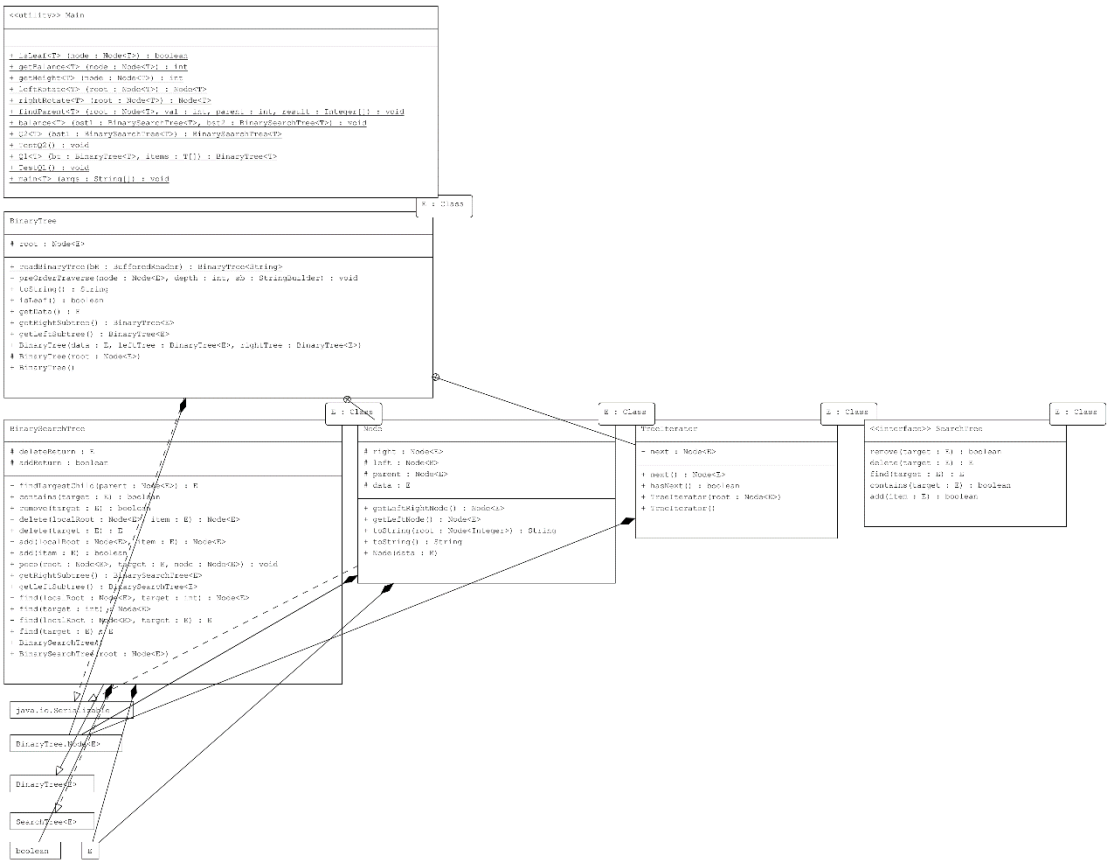
Detailed System Requirements:

- jdk and jre are requested from operating system to execute this java program.

-User must run makefile folder

Class Diagram:



Q1)

Solution Approach:

In my method, first I sort the given array After that I traverse the Given **BinaryTree** in **inorder** way and put sorted arrays elements one by one to the binarytree. When function completed the given **BinaryTree** will be **BinarySearchTree** and return it.

1- Sort given items array: I used Quick Sort

```
//1-    Sort given items array
Arrays.sort(items); //quick sort
```

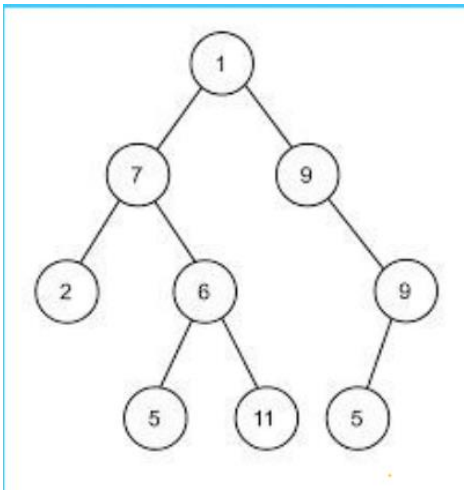
2- Traverse Given Binary Tree in **inorder** way
3- Put Sorted Arrays elements one by one to the binary Tree

```
//2-    Traverse Given Binary Tree in inorder way
BinaryTree.TreeIterator it = bt.new TreeIterator();
int i=0;
while(it.hasNext()) {
    it.next().data = items[i];
    i++;
}
```

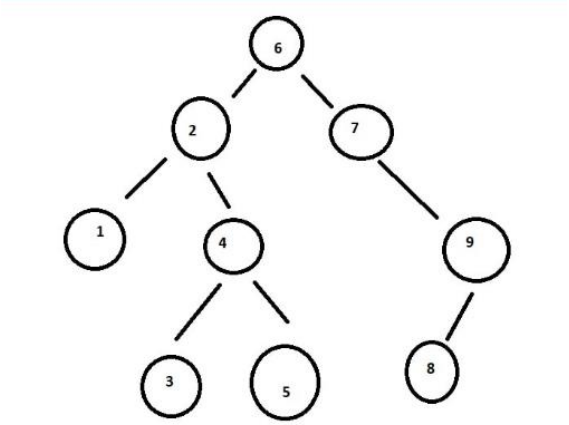
Note: **TreeITerator** is an iterator that helps to traverse the tree **inorder** way.

```
public static <T extends Comparable<T>> BinaryTree<T> Q1(BinaryTree<T> bt, T[] items) {
    //BinarySearchTree<T> bst = new BinarySearchTree<T>();

    System.out.print("Given arr: [");
    for(int i=0; i<items.length; ++i) System.out.print(items[i]+" ");
    System.out.println("] ");

    BinaryTree.TreeIterator it2 = bt.new TreeIterator();
    System.out.print("Given bt (inorder way): ");
    while(it2.hasNext()) System.out.print(it2.next().data+" ");
    System.out.println();

    //1-    Sort given items array
    Arrays.sort(items); //quick sort

    //2-    Traverse Given Binary Tree in inorder way
    BinaryTree.TreeIterator it = bt.new TreeIterator();
    int i=0;
    while(it.hasNext()) {
        it.next().data = items[i];
        i++;
    }

    //print
    System.out.println();
    System.out.print("After Operation (inorder way): ");
    it2 = bt.new TreeIterator();
    while(it2.hasNext()) System.out.print(it2.next().data+" ");

    return bt;
}
```

First we give this tree to **Q1** function



And we change above tree to this tree:




```
Before:

1
  7
    2
      null
      null
    6
      5
        null
        null
      11
        null
        null
  9
    null
    9
      5
        null
        null
      null
```

```
------------------------
After:

Given arr: [9 8 7 6 5 4 3 2 1 ]
Given bt (inorder way): 2 7 5 6 11 1 9 5 9

After Operation (inorder way): 1 2 3 4 5 6 7 8 9
6
  2
    1
      null
      null
    4
      3
        null
        null
      5
        null
        null
  7
    null
    9
      8
        null
        null
      null
```

Works Perfectly.

Time Complexity: O(n^2)

Solution Approach:

The main function is **balance** function in here. Balance function is a recursive function. We look left and rightsub trees first. After we arrive the leaf nodes we look the balance value of current node and determine which rotation process is required. When determined which rotation type must be done, that rotation will be carried out.

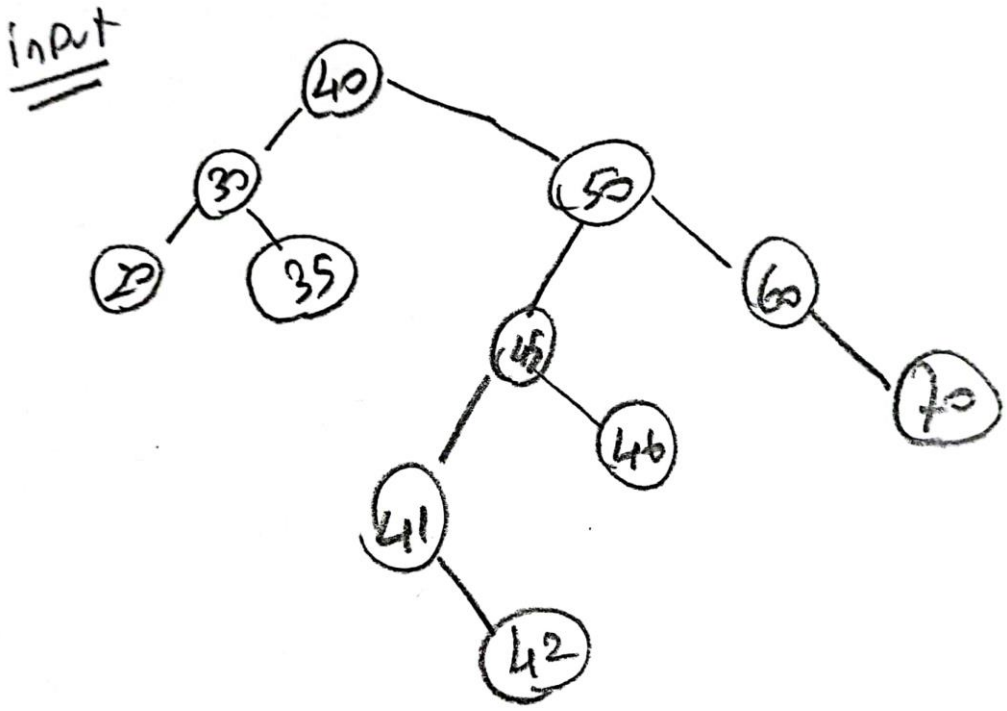
```java
/**
 * Balances the given binary search tree - O(nlogn)
 * @param bst1 binary search tree that will changed
 * @param bst2 different reference for bst1
 * */
public static <T extends Comparable<T>> void balance(BinarySearchTree<T> bst1, BinarySearchTree<T> bst2) {

    if (bst1 == null)
        return;

    var rootVal = bst1.root.data;
    balance(bst1.getLeftSubtree(), bst2);
    balance(bst1.getRightSubtree(), bst2);

    int balance = getBalance(bst1.root);

    Integer[] arr = new Integer[1];
    // LL
    if (balance > 1 && getBalance(bst1.root.left) >= 0) {
        var temp = bst1.root.data;
        bst1.root = rightRotate(bst1.root);
        if (temp != rootVal) {
            findParent(bst2.root, (int) temp, -1, arr);
            if(bst2.find(arr[0]).data.compareTo(bst1.root.data) < 0)
                bst2.find(arr[0]).right = bst1.root;
            else
                bst2.find(arr[0]).left = bst1.root;
        }
    }
```

```java
    // RR
    if (balance < -1 && getBalance(bst1.root.right) <= 0) {
        var temp = bst1.root.data;
        bst1.root = leftRotate(bst1.root);
        if(temp != rootVal){
            findParent(bst2.root, (int) temp, -1, arr);
            if (bst2.find(arr[0]).data.compareTo(bst1.root.data) < 0)
                bst2.find(arr[0]).right = bst1.root;
            else
                bst2.find(arr[0]).left = bst1.root;
        }
    }

     // LR
    if (balance > 1 && getBalance(bst1.root.left) == -1) {
        var temp = bst1.root.data;
        bst1.root.left = leftRotate(bst1.root.left);
        bst1.root = rightRotate(bst1.root);
        if(temp != rootVal){
            findParent(bst2.root, (int) temp, -1, arr);
            if (bst2.find(arr[0]).data.compareTo(bst1.root.data) < 0)
                bst2.find(arr[0]).right = bst1.root;
            else
                bst2.find(arr[0]).left = bst1.root;
        }
    }
```

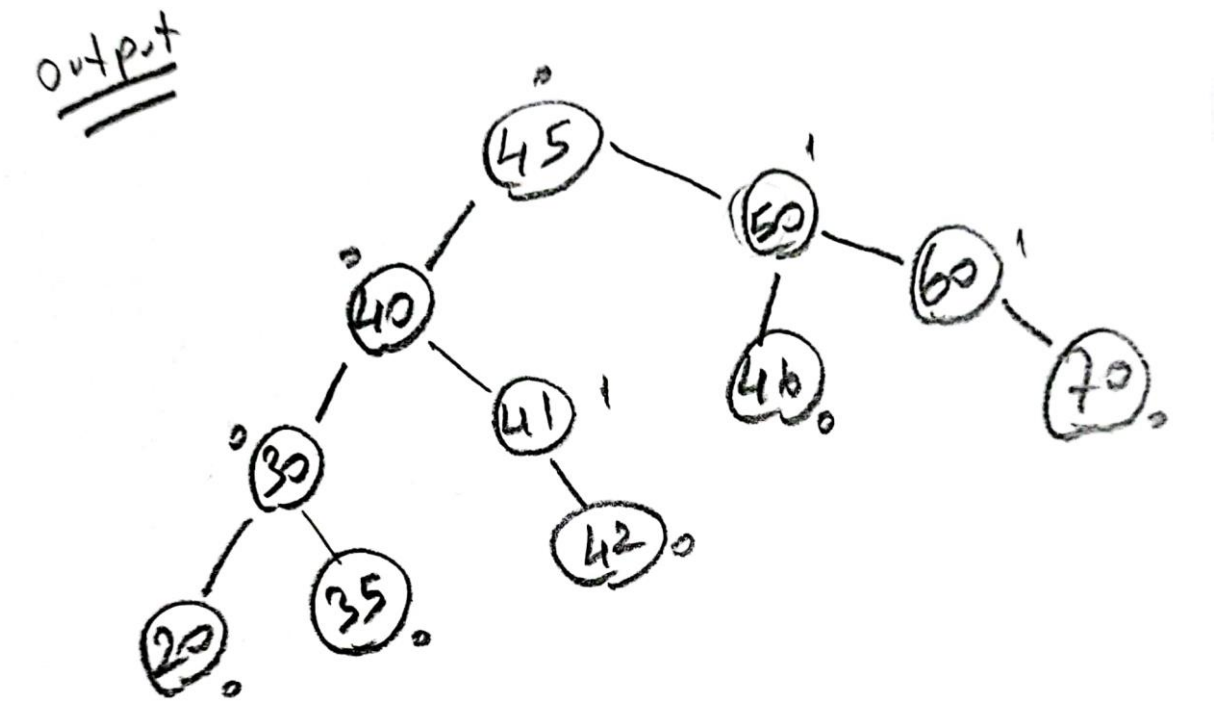
```
    // RL
    if (balance < -1 && getBalance(bst1.root.right) == 1) {
        var temp = bst1.root.data;
        bst1.root.right = rightRotate(bst1.root.right);
        bst1.root = leftRotate(bst1.root);
        if(temp != rootVal){
            findParent(bst2.root, (int)temp, -1, arr);
            if (bst2.find(arr[0]).data.compareTo(bst1.root.data) < 0)
                bst2.find(arr[0]).right = bst1.root;
            else  bst2.find(arr[0]).left = bst1.root;
        }

    }
```

Time Complexity:

O(nlog(n))



```
Before:
40
  30
    20
      null
      null
    35
      null
      null
  50
    45
      41
        null
        42
          null
          null
      46
        null
        null
    60
      null
      70
        null
        null


--------------------------------
```



```
After:
45
  40
    30
      20
        null
        null
      35
        null
        null
    41
      null
      42
        null
        null
  50
    46
      null
      null
    60
      null
      70
        null
        null
```

Works perfectly.