GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework #7 Report

Mert Gürşimşir
1901042646

PART 1

THEORETICAL RUN TIME COMPLEXITY ANALYSIS OF QUESTION 1

Java sort uses quick sort so it is → θ(nlogn) on average

Recursive method:

$$T(n) = 2T(n/2) + \theta(1)$$
$$T(n) = 2(2T(n/4) + \theta(1)) + \theta(1) = 2^2T(n/4) + 3\theta(1)$$
$$T(n) = 2^2[2T(n/8) + \theta(1)] + 3\theta(1) = 2^3T(n/8) + 7\theta(1)$$
$$T(n) = 2^3[2T(n/16) + \theta(1)] + 7\theta(1) = 2^4T(n/2^4) + 2^3\theta(1) + 2^2\theta(1) + 2^1\theta(1) + 2^0\theta(1)$$
$$T(n) = 2^5T(n/2^5) + 2^4\theta(1) + 2^3\theta(1) + 2^2\theta(1) + 2^1\theta(1) + 2^0\theta(1)$$
$$\ldots$$
$$T(n) = 2^kT(n/2^k) + 2^{k-1}\theta(1) + 2^{k-2}\theta(1) + \ldots + 2^0\theta(1)$$
$$2^k = n \quad , \quad k = log_2 n :$$
$$T(n) = nT(1) + (2^k - 1)\theta(1) = nT(1) + (n - 1)\theta(1) \qquad (T(1) = \theta(1))$$
$$T(n) = n + n - 1 = 2n - 1$$
$$T(n) = \theta(n) \rightarrow \text{on average}$$

Or shortly we can say if there are n nodes, there are 2n+1 calls for the traversal algorithms:

$$T(n) = \theta(n)$$

Adding elements of binary tree to binary search tree (createBST method):

$$T(n) = 2T(n/2) + n$$
$$T(n) = 2^2[T(n/2^2)] + n + n$$
$$T(n) = 2^3[T(n/2^3)] + 3n$$
$$\ldots$$
$$T(n) = 2^k(T(n/2^k)) + kn$$
$$2^k = n \quad , \quad k = log_2 n :$$
$$T(n) = 2^kT(1) + kn$$
$$T(n) = n + nlogn = O(nlogn)$$

IT IS NOT θ BECAUSE WORST CASE FOR ADDING TO BST IS LINEAR

All in all, this program runs in T(n) = θ(nlogn) time.

THEORETICAL RUN TIME COMPLEXITY ANALYSIS OF QUESTION 2

Rotate to right and left (rotateRight() and rotateLeft()) runs in constant time. → $\theta(1)$

convertAVL(node):

$T(n) = 2T(n/2) + n$
$T(n) = 2^2[T(n/2^2)] + n + n$
$T(n) = 2^3[T(n/2^3)] + 3n$
…
$T(n) = 2^k(T(n/2^k)) + kn$
$2^k = n$, $k = log_2 n$:
$T(n) = 2^kT(1) + kn$
$T(n) = n + nlogn = \theta(nlogn)$

height():

$T(n) = 2T(n/2) + \theta(1)$
$T(n) = 2(2T(n/4) + \theta(1)) + \theta(1) = 2^2T(n/4) + 3\theta(1)$
$T(n) = 2^2[2T(n/8) + \theta(1)] + 3\theta(1) = 2^3T(n/8) + 7\theta(1)$
$T(n) = 2^3[2T(n/16) + \theta(1)] + 7\theta(1) = 2^4T(n/2^4) + 2^3\theta(1) + 2^2\theta(1) + 2^1\theta(1) + 2^0\theta(1)$
$T(n) = 2^5T(n/2^5) + 2^4\theta(1) + 2^3\theta(1) + 2^2\theta(1) + 2^1\theta(1) + 2^0\theta(1)$
…
$T(n) = 2^kT(n/2^k) + 2^{k-1}\theta(1) + 2^{k-2}\theta(1) + \ldots + 2^0\theta(1)$
$2^k = n$, $k = log_2 n$:
$T(n) = nT(1) + (2^k - 1)\theta(1) = nT(1) + (n - 1)\theta(1)$ $(T(1) = \theta(1))$
$T(n) = n + n - 1 = 2n - 1$
$T(n) = \theta(n)$ → on average

balance():

$T(n) = \theta(n)$ → on average

All in all, this program runs in $T(n) = \theta(nlogn)$ time.

PART 2

1. SYSTEM REQUIREMENTS

<u>Environment</u>

Editor: Sublime Text

OS: Ubuntu

Javac Version: 11.0.15

<u>Q1</u>

To create binary search tree from the elements in the given array and from the structure of given binary tree you can simply use static method convert from the class convertToBST:

```
BinarySearchTree<Integer> bst = convertToBST.convert(root, array);
```

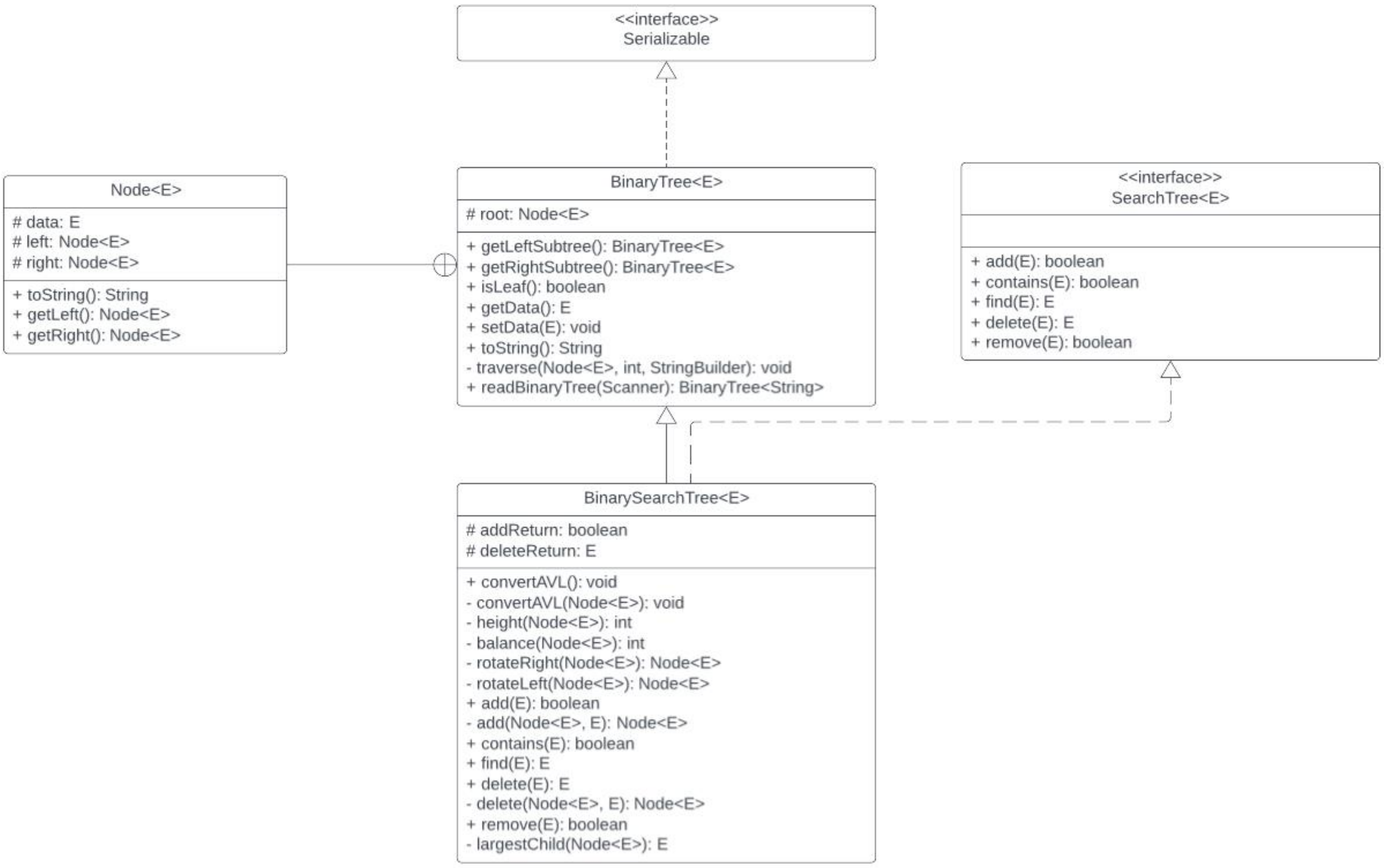
root: binary tree
array: array

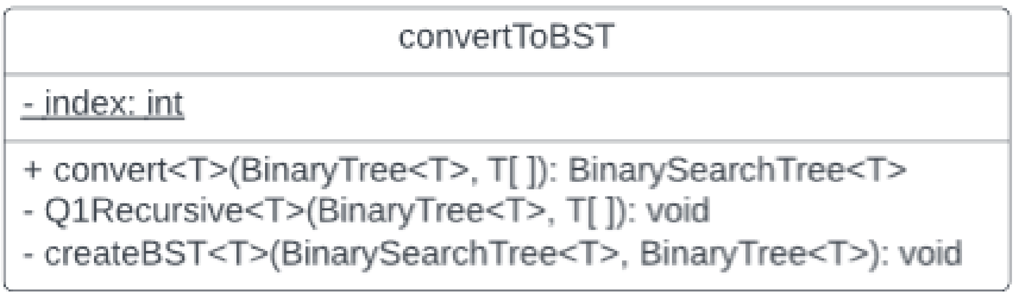
<u>Q2</u>

To convert binary search tree to AVLTree using rotation operations, you should have a binary search tree (bst1), then you can simply use convertAVL() method:

```
bst1.convertAVL();
```

2. CLASS DIAGRAMS





3. PROBLEM SOLUTION APPROACH

Q1

Firstly, I have thought about the design a little bit. After trying different strategies, I have decided this one: Firstly, to make things easier, I have sorted the array by quick sort. Then I have tried to solve the problem recursively. This is the best solution in tree structures as far as I am concerned. I began to fill the binary tree structure with the elements of array in the ascending order starting from bottom-left of the tree. After placing the elements to the binary tree with this approach, all I do is convert this binary tree to binary search tree by adding elements one by one properly.

Q2

Firstly, design process was a bit hard to consider because I couldn’t access the node structure outside the class. After some instructions of the teacher, I have written the method inside the BinarySearchTree class. Then I traverse through the nodes. While traversal, if any unbalanced node is seen, I immediately applied the proper rotation. These rotations provides balanced binary search tree which is AVLTree.

4. TEST CASES

Q1

```java
/*************************TEST FOR CONVERSION FROM BINARY TREE TO BINARY SEARCH TREE******************************/
System.out.println("*********TEST FOR CONVERSION FROM BINARY TREE TO BINARY SEARCH TREE**********");
System.out.println("TEST 1\n------");
Integer[] array = {10, 2, 8, 4, 7};
System.out.printf("Array of elements: ");
for (int i = 0; i < array.length; ++i) System.out.printf("%d ", array[i]);
System.out.println("\n==================================================================");

BinaryTree<Integer> node1 = new BinaryTree(0, null, null);
BinaryTree<Integer> node2 = new BinaryTree(0, null, null);
BinaryTree<Integer> node3 = new BinaryTree(0, node1, node2);
BinaryTree<Integer> node4 = new BinaryTree(0, null, null);
BinaryTree<Integer> root = new BinaryTree(0, node3, node4);
System.out.println("STRUCTURE OF THE BINARY TREE");
System.out.println("----------------------------");
System.out.println(root);
System.out.println("==================================================================");

System.out.println("BINARY SEARCH TREE");
System.out.println("------------------");
BinarySearchTree<Integer> bst = convertToBST.convert(root, array);
System.out.println(bst);
```

```java
System.out.println("\n\n\nTEST 2\n------");
Integer[] array2 = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
System.out.printf("Array of elements: ");
for (int i = 0; i < array2.length; ++i) System.out.printf("%d ", array2[i]);
System.out.println("\n==================================================================");

node1 = new BinaryTree(0, null, null);
node2 = new BinaryTree(0, null, node1);
node3 = new BinaryTree(0, null, node2);
node4 = new BinaryTree(0, null, node3);
BinaryTree<Integer> node5 = new BinaryTree(0, null, node4);
BinaryTree<Integer> node6 = new BinaryTree(0, null, node5);
BinaryTree<Integer> node7 = new BinaryTree(0, null, node6);
BinaryTree<Integer> node8 = new BinaryTree(0, null, node7);
BinaryTree<Integer> node9 = new BinaryTree(0, null, node8);
root = new BinaryTree(0, null, node9);
System.out.println("STRUCTURE OF THE BINARY TREE");
System.out.println("----------------------------");
System.out.println(root);
System.out.println("==================================================================");

System.out.println("BINARY SEARCH TREE");
System.out.println("------------------");
bst = convertToBST.convert(root, array2);
System.out.println(bst);
```

```java
System.out.println("\n\n\nTEST 3\n------");
Integer[] array3 = {42};
System.out.printf("Array of elements: ");
for (int i = 0; i < array3.length; ++i) System.out.printf("%d ", array3[i]);
System.out.println("\n==================================================================");


root = new BinaryTree(0, null, null);
System.out.println("STRUCTURE OF THE BINARY TREE");
System.out.println("----------------------------");
System.out.println(root);
System.out.println("==================================================================");

System.out.println("BINARY SEARCH TREE");
System.out.println("------------------");
bst = convertToBST.convert(root, array3);
System.out.println(bst);
```

Q2

```java
/*************************TEST FOR CONVERSION FROM BINARY SEARCH TREE TO AVL TREE******************************/
System.out.println("\n\n\n************TEST FOR CONVERSION FROM BINARY SEARCH TREE TO AVL TREE*********");
System.out.println("TEST 1\n------");
BinarySearchTree<Integer> bst1 = new BinarySearchTree();
bst1.add(33); bst1.add(53); bst1.add(61);
bst1.add(13); bst1.add(21); bst1.add(11);
bst1.add(8);  bst1.add(9);

System.out.println("BINARY SEARCH TREE\n------------------\n" + bst1);
System.out.println("==================================================================");

bst1.convertAVL();
System.out.println("AVL TREE\n--------\n" + bst1);
System.out.println("==================================================================");

System.out.println("\n\n\nTEST 2\n------");
BinarySearchTree<Integer> bst2 = new BinarySearchTree();
bst2.add(4); bst2.add(3); bst2.add(2);
bst2.add(1); bst2.add(5); bst2.add(6);
bst2.add(7);

System.out.println("BINARY SEARCH TREE\n------------------\n" + bst2);
System.out.println("==================================================================");

bst2.convertAVL();
System.out.println("AVL TREE\n--------\n" + bst2);
System.out.println("==================================================================");

System.out.println("\n\n\nTEST 3\n------");
BinarySearchTree<Integer> bst3 = new BinarySearchTree();
bst3.add(1); bst3.add(2); bst3.add(3);
bst3.add(4); bst3.add(5); bst3.add(6);

System.out.println("BINARY SEARCH TREE\n------------------\n" + bst3);
System.out.println("==================================================================");

bst3.convertAVL();
System.out.println("AVL TREE\n--------\n" + bst3);
System.out.println("==================================================================");
```

5. RUNNING AND RESULTS

```
*********TEST FOR CONVERSION FROM BINARY TREE TO BINARY SEARCH TREE**********
TEST 1
------
Array of elements: 10 2 8 4 7
==================================================================
STRUCTURE OF THE BINARY TREE
----------------------------
0
   0
      0
         null
         null
      0
         null
         null
   0
      null
      null

==================================================================
BINARY SEARCH TREE
------------------
8
   4
      2
         null
         null
      7
         null
         null
   10
      null
      null
```

```
TEST 2
------
Array of elements: 10 9 8 7 6 5 4 3 2 1
==================================================================
STRUCTURE OF THE BINARY TREE
----------------------------
0
   null
   0
      null
      0
         null
         0
            null
            0
               null
               0
                  null
                  0
                     null
                     0
                        null
                        0
                           null
                           null
```

```
==================================================================
BINARY SEARCH TREE
------------------
1
   null
   2
      null
      3
         null
         4
            null
            5
               null
               6
                  null
                  7
                     null
                     8
                        null
                        9
                           null
                           10
                              null
                              null



TEST 3
------
Array of elements: 42
==================================================================
STRUCTURE OF THE BINARY TREE
----------------------------
0
   null
   null

==================================================================
BINARY SEARCH TREE
------------------
42
   null
   null
```

```
************TEST FOR CONVERSION FROM BINARY SEARCH TREE TO AVL TREE*********
TEST 1
------
BINARY SEARCH TREE
------------------
33
   13
      11
         8
            null
            9
               null
               null
         null
      21
         null
         null
   53
      null
      61
         null
         null

==================================================================
AVL TREE
--------
11
   8
      null
      9
         null
         null
   33
      13
         null
         21
            null
            null
      53
         null
         61
            null
            null

==================================================================
```

```
TEST 2
------
BINARY SEARCH TREE
------------------
4
   3
      2
         1
            null
            null
         null
      null
   5
      null
      6
         null
         7
            null
            null

==================================================================
AVL TREE
--------
4
   2
      1
         null
         null
      3
         null
         null
   6
      5
         null
         null
      7
         null
         null

==================================================================
```

```
TEST 3
------
BINARY SEARCH TREE
------------------
1
   null
   2
      null
      3
         null
         4
            null
            5
               null
               6
                  null
                  null

==================================================================
AVL TREE
--------
3
   1
      null
      2
         null
         null
   5
      4
         null
         null
      6
         null
         null

==================================================================
```