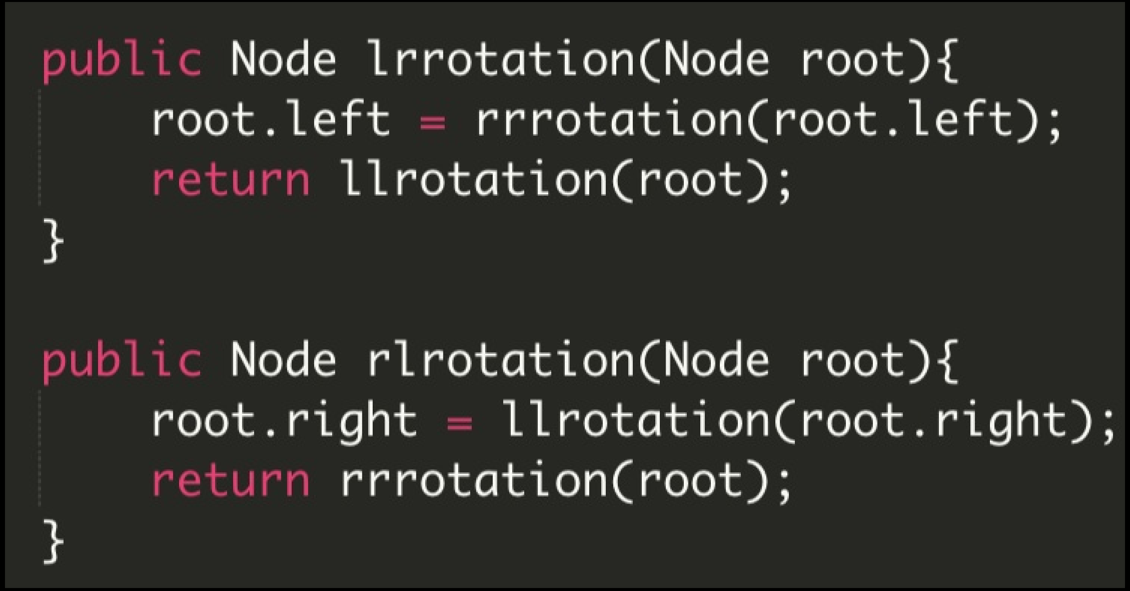




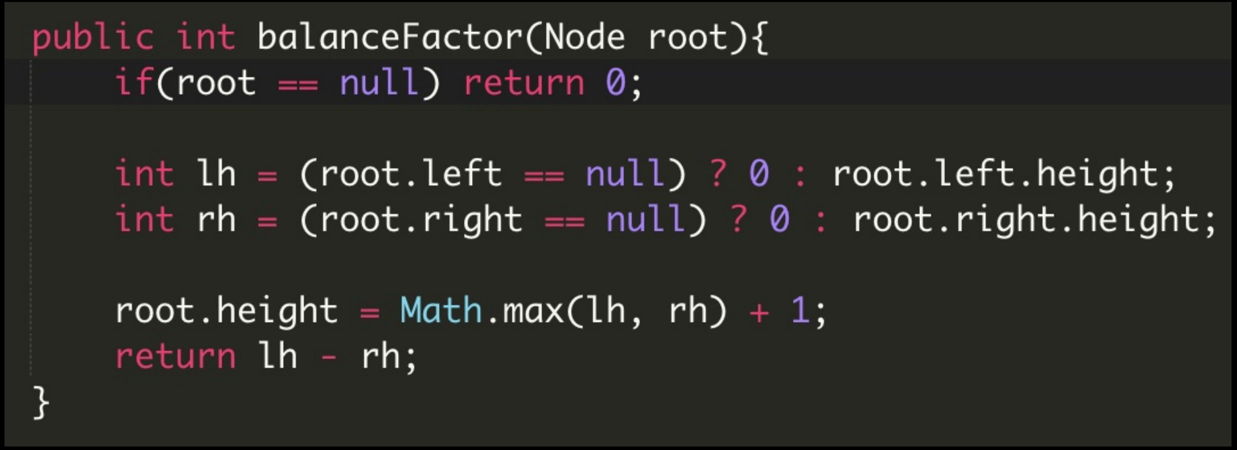
```
public Node lrrotation(Node root){
    root.left = rrrotation(root.left);
    return llrotation(root);
}

public Node rlrotation(Node root){
    root.right = llrotation(root.right);
    return rrrotation(root);
}
```



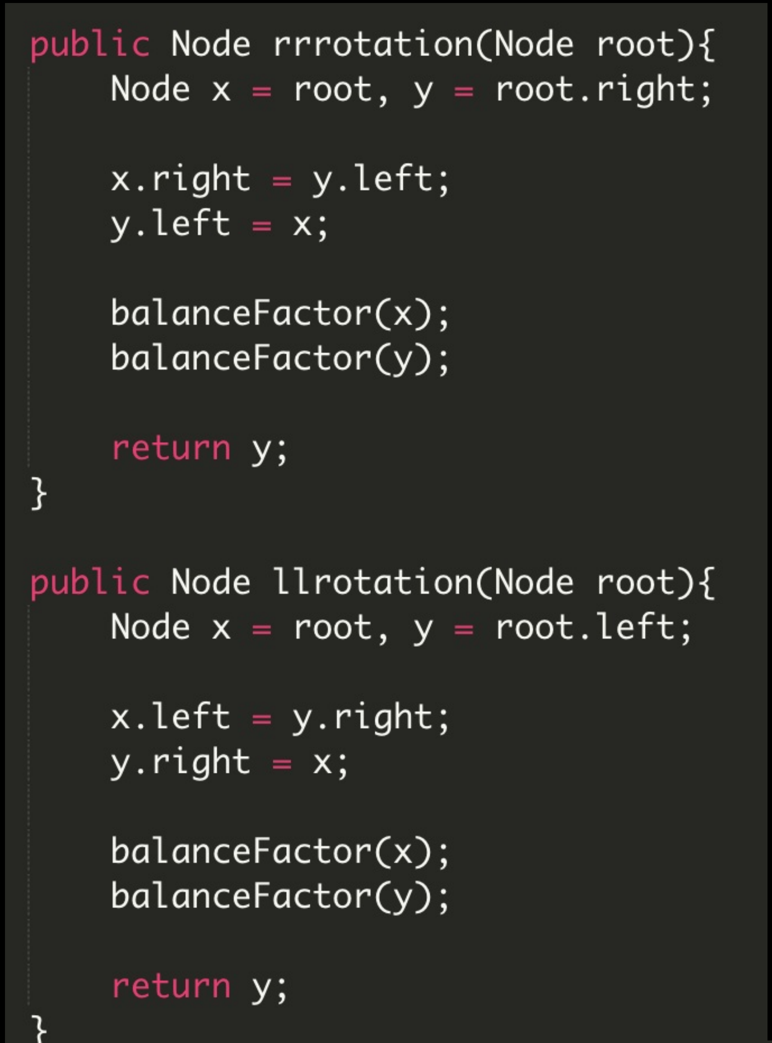
```
public Node rrrotation(Node root){
    Node x = root, y = root.right;

    x.right = y.left;
    y.left = x;

    balanceFactor(x);
    balanceFactor(y);

    return y;
}

public Node llrotation(Node root){
    Node x = root, y = root.left;

    x.left = y.right;
    y.right = x;

    balanceFactor(x);
    balanceFactor(y);

    return y;
}
```



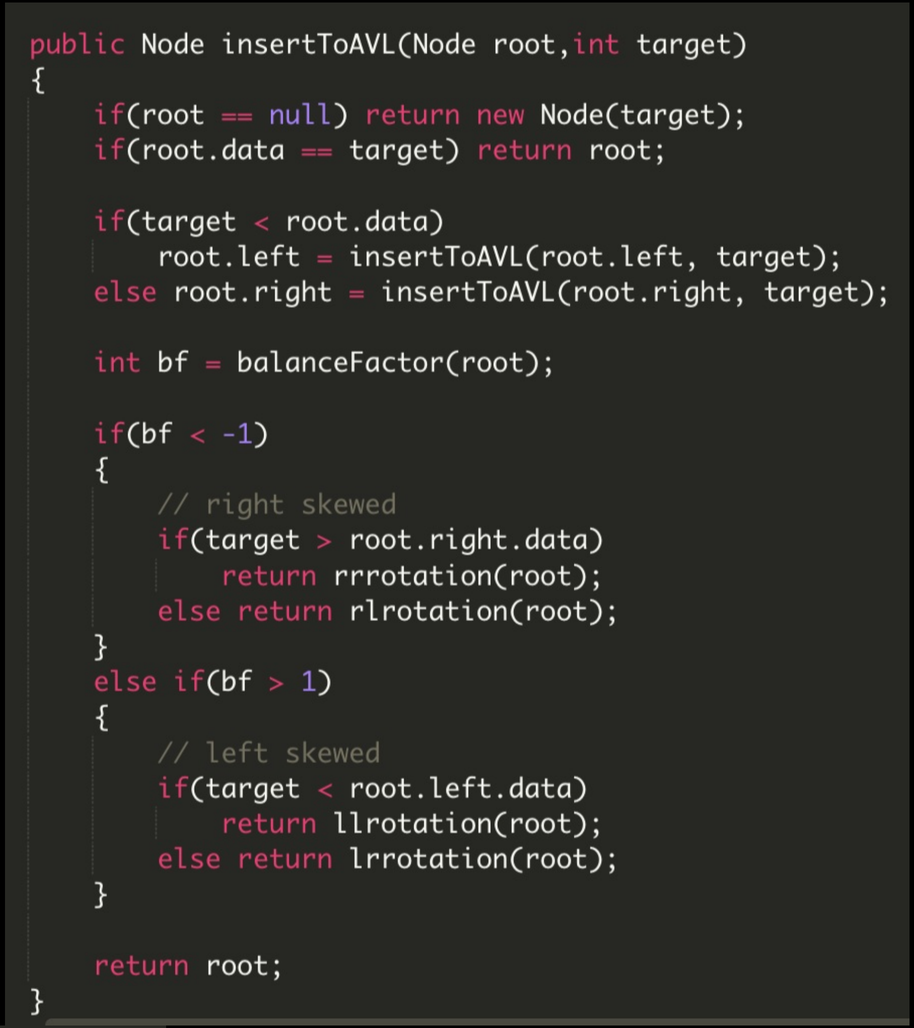
```
public Node lrrotation(Node root){
    root.left = rrrotation(root.left);
    return llrotation(root);
}

public Node rlrotation(Node root){
    root.right = llrotation(root.right);
    return rrrotation(root);
}
```



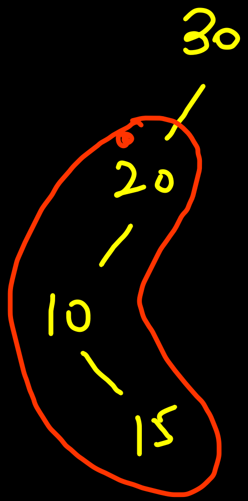
```
public int balanceFactor(Node root){
    if(root == null) return 0;

    int lh = (root.left == null) ? 0 : root.left.height;
    int rh = (root.right == null) ? 0 : root.right.height;

    root.height = Math.max(lh, rh) + 1;
    return lh - rh;
}
```

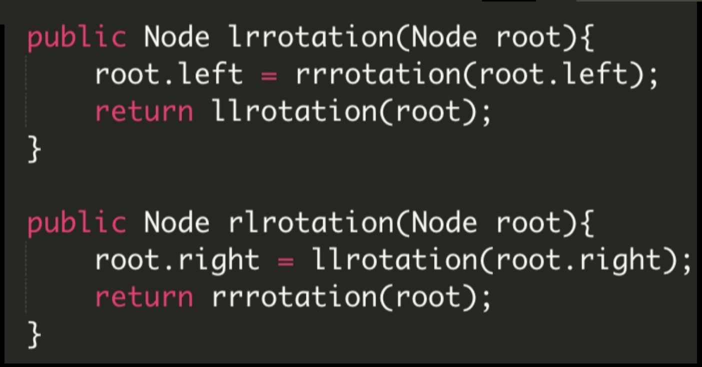


```
public Node rrrotation(Node root){
    Node x = root, y = root.right;

    x.right = y.left;
    y.left = x;

    balanceFactor(x);
    balanceFactor(y);

    return y;
}

public Node llrotation(Node root){
    Node x = root, y = root.left;

    x.left = y.right;
    y.right = x;

    balanceFactor(x);
    balanceFactor(y);

    return y;
}
```

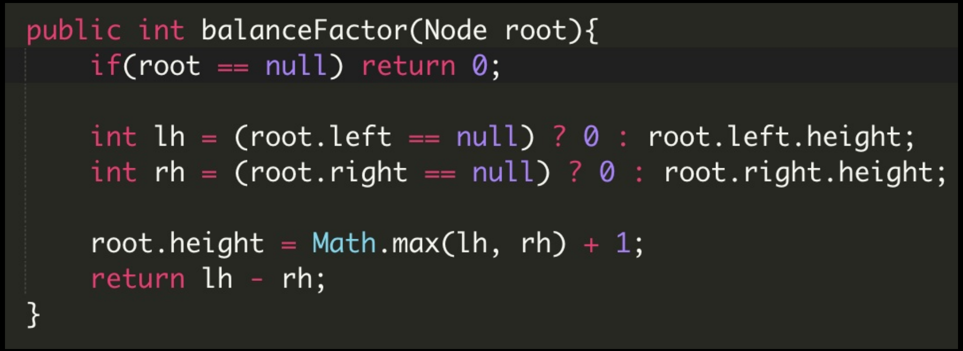


```
public Node insertToAVL(Node root,int target)
{
    if(root == null) return new Node(target);
    if(root.data == target) return root;

    if(target < root.data)
        root.left = insertToAVL(root.left, target);
    else root.right = insertToAVL(root.right, target);

    int bf = balanceFactor(root);

    if(bf < -1)
    {
        // right skewed
        if(target > root.right.data)
            return rrrotation(root);
        else return rlrotation(root);
    }
    else if(bf > 1)
    {
        // left skewed
        if(target < root.left.data)
            return llrotation(root);
        else return lrrotation(root);
    }

    return root;
}
```





```
public Node lrrotation(Node root){
    root.left = rrrotation(root.left);
    return llrotation(root);
}

public Node rlrotation(Node root){
    root.right = llrotation(root.right);
    return rrrotation(root);
}
```



```
public int balanceFactor(Node root){
    if(root == null) return 0;

    int lh = (root.left == null) ? 0 : root.left.height;
    int rh = (root.right == null) ? 0 : root.right.height;

    root.height = Math.max(lh, rh) + 1;
    return lh - rh;
}
```