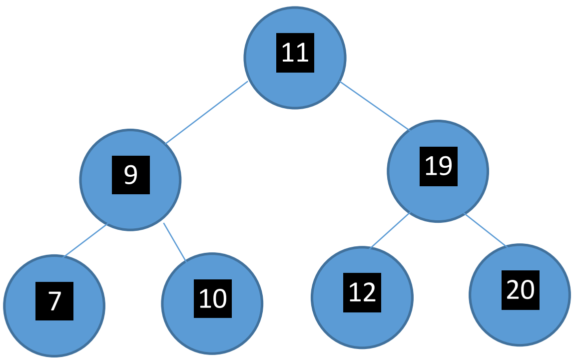
AVL TREE

V-1-(Intro):-

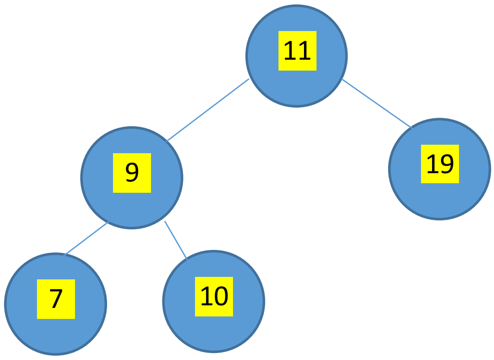
AVL TREE is named upon its investor (Adelson-Velsky & Landis) to reduce time complexity and it has self-balancing Algo.

V-2-(Balanced & UnBalanced Tree):-

Binary Search Tree(BST) has O (log n) time complexity in most of the operation but only if the tree is balanced like below:-



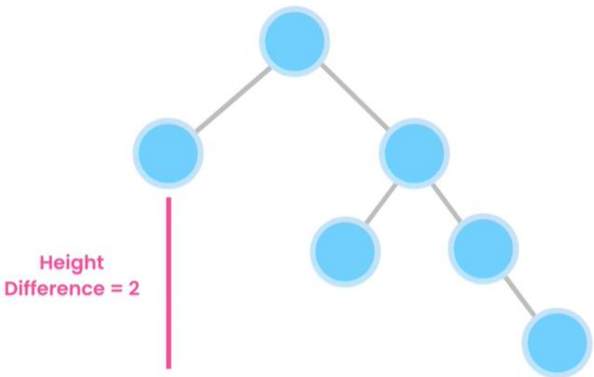
Pic-1



Pic-2

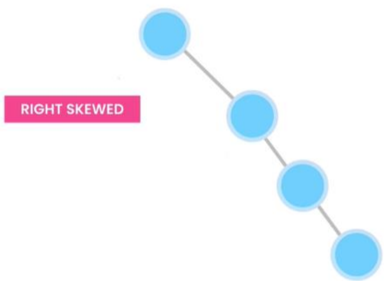
Looked at the picture-1 & picture-2 every node has two child or zero children for leaf node. In BST ***Height(left subtree) - Height(right subtree) <= 1***. That mean we can't have a long branch.

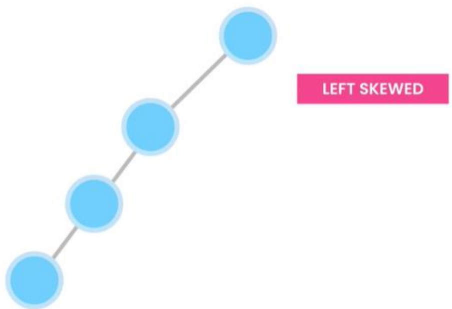
Picture -1 is an example of perfect tree(left and right subtree are full with children).



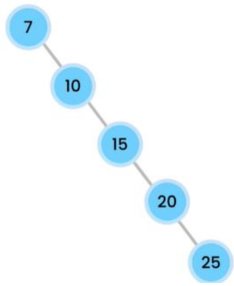
The picture showed beside is an unbalnced tree. Cause the height of left and right subtree difference is greater than 1.

There are more trees. Look at the picture below(worst binary tree looked like linked list):-

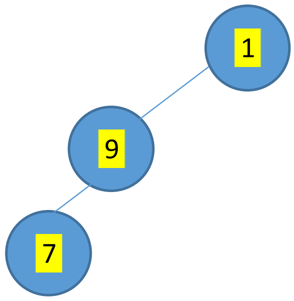




If we insert items in ascending order then the tree become right skewed binary tree.



If we insert items in descending order then the tree become left skewed binary tree.



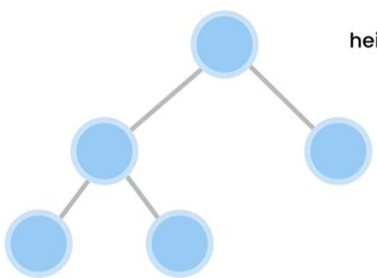
Even if BST (if not sorted) it became unbalanced.

SELF-BALANCING TREES

- AVL Trees (Adelson-Velsky and Landis)
- Red-black Trees
- B-trees
- Splay Trees
- 2-3 Trees

Self-Balancing Trees has property of balancing itself. Actually , there are many types of trees but we don't need to learn them all.

V-3-(Rotations):-



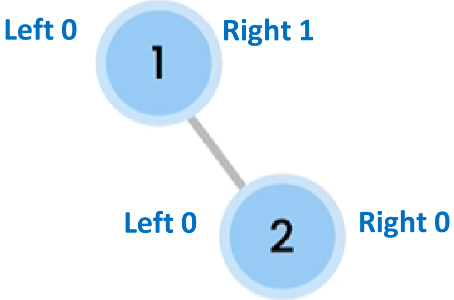
height(left) - height(right) <= 1

AVL tree is a special type of binary tree what automatically readjust themselves whenever we add nodes by considering that the left and right subtree height difference must be <= 1.

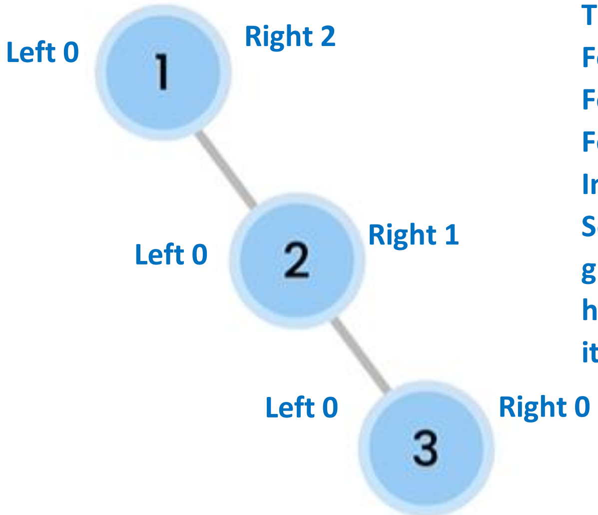
(4 Types) ROTATIONS

- Left (LL)
- Right (RR)
- Left-Right (LR)
- Right-Left (RL)

Illustrating imbalance in tree:-

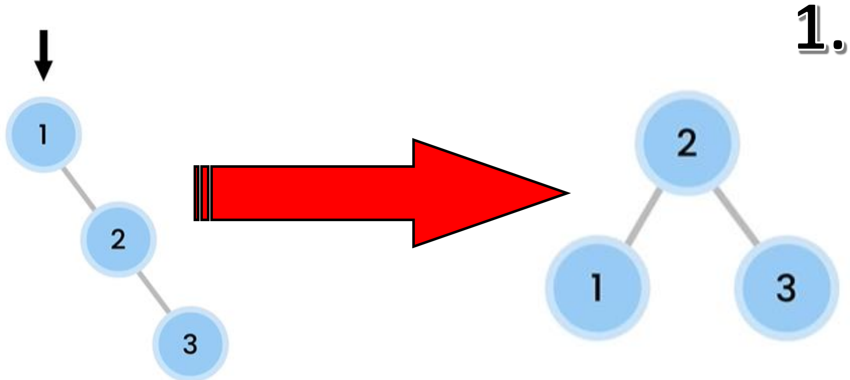


The beside picture tree is a balnce tree cause :-
For Node 2 , left subtree height = right subtree height = 0
For Node 1, left subtree height = 0 & right subtree height=1
In balance tree left subtree height – right subtree height <=1
So, that satisfy here.

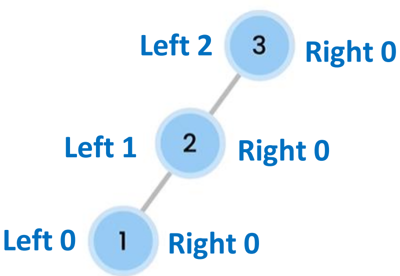


The beside picture tree is an unbalance tree cause :-
For Node 3 , left subtree height = right subtree height = 0
For Node 2, left subtree height = 0 & right subtree height=1
For Node 1, left subtree height = 0 & right subtree height=2
In balance tree left subtree height – right subtree height <=1
So, for Node 1 left and right subtree height differnce is greater than 1. So, the whole tree is unbalance. As this tree is heavy on right side. So, we have to do left rotations to make it balanced.

1. Left Rotation

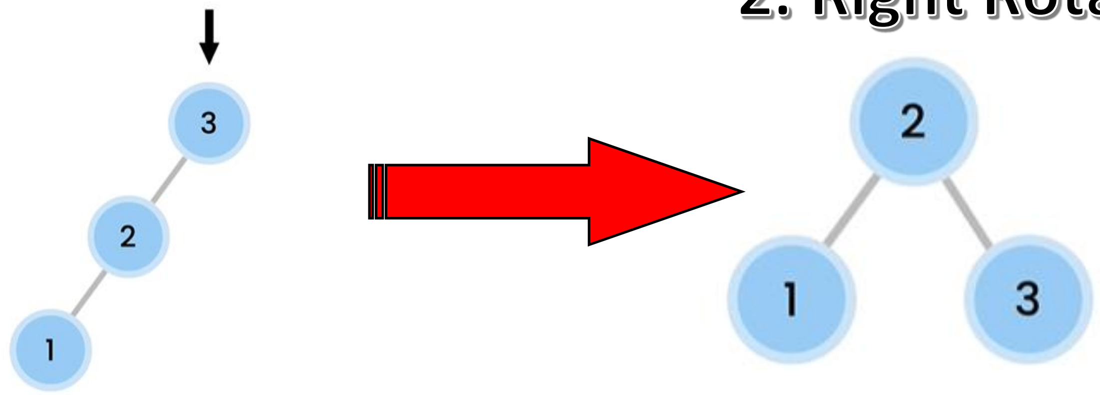


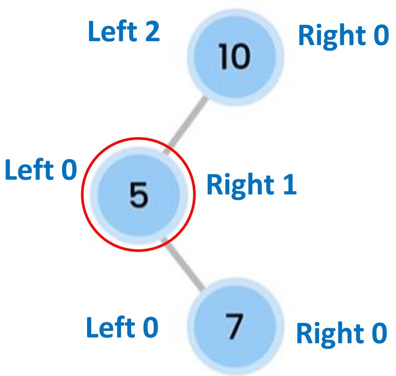
After Rotations one side goes up other side came down. That means height of one side increases and height of other side decreases.



The beside picture tree is an unbalance tree cause :-
For Node 1 , left subtree height = right subtree height = 0
For Node 2, left subtree height = 1 & right subtree height=0
For Node 3, left subtree height = 2 & right subtree height=0
In balance tree left subtree height – right subtree height <=1
So, for Node 3 left and right subtree height differnce is greater than 1. So, the whole tree is unbalance. As this tree is heavy on left side. So, we have to do right rotations to make it balanced.

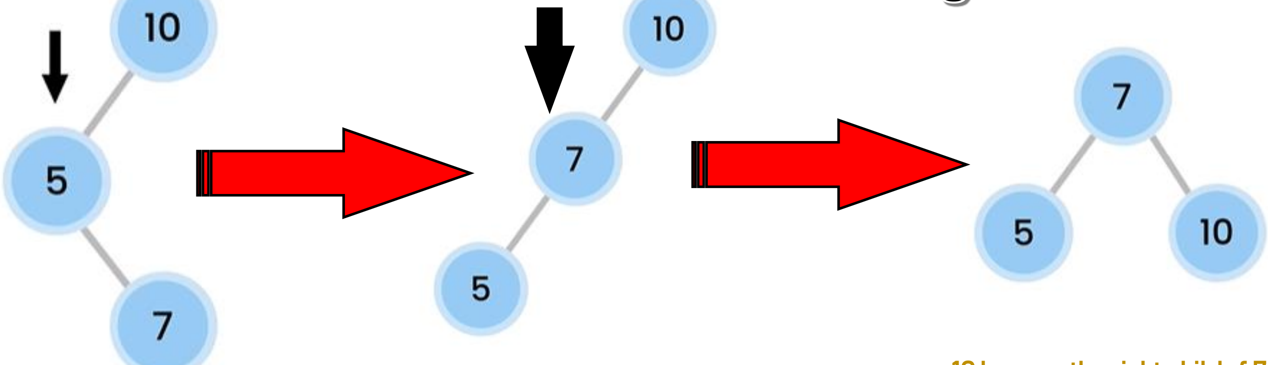
2. Right Rotation





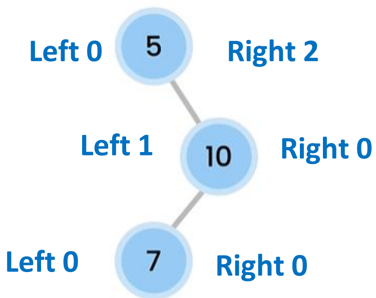
The beside picture tree is an unbalance tree cause :-
For Node 7 , left subtree height = right subtree height = 0
For Node 5, left subtree height = 0 & right subtree height=1
For Node 10, left subtree height = 2 & right subtree height=0
In balance tree left subtree height – right subtree height <=1
So, for Node 10 left and right subtree height differnce is greater than 1. As, imbalance is happening in left child right subtree. So, we have to do left rotation followed by right rotations to make it balanced.(left-right rotations)

3. Left-Right Rotation



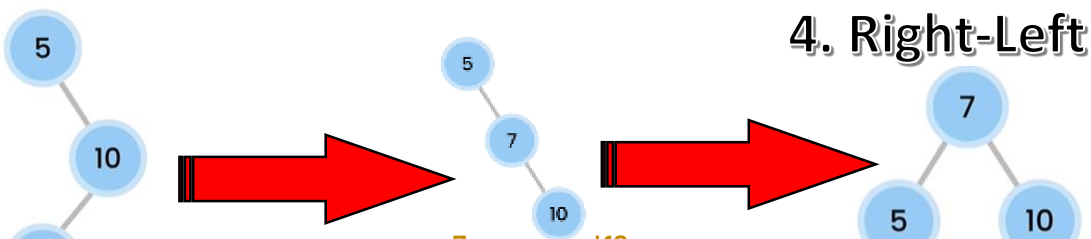
5 goes down and 7 goes up.

10 became the right child of 7.



The beside picture tree is an unbalance tree cause :-
For Node 7 , left subtree height = right subtree height = 0
For Node 10, left subtree height = 1 & right subtree height=0
For Node 5, left subtree height = 0 & right subtree height=2
In balance tree left subtree height – right subtree height <=1
So, for Node 5 left and right subtree height differnce is greater than 1. As, imbalance is happening in right child left subtree. So, we have to do right rotation followed by left rotations to make it balanced.(right-left rotations)

4. Right-Left Rotation



7 goes up and 10 goes up.

5 became the left child of 7.

N:B:- Easy Thing to identify where to do left-right rotation is if we are doing a left rotation and after that the basic condition of a tree being balanced (left subtree height – right subtree height <=1) not satisfied then we have to do right rotation afterwards. On the other hand, if we are doing a right rotation and after that the basic condition of a tree being balanced (left subtree height – right subtree height <=1) not satisfied then we have to do left rotation afterwards .

V-4-(AVL TREES):-

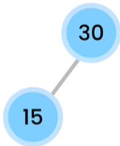
Avl Tree Working Simulations(How it works):-

1. Insert 30.

Ans: As it is the first element to be inserted, It is considered as root.

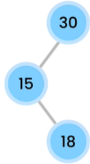
2. Insert 15.

Ans: As 30 is already inserted & it is the root node. And 15 is less than 30. So, it should be inserted as the left child of root node 30.

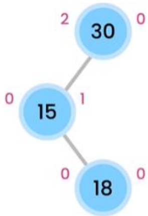


3. Insert 18.

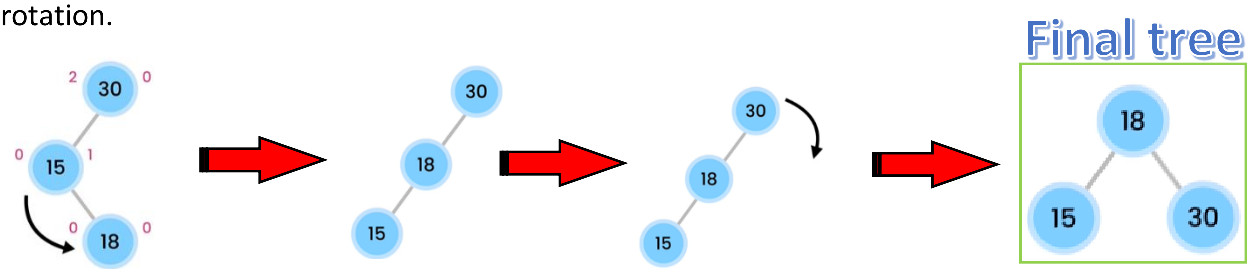
Ans: Now, 15>**18**<30. So, **18** should be right child of node 15.



But the tree is unbalanced here. Cause, for node 30 ***left subtree height – right subtree height >1.***The picture below has the calculation of height for each node.



As the tree is imbalanced is left child right subtree. So, we have to implement left-right rotation.

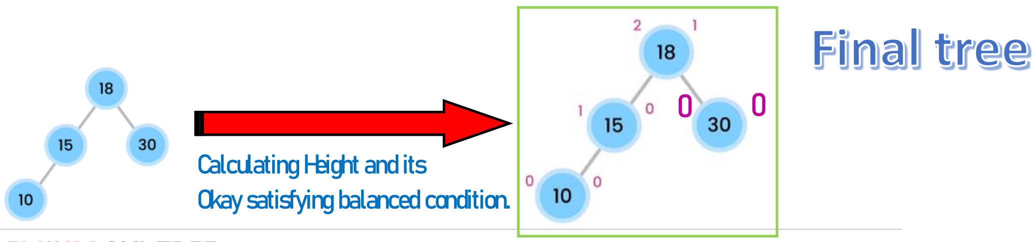


15 goes down and 18 goes up.

As the tree become left –skewd , we have to perform right rotation to make it balanced. So, 30 goes down and became the right child of node 18.

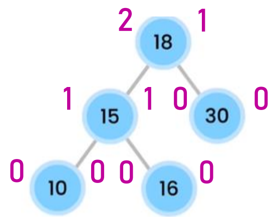
4. Insert 10.

Ans: As 10<18 & 10<15.



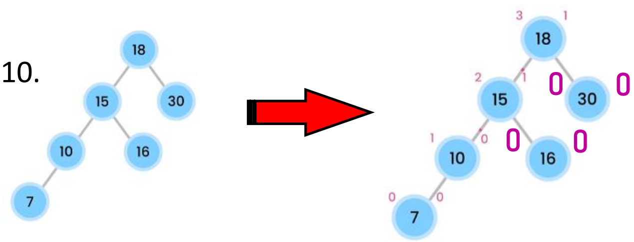
5. Insert 16.

Ans: As 16<18 & 16>15. So, 16 should be the right child of node 15. (It's still balanced)



6. Insert 7.

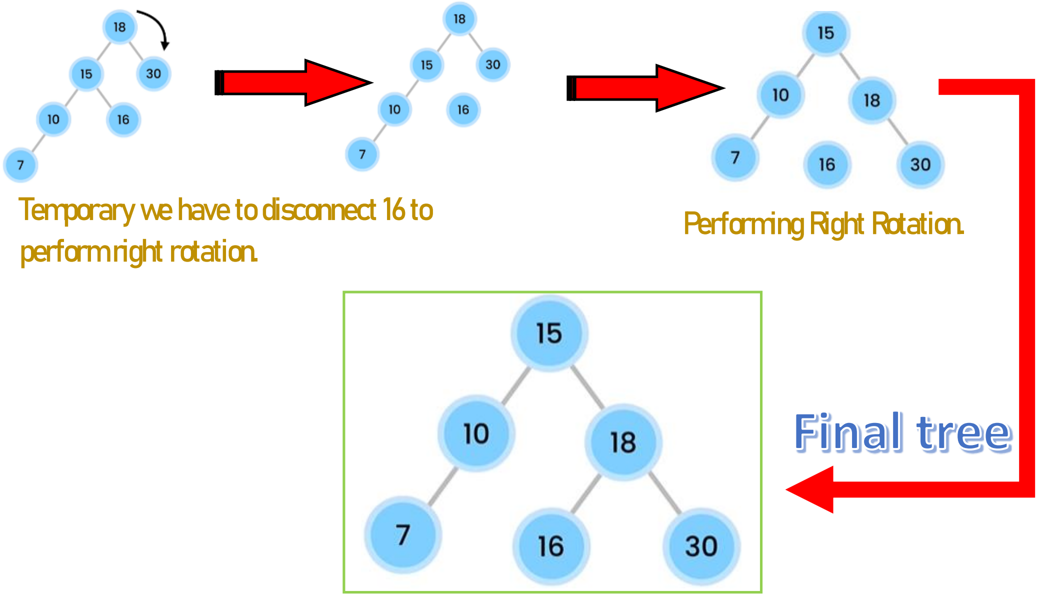
Ans: As 7<18, 7<15,7<10.



Calculating Height and it breaks The balanced condition for node 18.

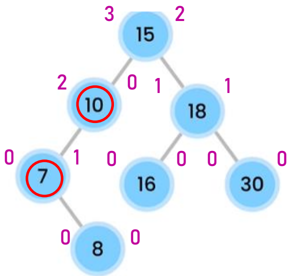
3-1 = 2 >1.

The imbalance occurs in left child , left subtree. So, we have to do right rotation. It is quite tricky.



As 16>15 but 16<18. So, 16 should be the left child of node 18.

7. Insert 8.



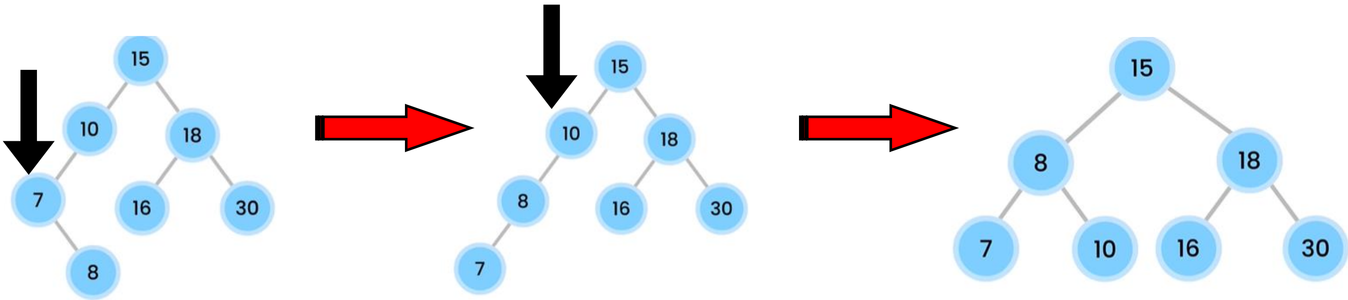
Ans: As, 8<15, 8<10, 8>7.

The imbalance

Occur in left child

Right subtree of

root node 15. So, We have to perform left-right rotation.



7 goes down 8 goes up.

10 goes down and became the right child of node 8 & 8 became the left child of root node 15.

V-5-(AVL Rotation Exercise):-

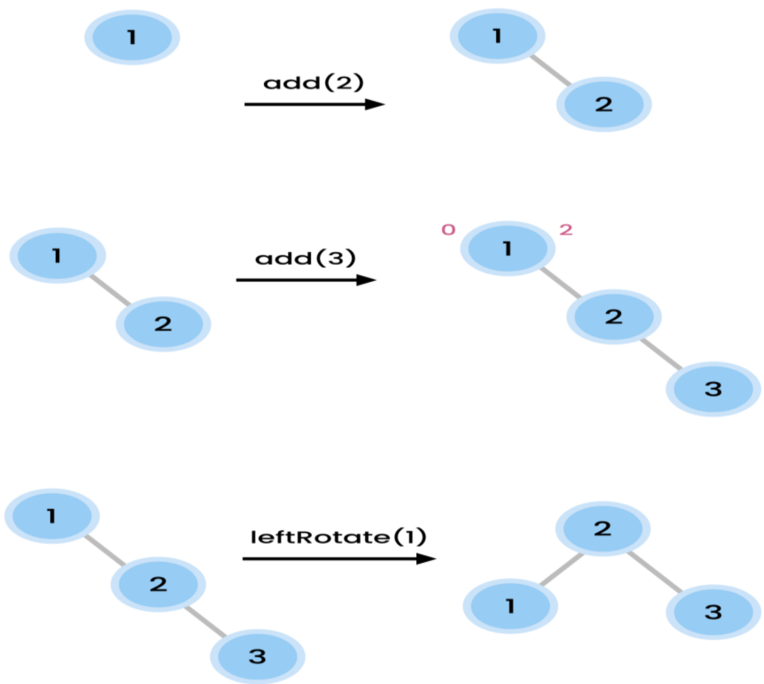
AVL Rotations

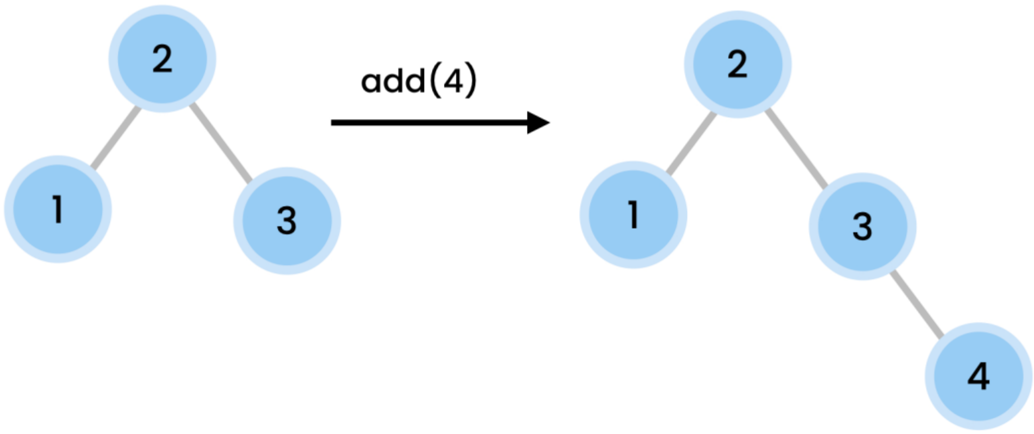
Add the following set of numbers to an AVL tree. For each set, start with
an empty tree. Draw the tree at each step and show the type of rotations
required to re-balance the tree. Solutions are on the next page.

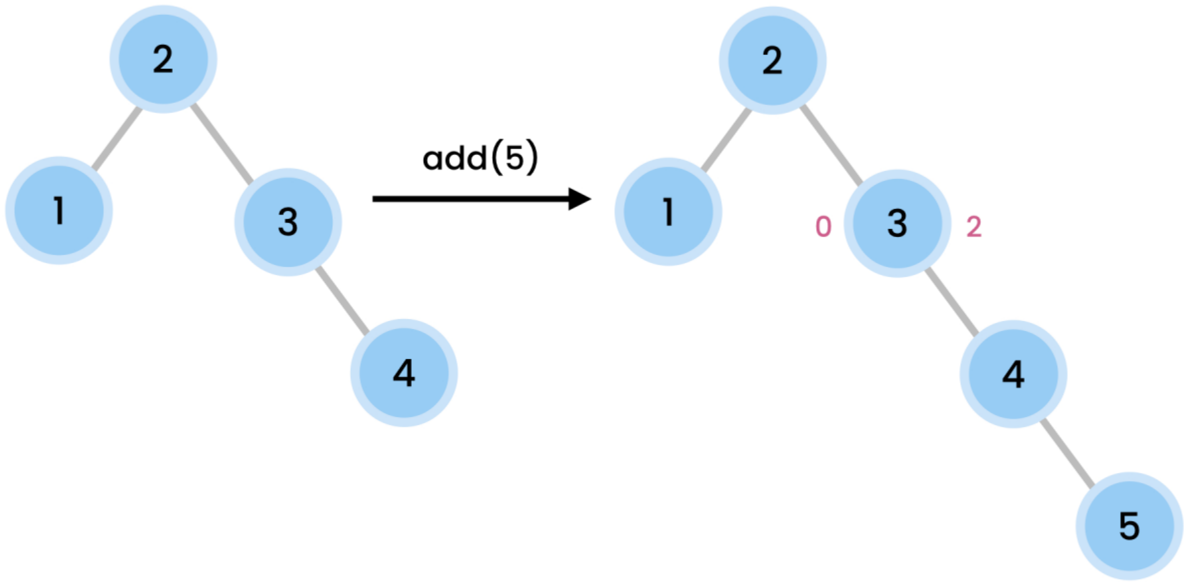
Set 1: (1, 2, 3, 4, 5)
Set 2: (5, 10, 3, 12, 15, 14)
Set 3: (12, 3, 9, 4, 6, 2)

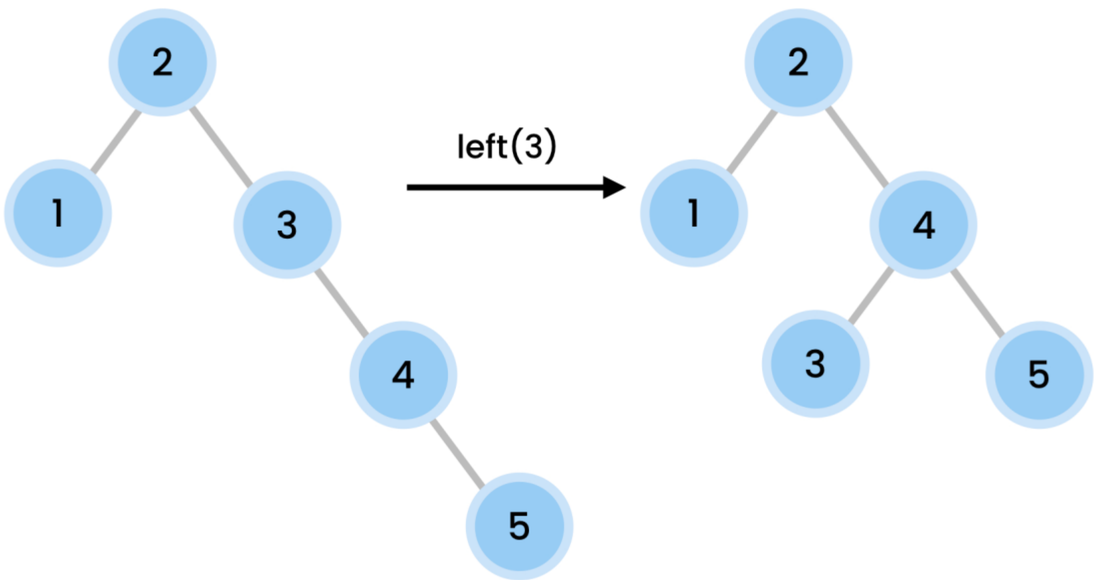
You can use the following tool to visualize an AVL tree:
https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

Solution to Set 1 (1, 2, 3, 4, 5)

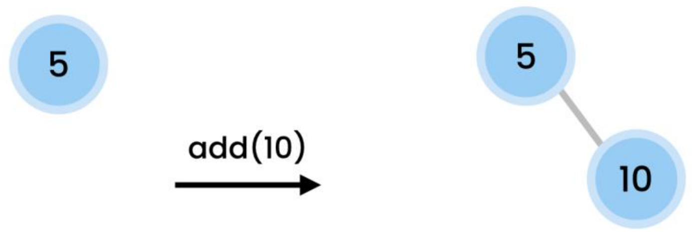


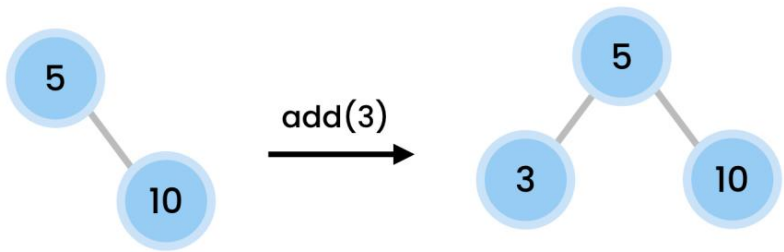

add(4)

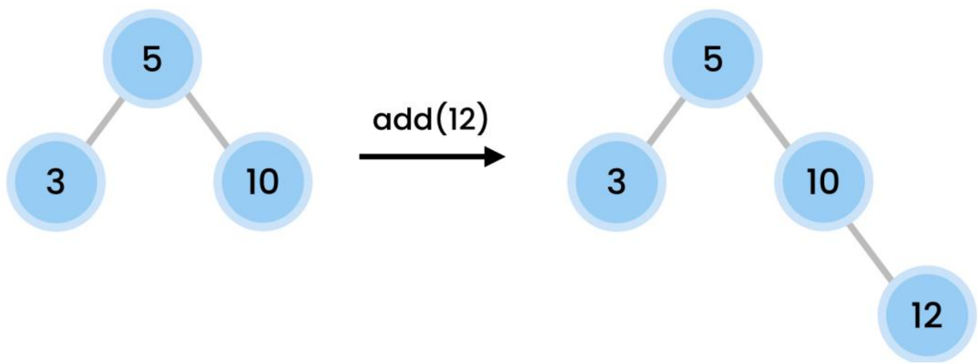

add(5)
0
2

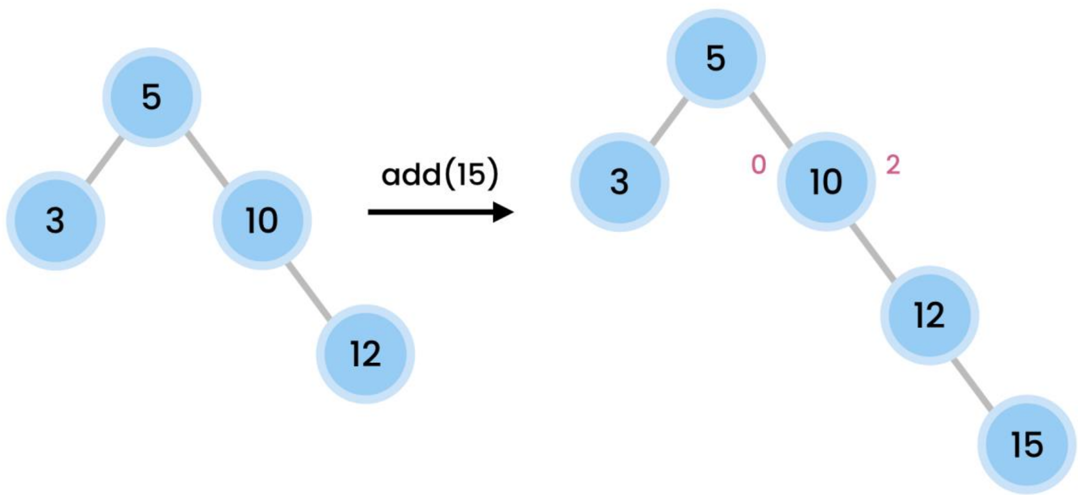

left(3)

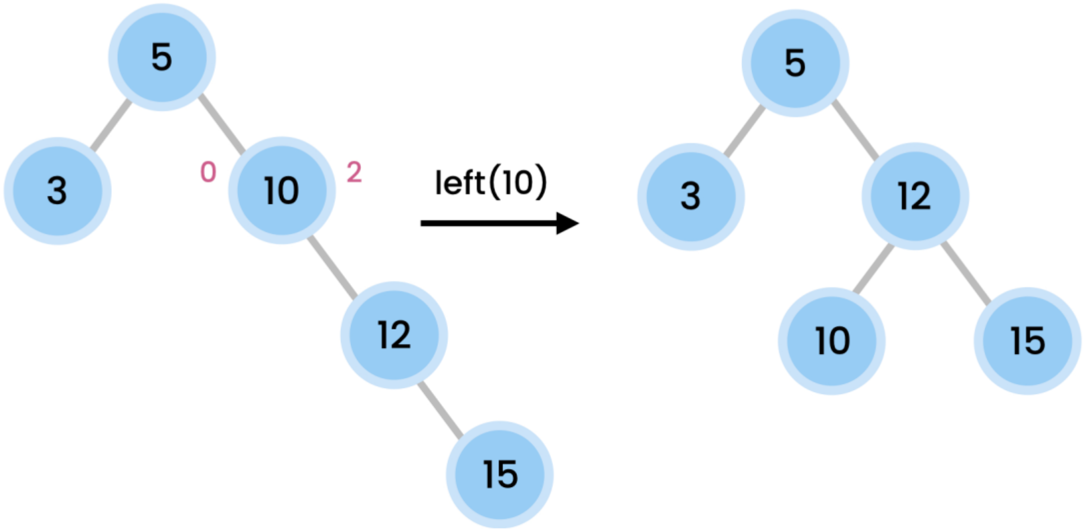
Solution to Set 2 (5, 10, 3, 12, 15, 14)

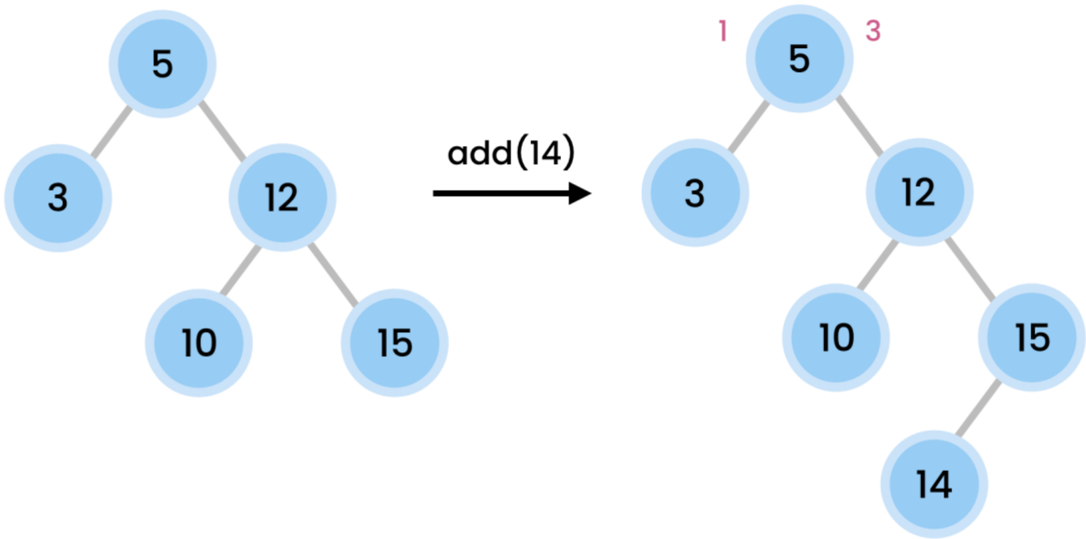


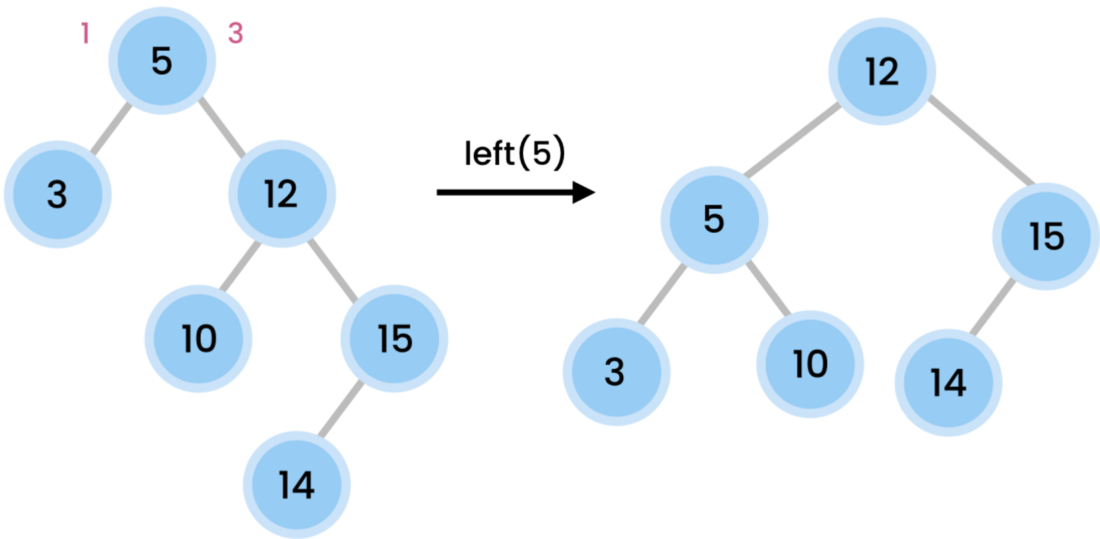




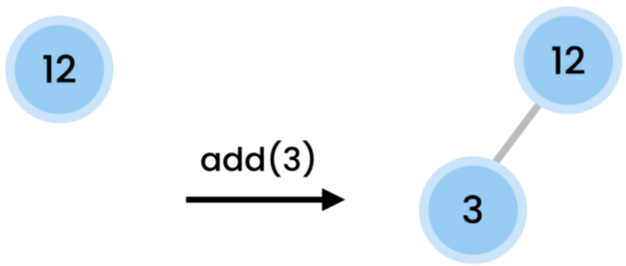


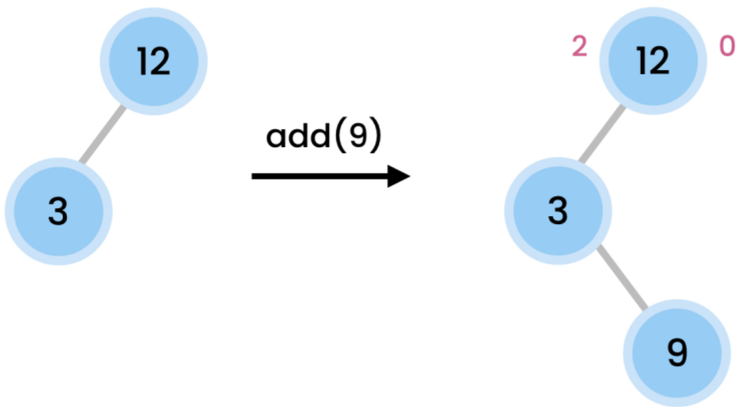

5
3
0
10
2
12
15
left(10)
5
3
12
10
15

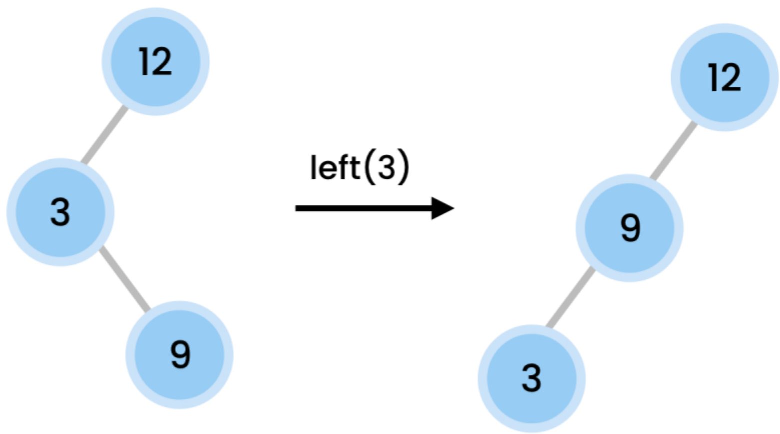

5
3
12
10
15
add(14)
1
5
3
3
12
10
15
14

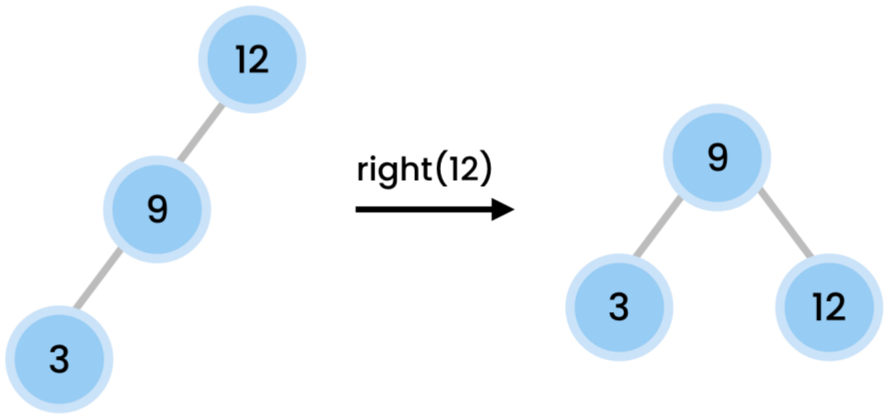

1
5
3
3
12
10
15
14
left(5)
12
5
15
3
10
14

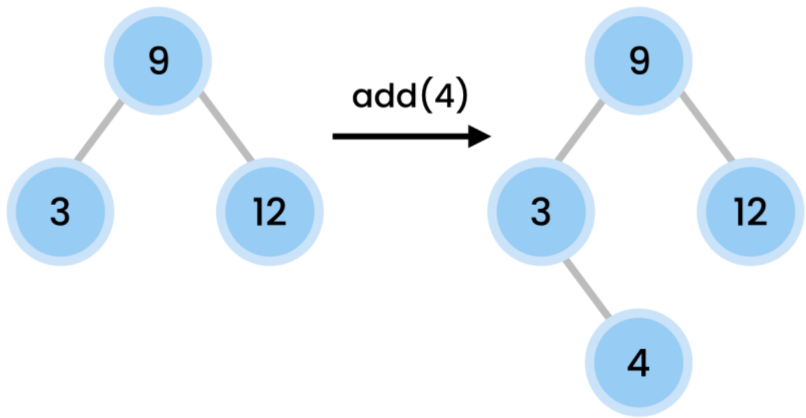
Solution to Set 3 (12, 3, 9, 4, 6, 2)

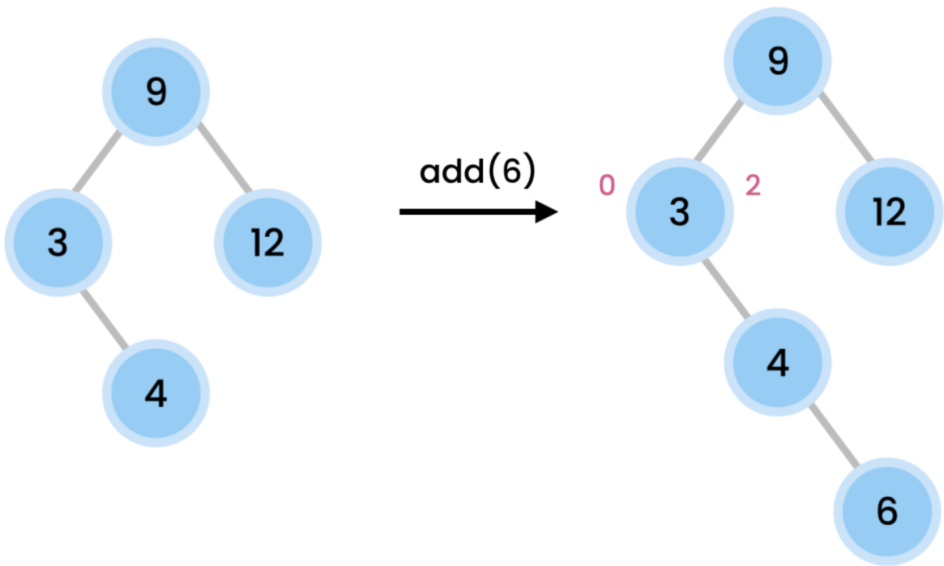


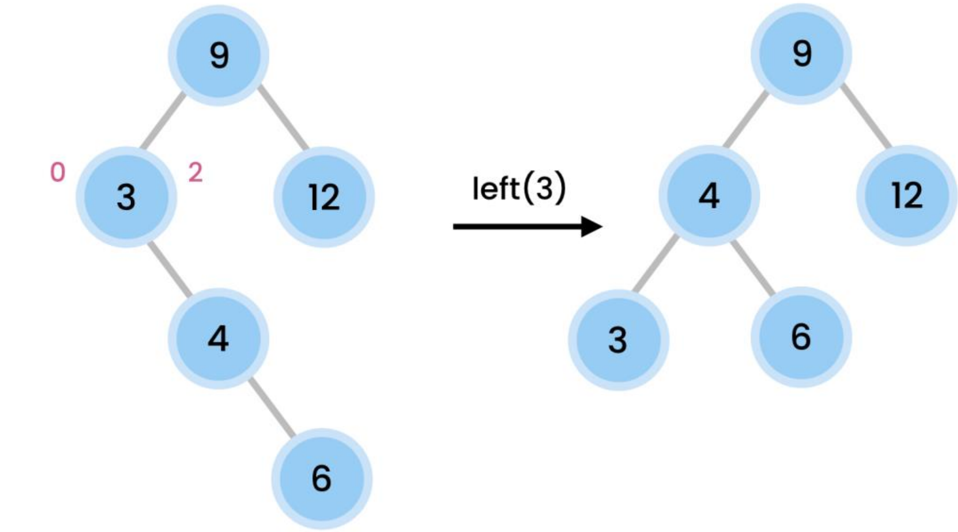


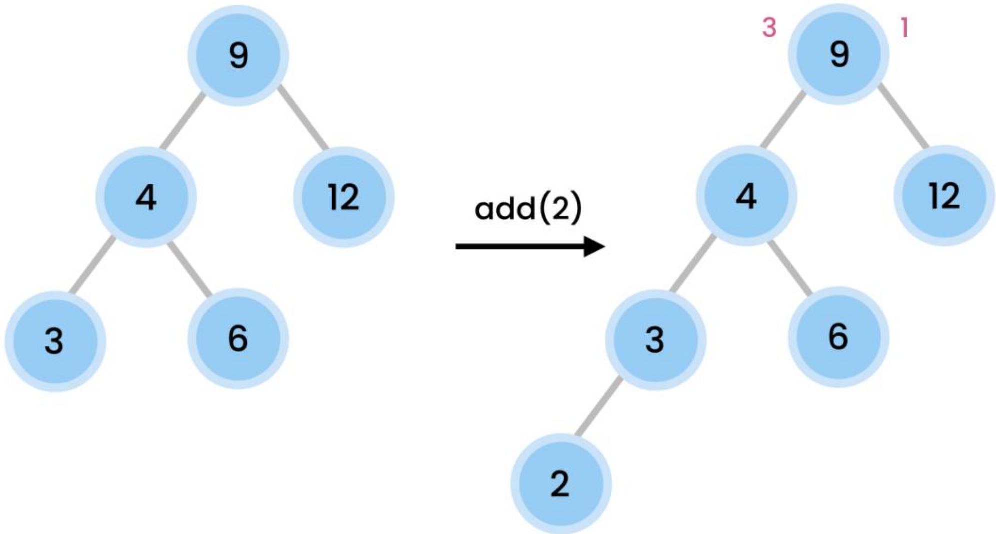


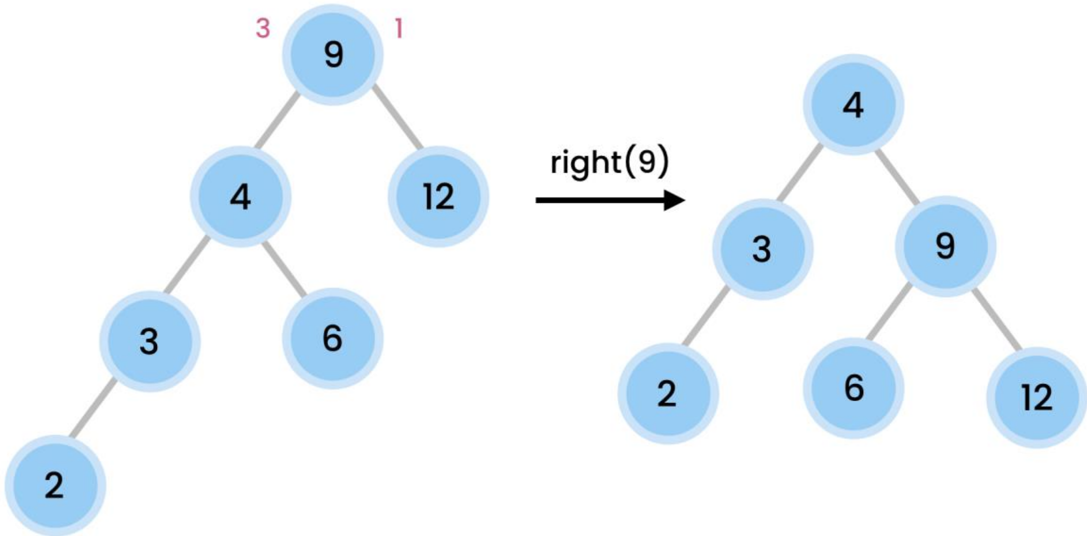

12
9
3
right(12)
9
3
12


9
3
12
add(4)
9
3
12
4


9
3
12
4
add(6)
9
0
3
2
12
4
6


0
2
left(3)


add(2)
3
1


3
1
right(9)

V-6-(AVL Tree Building):-

```
// AVLTree
// AVLNode
// insert() - recursion
```

V-(7+8)-(Insert Method+height+Basic Class):-

```
public class AVLTree {
    private class AVlNode{
      private int height;
      private int value;
      private AVlNode leftchild;
      private AVlNode rightchild;
      public AVlNode(int value){
          this.value=value;
      }
      @Override
      public String toString(){
          return "Value= "+this.value;
      }
    }
    private AVlNode root;
    public void insert(int value){
        root = insert(root,value);
    }
    public AVlNode insert(AVlNode root,int value){
        if(root==null)
            return new AVlNode(value);
        if(value<root.value)

root.leftchild=insert(root.leftchild,value);
        else

root.rightchild=insert(root.rightchild,value);
        root.height =
Math.max(height(root.leftchild),height(root.rightchild)
)+1;
        return root;
    }
    private int height(AVlNode node){
    return (node==null) ? -1: node.height;
    }
}
```

V-(10+11)-(Balance tree checking+Updated Codes of AVLTree Class):-

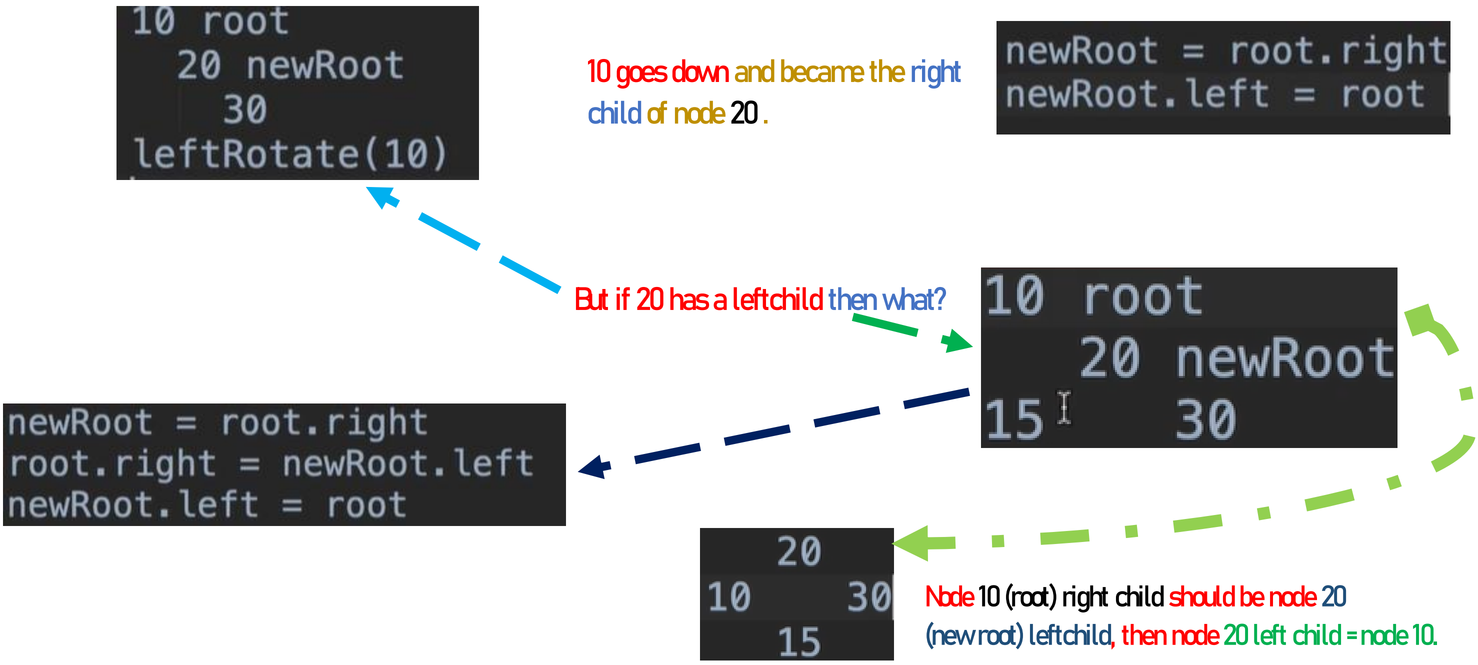
```
public void insert(int value){
    root = insert(root,value);
}
public AVlNode insert(AVlNode root,int value){
    if(root==null)
        return new AVlNode(value);
    if(value<root.value)
        root.leftchild=insert(root.leftchild,value);
    else
        root.rightchild=insert(root.rightchild,value);
    root.height = Math.max(height(root.leftchild),height(root.rightchild))+1;
    var balancefactor = balancefactor(root);
    if(isLeftHeavy(root))
        System.out.println(root.value+" is leftHeavy");
    if(isRightHeavy(root))
        System.out.println(root.value+" is rightHeavy");
    return root;
}
private boolean isLeftHeavy(AVlNode node){
    return balancefactor(node)>1;
}
private boolean isRightHeavy(AVlNode node){
    return balancefactor(node)<-1;
}
private int balancefactor(AVlNode node){
    // for empty tree it is balanced already
    // otherwise calculate the differences of left and right subtree and return
    return (node==null)? 0:height(node.leftchild)-height(node.rightchild);
}
private int height(AVlNode node){
return (node==null) ? -1: node.height;
}
```

V-(12+13)-(Detect Rotation on tree+Updated Codes of AVLTree Class):-

```
public class AVLTree {
    private class AVlNode{
      private int height;
      private int value;
      private AVlNode leftchild;
      private AVlNode rightchild;
      public AVlNode(int value){
          this.value=value;
      }
      @Override
      public String toString(){
          return "Value= "+this.value;
      }
    }
    private AVlNode root;
    public void insert(int value){
        root = insert(root,value);
    }
    public AVlNode insert(AVlNode root,int value){
        if(root==null)
            return new AVlNode(value);
        if(value<root.value)
```

```java
                root.leftchild=insert(root.leftchild,value);
            else
                root.rightchild=insert(root.rightchild,value);
            root.height = Math.max(height(root.leftchild),height(root.rightchild))+1;
            balance(root);
            return root;
        }
        private void balance(AVlNode root){
            if(isLeftHeavy(root)){//left child,right-subtree te balance harale left rotate
on root.leftchild then right rotate on root
                if(balancefactor(root.leftchild)<0){
                    System.out.println("Left Rotate "+root.leftchild.value);
                }
                System.out.println("Right Rotate "+root.value);
            }
            if(isRightHeavy(root)){//right child ,left sub-tree te balance harale hole
right rotation on root.rightchild, then left rotate on root
               if(balancefactor(root.rightchild)>0){
                   System.out.println("Right Rotate "+root.rightchild.value);
               }
                System.out.println("Left Rotate "+root.value);
            }
        }
        private boolean isLeftHeavy(AVlNode node){
            return balancefactor(node)>1;
        }
        private boolean isRightHeavy(AVlNode node){
            return balancefactor(node)<-1;
        }
        private int balancefactor(AVlNode node){
            // for empty tree it is balanced already
            // otherwise calculate the differences of left and right subtree and return
            return (node==null)? 0:height(node.leftchild)-height(node.rightchild);
        }
        private int height(AVlNode node){
        return (node==null) ? -1: node.height;
        }
}
```

V-(12+13)-(Detect Rotation on tree+Updated Codes of AVLTree Class):-



V-(14+15)-(Performing Rotation on tree+Updated Codes of AVLTree Class):-

```
public class AVLTree {
    private class AVlNode{
      private int height;
      private int value;
      private AVlNode leftchild;
      private AVlNode rightchild;
      public AVlNode(int value){
          this.value=value;
      }
      @Override
      public String toString(){
          return "Value= "+this.value;
      }
    }
    private AVlNode root;
    public void insert(int value){
        root = insert(root,value);
    }
    public AVlNode insert(AVlNode root,int value){
        if(root==null)
            return new AVlNode(value);
        if(value<root.value)
            root.leftchild=insert(root.leftchild,value);
        else
            root.rightchild=insert(root.rightchild,value);
        setHeight(root);

        return balance(root);
    }
    private AVlNode balance(AVlNode root){
        if(isLeftHeavy(root)){//left child,right-subtree te balance harale
left rotate on root.leftchild then right rotate on root
            if(balancefactor(root.leftchild)<0){
               root.leftchild = rotateLeft(root.leftchild);//same explanation
but opposite of rightheavy
            }
            return rotateRight(root);
        }
        if(isRightHeavy(root)){//right child ,left sub-tree te balance harale
hole right rotation on root.rightchild, then left rotate on root
           if(balancefactor(root.rightchild)>0){
               root.rightchild=rotateRight(root.rightchild);
               /*Here is the very extreme tricky part of this whole class
               10(root)
                 30(root.rightchild)
               20
               if we perform rotateRight(30) then it will look like below:-
               10
                 20(newRoot)
                   30(previously root.rightchild but now newRoot will be
root.rightchild er leftchild)
                   that means 20 should be set to rightCHild of root 10 , for
this logic will be root.rightChild = rotateRight(root.rightchild);
               * */
           }
            return rotateLeft(root);
```

```
        }
        return root;
    }
    private AVlNode rotateLeft(AVlNode root){
        var newRoot = root.rightchild;//setting newroot
        root.rightchild = newRoot.leftchild;
        newRoot.leftchild = root;
        //now we have to te reset these two roots height
        /*
        *  root.height =
Math.max(height(root.leftchild),height(root.rightchild))+1;
        * This logic will be used in multiple places. So we have to write e
helper method of calculating the new roots height
        * */
        setHeight(root);
        setHeight(newRoot);
        return newRoot;// returning the new root
    }
    private AVlNode rotateRight(AVlNode root){
        var newRoot = root.leftchild;//setting newroot
        root.leftchild = newRoot.rightchild;
        newRoot.rightchild = root;
        //now we have to te reset these two roots height
        /*
         *  root.height =
Math.max(height(root.leftchild),height(root.rightchild))+1;
         * This logic will be used in multiple places. So we have to write e
helper method of calculating the new roots height
         * */
        setHeight(root);
        setHeight(newRoot);
        return newRoot;// returning the new root
    }
    private void setHeight(AVlNode node){
        node.height =
Math.max(height(node.leftchild),height(node.rightchild))+1;
    }
    private boolean isLeftHeavy(AVlNode node){
        return balancefactor(node)>1;
    }
    private boolean isRightHeavy(AVlNode node){
        return balancefactor(node)<-1;
    }
    private int balancefactor(AVlNode node){
        // for empty tree it is balanced already
        // otherwise calculate the differences of left and right subtree and
return
        return (node==null)? 0:height(node.leftchild)-height(node.rightchild);
    }
    private int height(AVlNode node){
    return (node==null) ? -1: node.height;
    }
}
```

V-(16)-(AVLTree Exercise):-

Ex-1- Check to see if a binary tree is balanced.

Ans:-

```
public boolean isBalanced() {
    return isBalanced(root);
}

private boolean isBalanced(AVlNode root) {
    if (root == null)
        return true;

    var balanceFactor = height(root.leftchild) - height(root.rightchild);

    return Math.abs(balanceFactor) <= 1 &&
            isBalanced(root.leftchild) &&
            isBalanced(root.rightchild);
}
```

Ex-2 . Is the tree perfect or not?

[hint :- If all parents had exactly two child in a tree then we called it a perfect binary Tree]

[Resource(for more):- https://www.programiz.com/dsa/perfect-binary-tree]

Ans:-

```
   public boolean isPerfect() {
    return size() == (Math.pow(2, height(root) + 1) - 1);
}
public int size() {
    return size(root);
}

private int size(AVlNode root) {
    if (root == null)
        return 0;

    if (isLeaf(root))
        return 1;

    return 1 + size(root.leftchild) + size(root.rightchild);
}
private boolean isLeaf(AVlNode node) {
    return node.leftchild == null && node.rightchild == null;
}
private int height(AVlNode node){
return (node==null) ? -1: node.height;
}
```

V-(17)-(Summary):-

AVL TREES

- Balanced and unbalanced
- BST
  - Average: O(log n)
  - Worst: O(n)
- Self-balancing trees
- Rotations: Left, Right, Left-Right and Right-Left