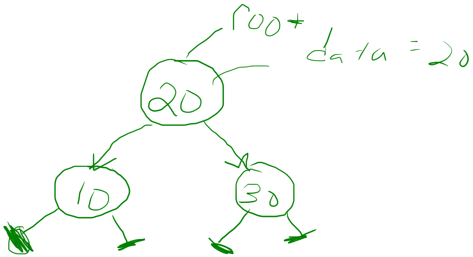
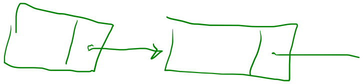
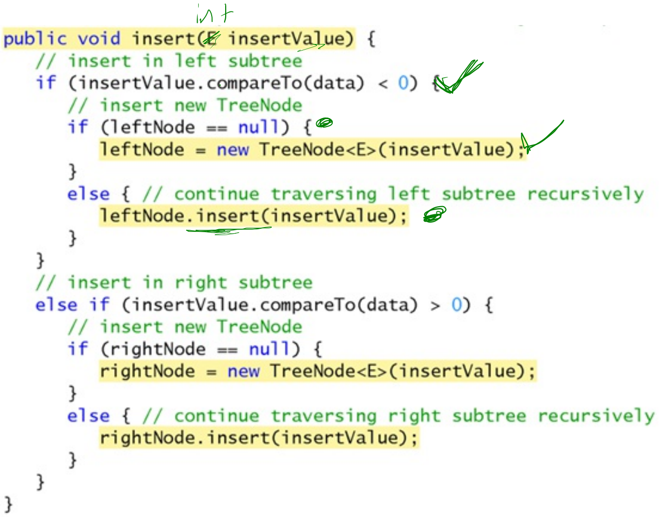
```
class TreeNode {
    int data;
    TreeNode left;
    TreeNode right;

    ...getLeft()....
}
```



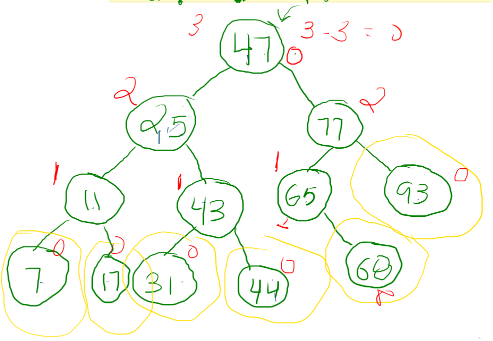




int

```
public void insert(E insertValue) {
   // insert in left subtree
   if (insertValue.compareTo(data) < 0) {
      // insert new TreeNode
      if (leftNode == null) {
         leftNode = new TreeNode<E>(insertValue);
      }
      else { // continue traversing left subtree recursively
         leftNode.insert(insertValue);
      }
   }
   // insert in right subtree
   else if (insertValue.compareTo(data) > 0) {
      // insert new TreeNode
      if (rightNode == null) {
         rightNode = new TreeNode<E>(insertValue);
      }
      else { // continue traversing right subtree recursively
         rightNode.insert(insertValue);
      }
   }
}
```

47,25,11,43,77,65,93,7,31,17,44,68



$n = 12$   3

$O(\log_2 n)$

root node - node at top

leaf node - node with no children

height - max # of edges to a leaf node

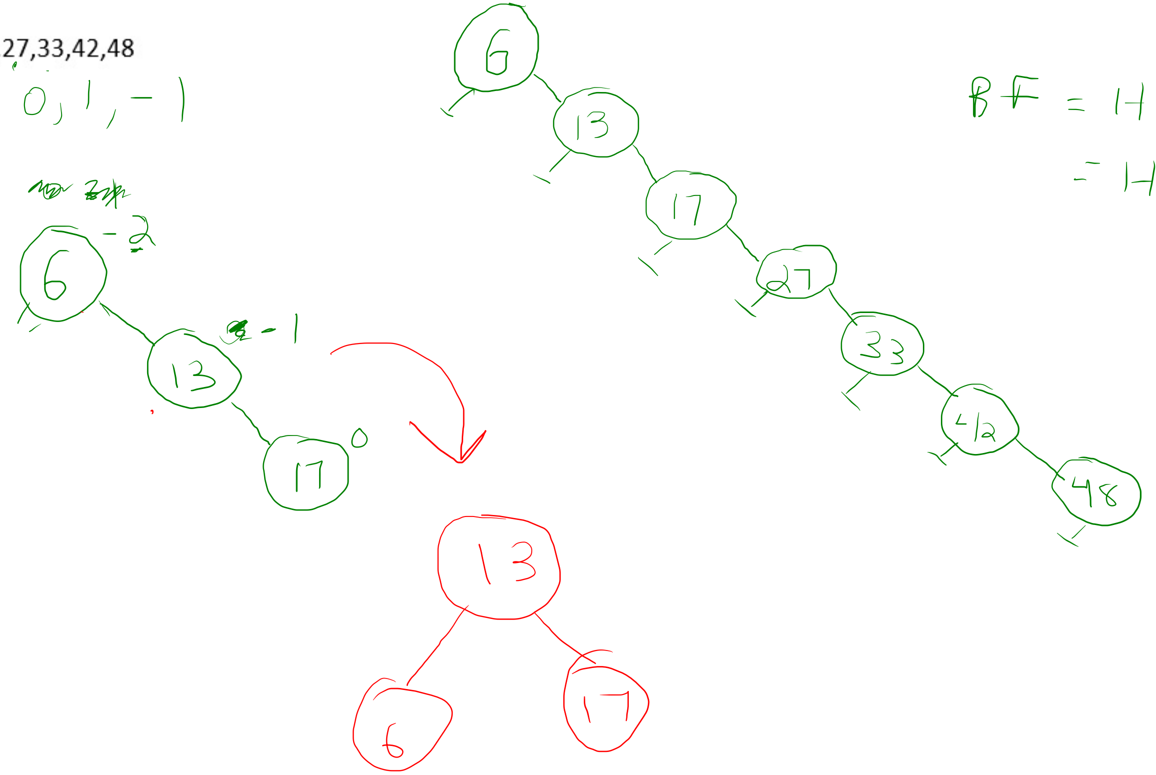
BF = HL - HR

Balanced Node = BF of 1, 0 , -1

Balanced Tree = all nodes are balanced, ie, BF of 1,0,-1

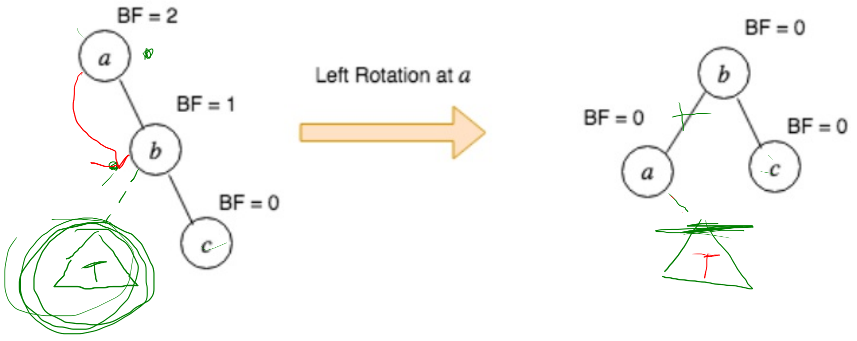
6,13,17,27,33,42,48

0, 1, -1



BF = HL - HR

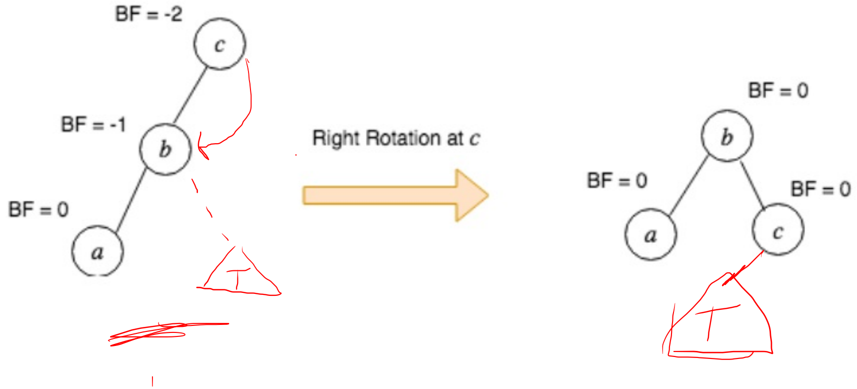
= HR - HL



```
public static Node leftRotate(Node aNode) {
      Node bNode= aNode.right;
      Node T = bNode.left

       bNode.left = aNode;
       aNode.right = T;

       return bNode;
}
```



```
public Node rightRotate(Node cNode) {
      Node bNode = cNode.left;
      Node T = bNode.right;


      bNode.right = cNode;
      cNode.left = T;

      return bNode;
}
```

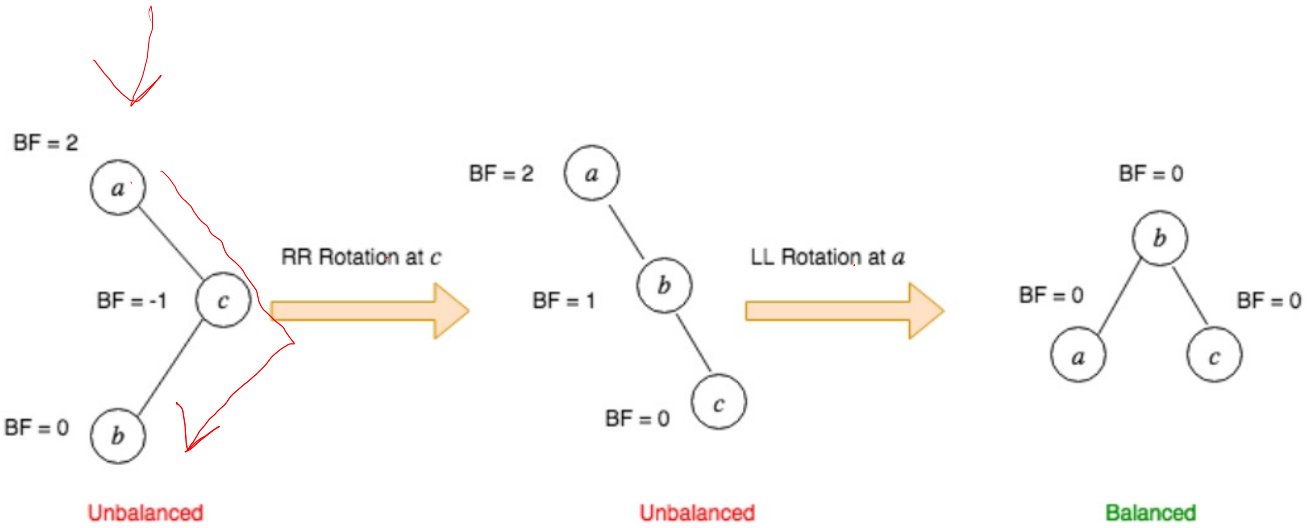


Fig 7: Illustrating the left-right rotation

We perform the left right rotation (LR) on node $x$ when

- Node $x$ is right heavy
- Node $x$'s right subtree is left heavy

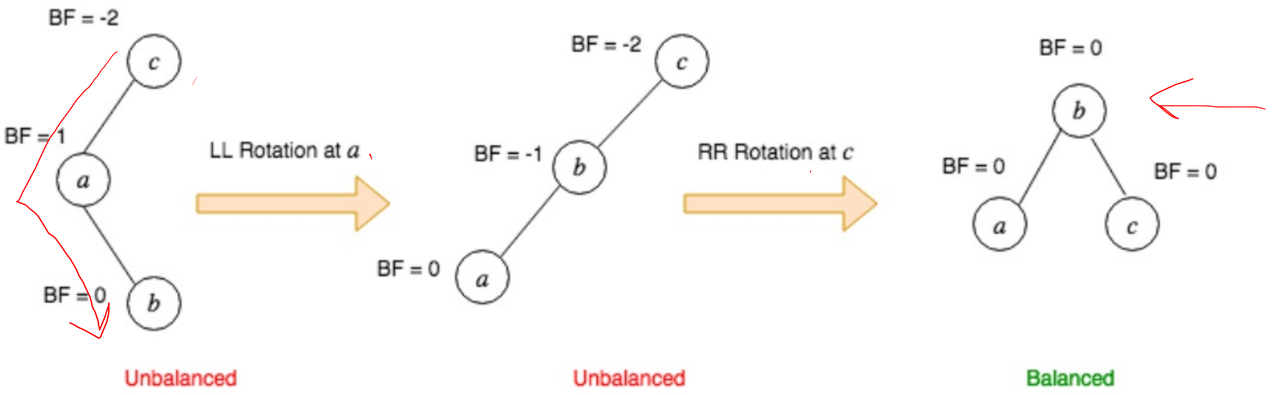
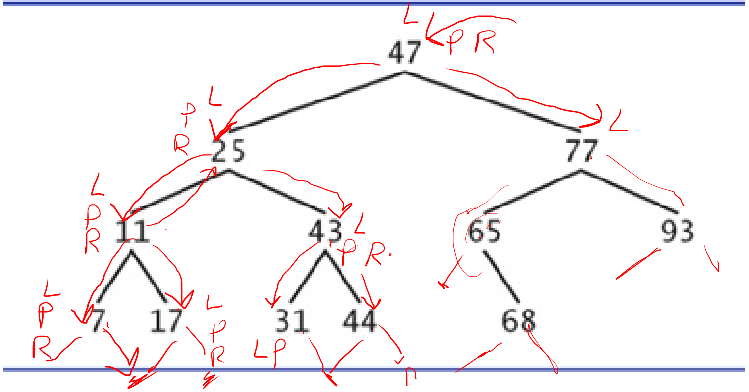


Fig 8: Illustrating the right-left rotation

We perform the right left rotation (LR) on node $x$ when

- Node $x$ is left heavy
- Node $x$'s left subtree is right heavy



{7, 11, 17, 25, 31, 43, 44, 47, 65, 68
77, 93

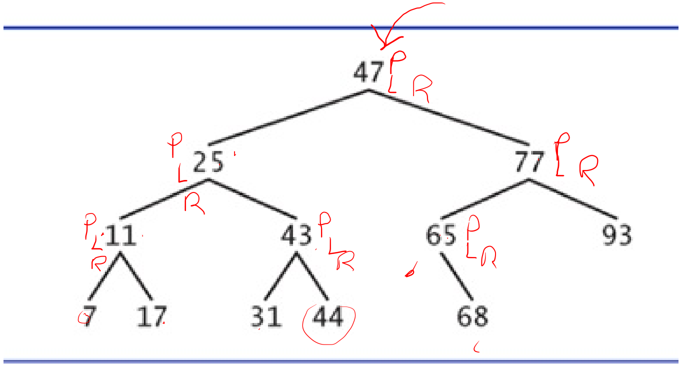
In Order  Traversal

1.Left

2. Process

3. Right

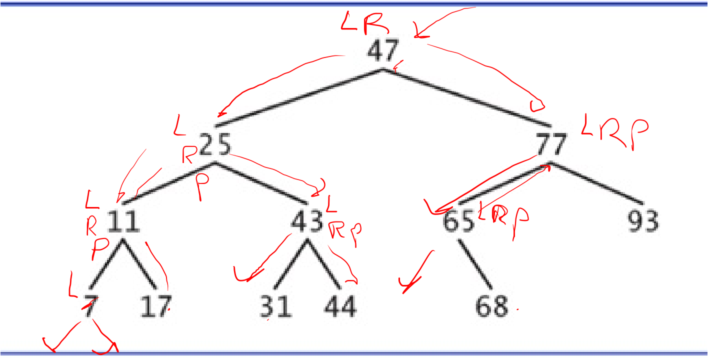
Proces => print out the value of node



47, 25, 11, 7, 17, 43, 31, 44, 77, 65, 68, 93

Pre-Order Traversal

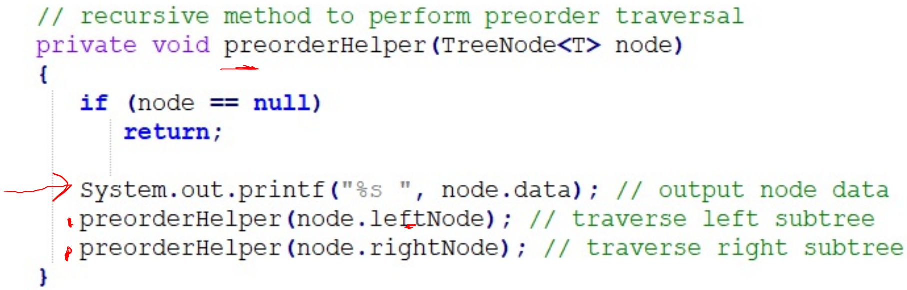
1. Process
2. Left
3. Right



7,17,11,31,44,43,25,68,65,93,77,47

Post - Order Traversal

1. Left
2. Right
3. Proces

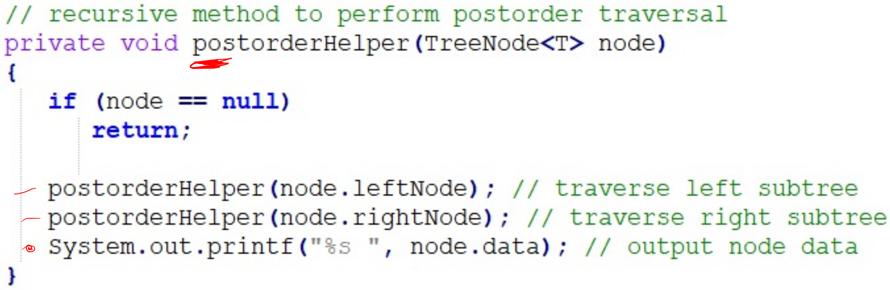


```java
// recursive method to perform preorder traversal
private void preorderHelper(TreeNode<T> node)
{
    if (node == null)
        return;

    System.out.printf("%s ", node.data); // output node data
    preorderHelper(node.leftNode); // traverse left subtree
    preorderHelper(node.rightNode); // traverse right subtree
}
```

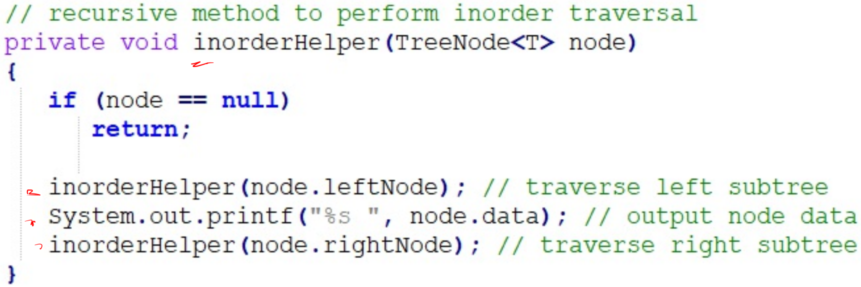
TreeNode<Integer> tree;

tree.preOrderHelper(tree.root);



```java
// recursive method to perform postorder traversal
private void postorderHelper(TreeNode<T> node)
{
    if (node == null)
        return;

    postorderHelper(node.leftNode); // traverse left subtree
    postorderHelper(node.rightNode); // traverse right subtree
    System.out.printf("%s ", node.data); // output node data
}
```



```java
// recursive method to perform inorder traversal
private void inorderHelper(TreeNode<T> node)
{
    if (node == null)
        return;

    inorderHelper(node.leftNode); // traverse left subtree
    System.out.printf("%s ", node.data); // output node data
    inorderHelper(node.rightNode); // traverse right subtree
}
```