GTU Department of Computer Engineering

CSE 222/505 - Spring 2022

Homework #7 Report

Serhat SARI
200104004028

1- SYSTEM REQUIREMENTS

a- Non-Functional System Requirements

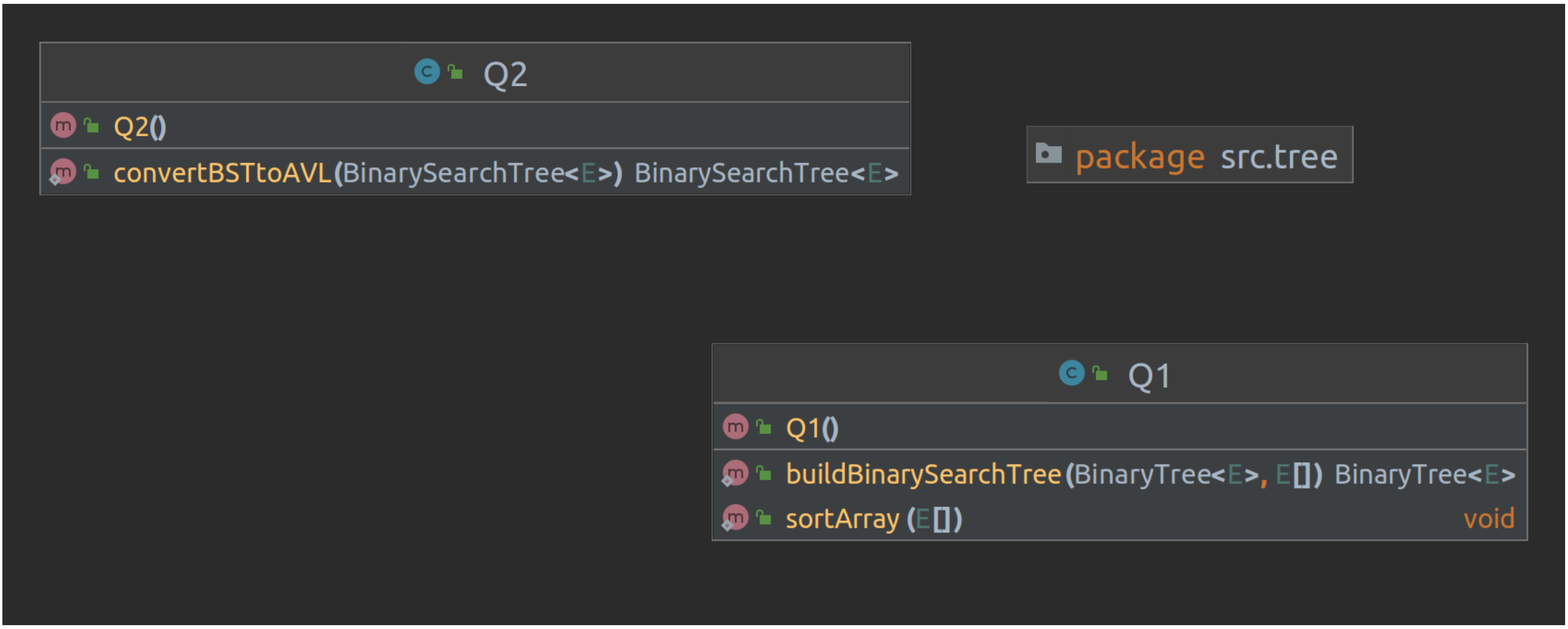
1- Back-end Software : Java 11

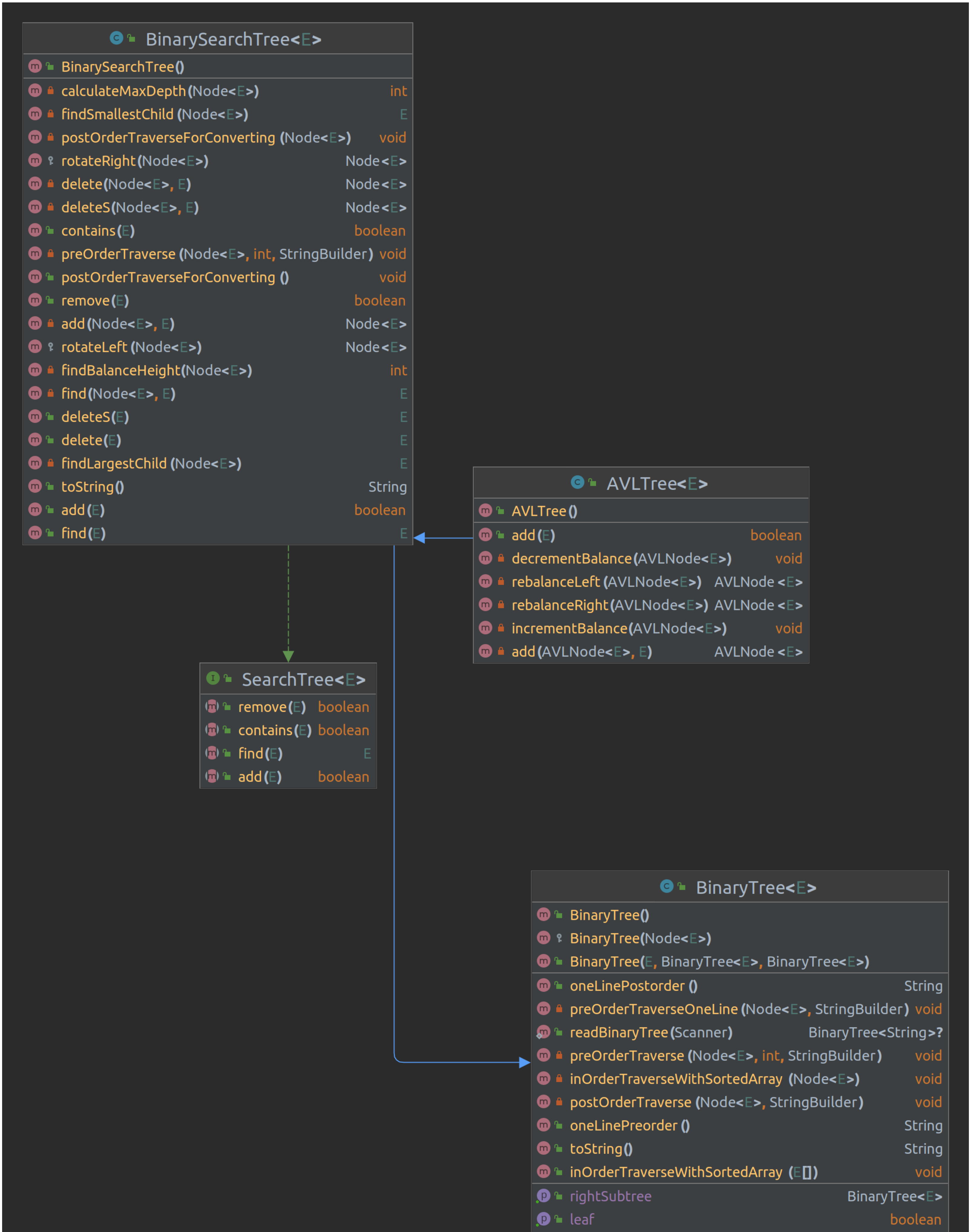
2- Software should be able to compile with “javac” on a linux distribution.

b-Functional System Requirements

In the first question, you should give binary tree and array to method.
In the second question, you should give binary search tree to the method.

2- CLASS DIAGRAM


Q2
Q2()
convertBSTtoAVL(BinarySearchTree<E>) BinarySearchTree<E>
package src.tree
Q1
Q1()
buildBinarySearchTree(BinaryTree<E>, E[]) BinaryTree<E>
sortArray(E[]) void



BinarySearchTree<E>

BinarySearchTree()	
calculateMaxDepth(Node<E>)	int
findSmallestChild(Node<E>)	E
postOrderTraverseForConverting(Node<E>)	void
rotateRight(Node<E>)	Node<E>
delete(Node<E>, E)	Node<E>
deleteS(Node<E>, E)	Node<E>
contains(E)	boolean
preOrderTraverse(Node<E>, int, StringBuilder)	void
postOrderTraverseForConverting()	void
remove(E)	boolean
add(Node<E>, E)	Node<E>
rotateLeft(Node<E>)	Node<E>
findBalanceHeight(Node<E>)	int
find(Node<E>, E)	E
deleteS(E)	E
delete(E)	E
findLargestChild(Node<E>)	E
toString()	String
add(E)	boolean
find(E)	E

AVLTree<E>

AVLTree()	
add(E)	boolean
decrementBalance(AVLNode<E>)	void
rebalanceLeft(AVLNode<E>)	AVLNode<E>
rebalanceRight(AVLNode<E>)	AVLNode<E>
incrementBalance(AVLNode<E>)	void
add(AVLNode<E>, E)	AVLNode<E>

SearchTree<E>

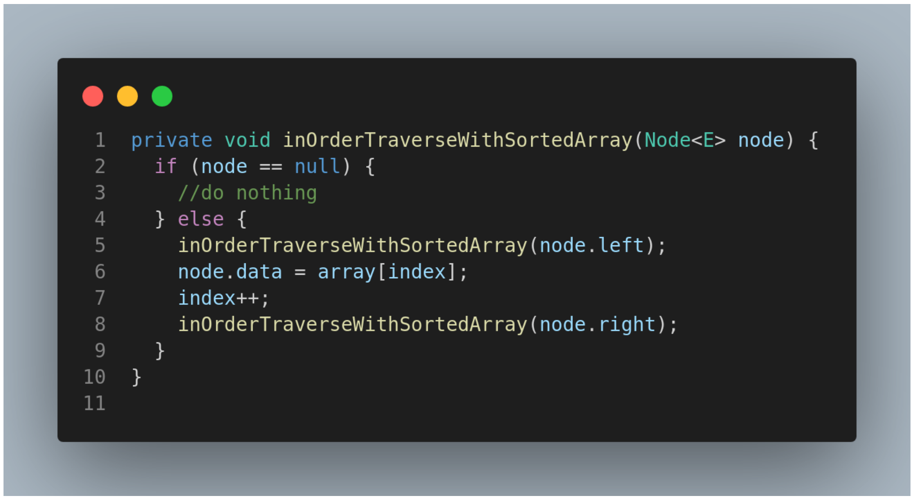
remove(E)	boolean
contains(E)	boolean
find(E)	E
add(E)	boolean

BinaryTree<E>

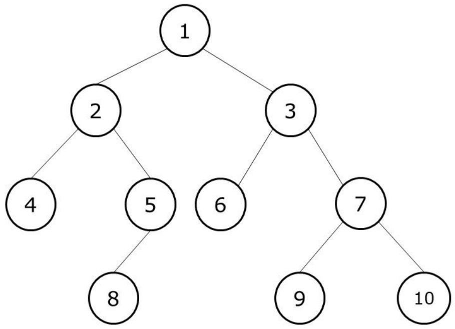
BinaryTree()	
BinaryTree(Node<E>)	
BinaryTree(E, BinaryTree<E>, BinaryTree<E>)	
oneLinePostorder()	String
preOrderTraverseOneLine(Node<E>, StringBuilder)	void
readBinaryTree(Scanner)	BinaryTree<String>?
preOrderTraverse(Node<E>, int, StringBuilder)	void
inOrderTraverseWithSortedArray(Node<E>)	void
postOrderTraverse(Node<E>, StringBuilder)	void
oneLinePreorder()	String
toString()	String
inOrderTraverseWithSortedArray(E[])	void
rightSubtree	BinaryTree<E>
leaf	boolean

3- Problem Solution Approach

For the first question, i used inorder traverse to travel the tree.
Because it will show the tree elements from the lowest to greatest.
I sorted the given array. And i updated every node of tree by
traversing inorder. Shortly, i sorted the array first. Then i traverse the tree
with inorder so i can update the nodes. Then i update every node starting from
the lowest to the greatest.



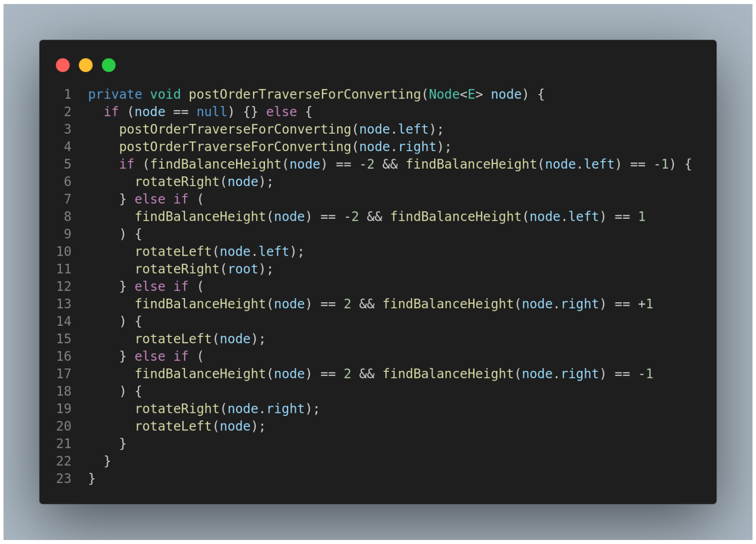
```java
private void inOrderTraverseWithSortedArray(Node<E> node) {
  if (node == null) {
    //do nothing
  } else {
    inOrderTraverseWithSortedArray(node.left);
    node.data = array[index];
    index++;
    inOrderTraverseWithSortedArray(node.right);
  }
}
```



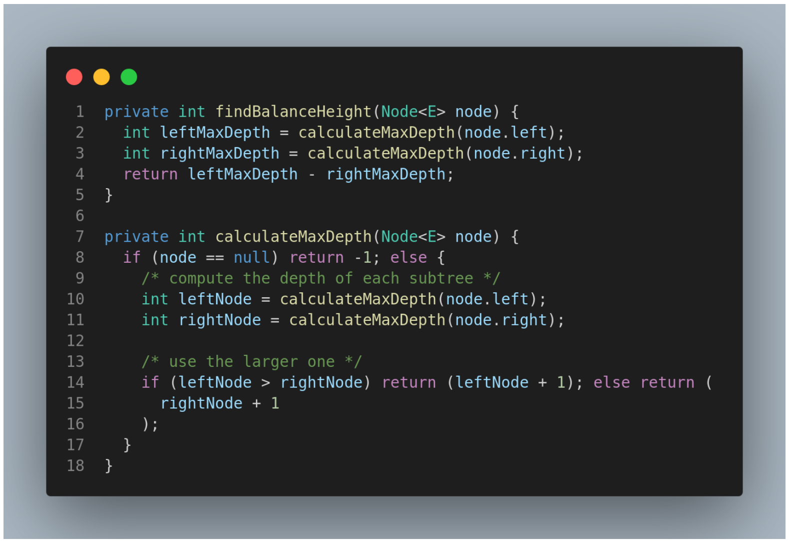
Inorder Traversal:
[left,root,right]

4	2	8	5	1	6	3	9	7	10

For the second question, i implement a method that finds the height of node.
And i traverse the tree with postorder and whenever there is unbalanced
node, i rotate the node according to the weight side.

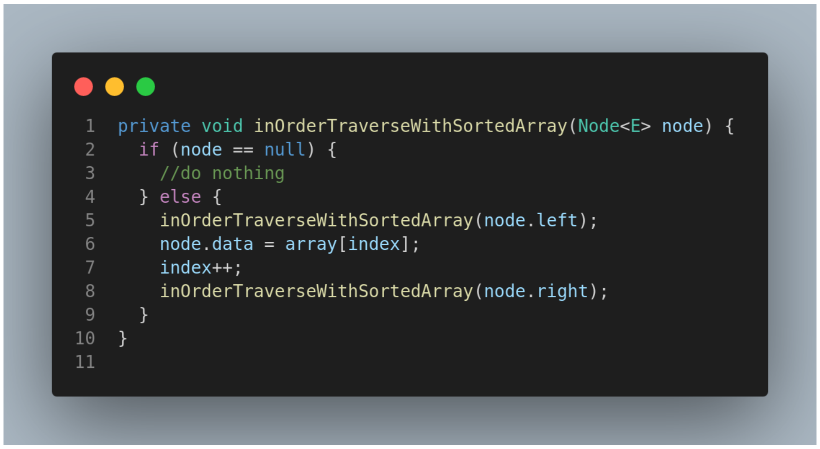


```java
private void postOrderTraverseForConverting(Node<E> node) {
  if (node == null) {} else {
    postOrderTraverseForConverting(node.left);
    postOrderTraverseForConverting(node.right);
    if (findBalanceHeight(node) == -2 && findBalanceHeight(node.left) == -1) {
      rotateRight(node);
    } else if (
      findBalanceHeight(node) == -2 && findBalanceHeight(node.left) == 1
    ) {
      rotateLeft(node.left);
      rotateRight(root);
    } else if (
      findBalanceHeight(node) == 2 && findBalanceHeight(node.right) == +1
    ) {
      rotateLeft(node);
    } else if (
      findBalanceHeight(node) == 2 && findBalanceHeight(node.right) == -1
    ) {
      rotateRight(node.right);
      rotateLeft(node);
    }
  }
}
```



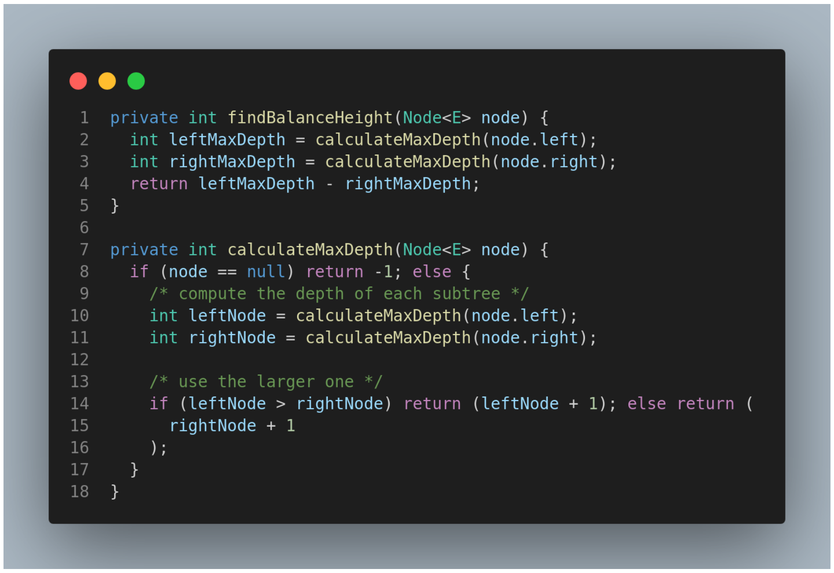
```java
private int findBalanceHeight(Node<E> node) {
  int leftMaxDepth = calculateMaxDepth(node.left);
  int rightMaxDepth = calculateMaxDepth(node.right);
  return leftMaxDepth - rightMaxDepth;
}

private int calculateMaxDepth(Node<E> node) {
  if (node == null) return -1; else {
    /* compute the depth of each subtree */
    int leftNode = calculateMaxDepth(node.left);
    int rightNode = calculateMaxDepth(node.right);

    /* use the larger one */
    if (leftNode > rightNode) return (leftNode + 1); else return (
      rightNode + 1
    );
  }
}
```

COMPLEXITY ANALYSIS



```
private void inOrderTraverseWithSortedArray(Node<E> node) {
  if (node == null) {
    //do nothing
  } else {
    inOrderTraverseWithSortedArray(node.left);
    node.data = array[index];
    index++;
    inOrderTraverseWithSortedArray(node.right);
  }
}
```

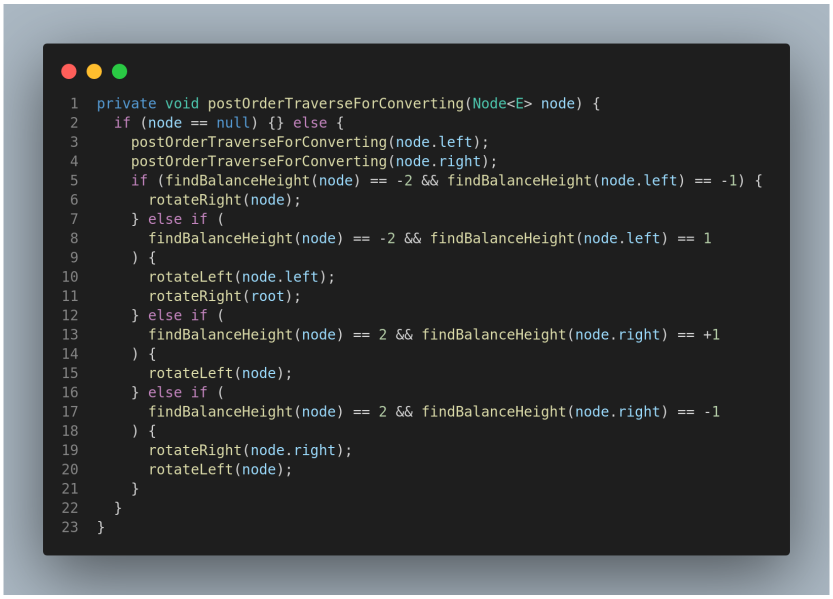
Conplexity = O(log(n))



```
private int findBalanceHeight(Node<E> node) {
  int leftMaxDepth = calculateMaxDepth(node.left);
  int rightMaxDepth = calculateMaxDepth(node.right);
  return leftMaxDepth - rightMaxDepth;
}

private int calculateMaxDepth(Node<E> node) {
  if (node == null) return -1; else {
    /* compute the depth of each subtree */
    int leftNode = calculateMaxDepth(node.left);
    int rightNode = calculateMaxDepth(node.right);

    /* use the larger one */
    if (leftNode > rightNode) return (leftNode + 1); else return (
      rightNode + 1
    );
  }
}
```

findBalanceHeight Complexity:
O(log(n))

calculateMaxDepth Complexity:
O(log(n))



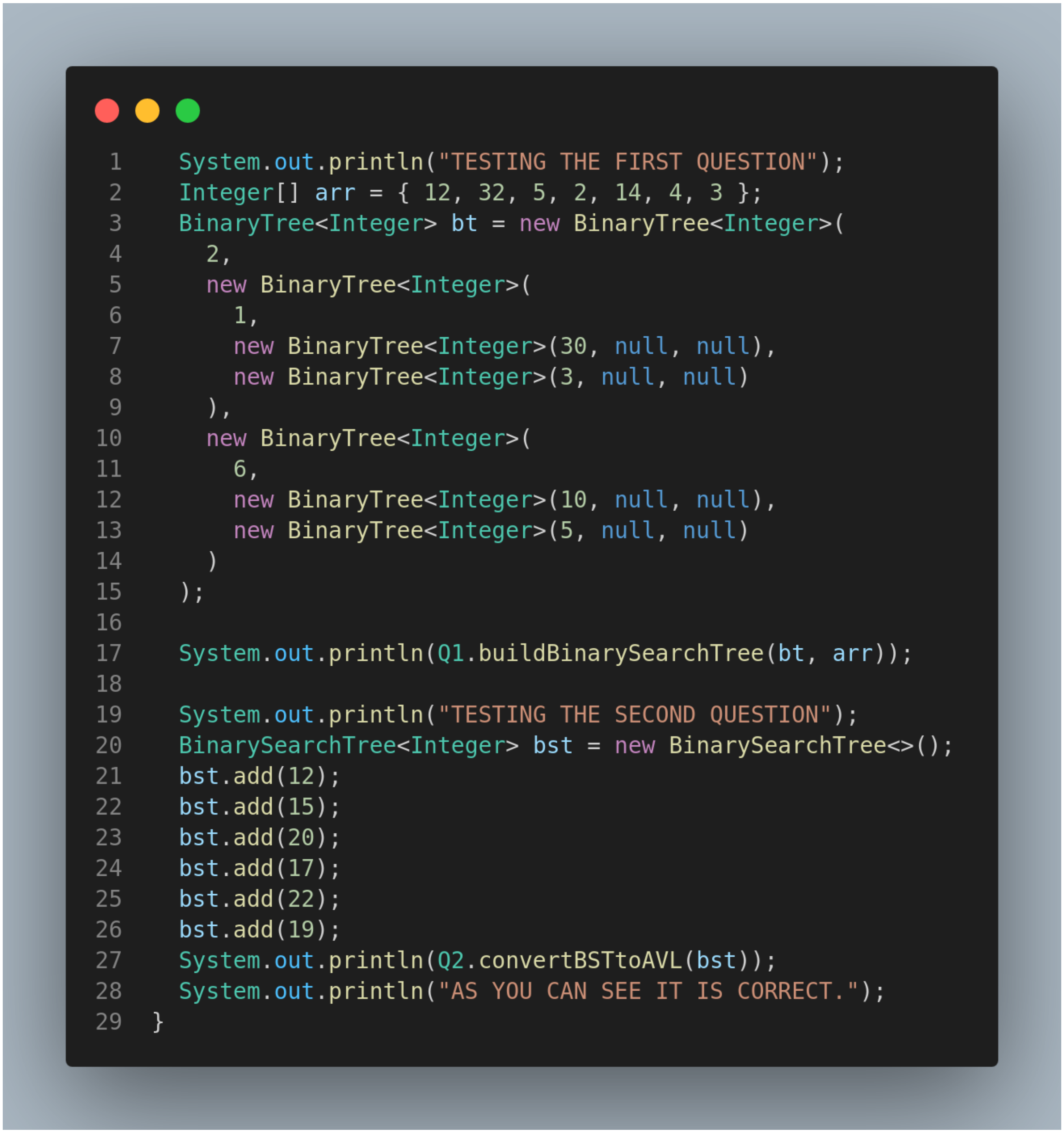
```
private void postOrderTraverseForConverting(Node<E> node) {
  if (node == null) {} else {
    postOrderTraverseForConverting(node.left);
    postOrderTraverseForConverting(node.right);
    if (findBalanceHeight(node) == -2 && findBalanceHeight(node.left) == -1) {
      rotateRight(node);
    } else if (
      findBalanceHeight(node) == -2 && findBalanceHeight(node.left) == 1
    ) {
      rotateLeft(node.left);
      rotateRight(root);
    } else if (
      findBalanceHeight(node) == 2 && findBalanceHeight(node.right) == +1
    ) {
      rotateLeft(node);
    } else if (
      findBalanceHeight(node) == 2 && findBalanceHeight(node.right) == -1
    ) {
      rotateRight(node.right);
      rotateLeft(node);
    }
  }
}
```

postOrderTraverseForCOnverting Complexity:

O(log(n))

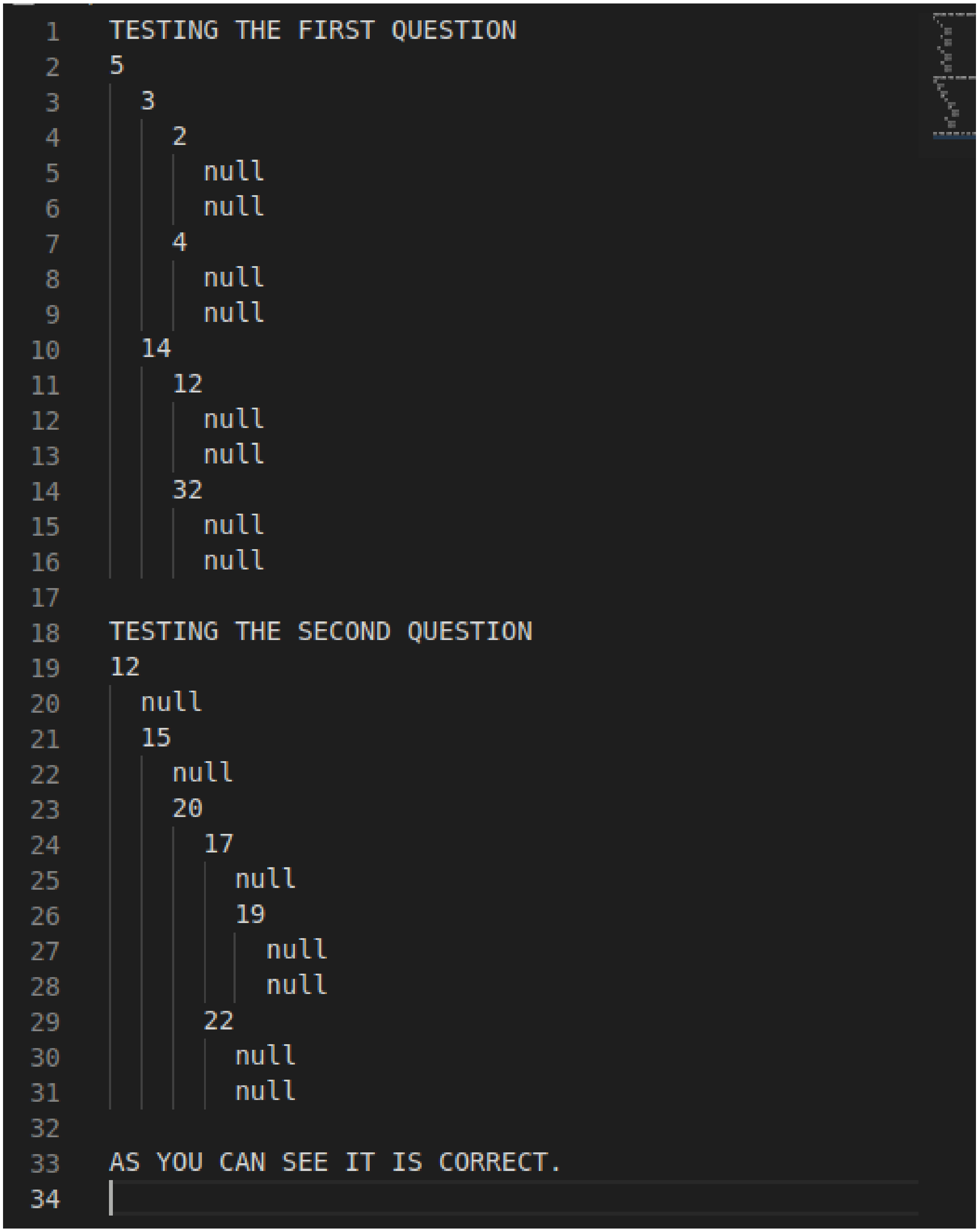
rotateLeft and rotateRight
Complexity: theta(1)

4- Test Cases



```java
  System.out.println("TESTING THE FIRST QUESTION");
  Integer[] arr = { 12, 32, 5, 2, 14, 4, 3 };
  BinaryTree<Integer> bt = new BinaryTree<Integer>(
    2,
    new BinaryTree<Integer>(
      1,
      new BinaryTree<Integer>(30, null, null),
      new BinaryTree<Integer>(3, null, null)
    ),
    new BinaryTree<Integer>(
      6,
      new BinaryTree<Integer>(10, null, null),
      new BinaryTree<Integer>(5, null, null)
    )
  );

  System.out.println(Q1.buildBinarySearchTree(bt, arr));

  System.out.println("TESTING THE SECOND QUESTION");
  BinarySearchTree<Integer> bst = new BinarySearchTree<>();
  bst.add(12);
  bst.add(15);
  bst.add(20);
  bst.add(17);
  bst.add(22);
  bst.add(19);
  System.out.println(Q2.convertBSTtoAVL(bst));
  System.out.println("AS YOU CAN SEE IT IS CORRECT.");
}
```

5- Running Commands And Result



```
TESTING THE FIRST QUESTION
5
  3
    2
      null
      null
    4
      null
      null
  14
    12
      null
      null
    32
      null
      null

TESTING THE SECOND QUESTION
12
  null
  15
    null
    20
      17
        null
        19
          null
          null
      22
        null
        null

AS YOU CAN SEE IT IS CORRECT.
```