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

171044073
Berkan AKIN

1. Problem Definetion

Question 1)
A program should be written on java that will run on all computers. A function will be written for the program. The function must have 2 parameters. Function parameters are a binary tree and an array. array and binnary tree must have equal size elements. The function should put elements inside the binnary tree without breaking the structure.
Question 2)
A program should be written on java that will run on all computers. A function will be written for the program. The function will take a binary search tree as a parameter. We need to convert this tree to avl tree. We have to return the balanced avl tree.
Question 3)
A program should be written on java that will run on all computers. A custom skipList must be implemented. When adding an element to this skipList, the level of the element should be determined according to the highest element on the right and left. We have to set a level by creating a probability. We should increase the maximum level that can be every ten elements, and we should increase the highest level elements by one more level. Other skipList methods should work normally.

2. System Requirements
Question 1)
The system should be able to fill the input binary tree it receives with the array elements it receives.

Question 2)
The system will balance the Binary search tree it receives and turn it into an avl tree.

Question 3)
It is necessary to implement custom skipList. The probability of appending an item to a higher level is calculated by dividing the number of items between two tall neighbors by 10. A function to calculate this distance is required for this. For the nodes to be added, a connect function is required to connect the level levels.

3) Problem Solution Approach
Question 1)
The system should be able to fill the input binary tree it receives with the array elements it receives.
`transformation(E []arr,BinaryTree<E> bt )` Main function In this function, I sort the array and send these elements to the helper function from smallest to largest.
`add(BinaryTree.Node<E> localRoot ,E item)` Helper function
Here, I place the incoming elements in a recursive way from the leftmost to the right. Nodes contain flags. I mark the node I placed as true.
Question 2)
I send all nodes of the given Bst tree from the leftmost to the rightmost to the auxiliary rotatoin() function. He looks at the rotatoion fun balance and calls another auxiliary function. bstRotation(). Bstrotation() does the appropriate rotation.
`transformation(BinarySearchTree bst)` Main function

`bstRotation(BinaryTree.Node<E> root)` Helper function. This function does the appropriate rotation.

Question 3)
First, we need to write the add function. While writing the add function, it is necessary to write an auxiliary connect function. For connecting added node levels. A function that calculates distance needs to be written. When the distance is calculated, the probability must be calculated and given a node level.

GENERAL PROBLEM SOLUTION APPROACH My Problem solution steps are;

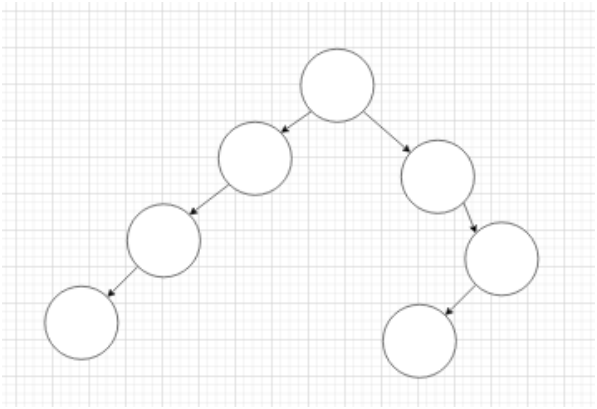
1. – Specify the problem requirements
2. – Analyze the problem
3. – Design an algorithm and Program
4. – Implement the algorithm
5. – Test and verify the program
6. – Maintain and update the program

4) Test cases
Question 1)
Transformation function test #1

```
BinarySearchTree transformation(E []arr,BinaryTree<E> bt )
```

Given input array = `{11,8,13,19,6,4,15};`

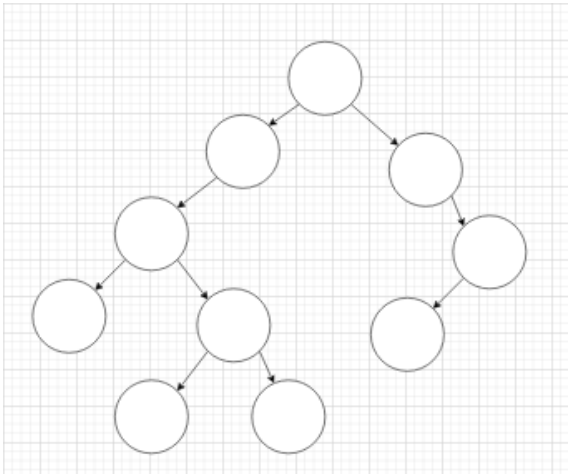
Given binary Tree=



Transformation function test #2

Given input array = `{8,9,6,5,3,7,4,2,1};`

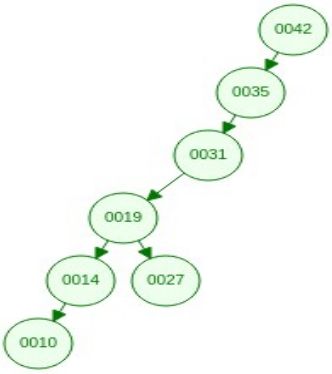
Given binary tree =



Question 2)

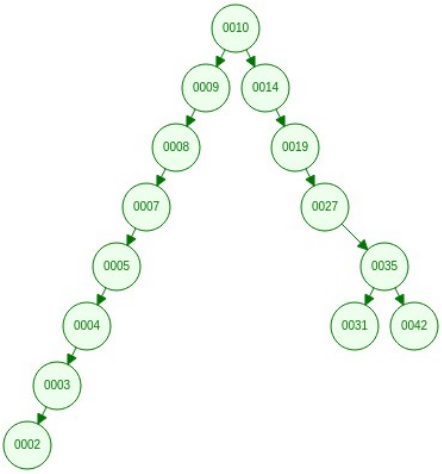
BinarySearchTree transformation(BinarySearchTree bst) function test.
Test #1

Given Binary Search Tree =



Test #2

Given Binary Search Tree =



Question 3)

Adding the first node

```
System.out.println("Question #2 Test");
SkipList<Integer> tmp = new SkipList();
System.out.println("Adding the first node");
tmp.insert(50);
tmp.print();
```

Adding to the back

```
System.out.println("Adding to the back");
tmp.insert(40);
tmp.print();
```

Insertion of the node

```
System.out.println("Insertion of the node");
tmp.insert(45);
tmp.print();
```

Adding at the end

```
System.out.println("Adding at the end");
tmp.insert(100);
tmp.print();
```

Testing the connection between levels

```
System.out.println("Level 2 and Upper Item Test");
tmp.printLevel2();

System.out.println("Level 3 and Upper Item Test");
tmp.printLevel2();

System.out.println("Level 4 and Upper Item Test");
tmp.printLevel2();
```

Leveling up test in multiples of 10

```
if(size > (maxLevel-1)*10) {
    maxLevel++;
    increaseLevel();
    connect();
}
```

```
System.out.println("Leveling up test in multiples of 10");
tmp.print();
System.out.println("After adding the 11th element");
tmp.insert(15);
tmp.print();
```

Calculation of probability based on the number of elements among the largest nodes

```
lenght = findLenght(iter);
level = getLevel(lenght);
SLNode<E> tmp = new SLNode(level,item);
```

Connecting nodes between same levels

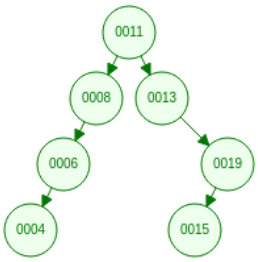
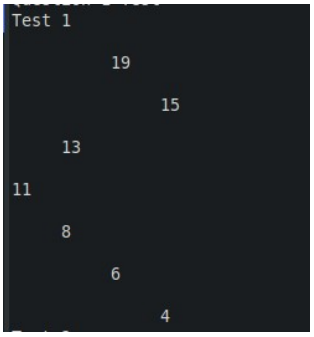
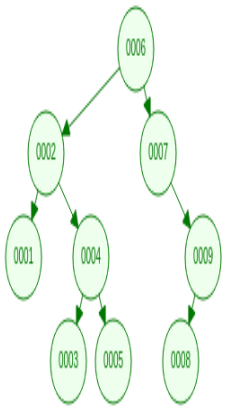
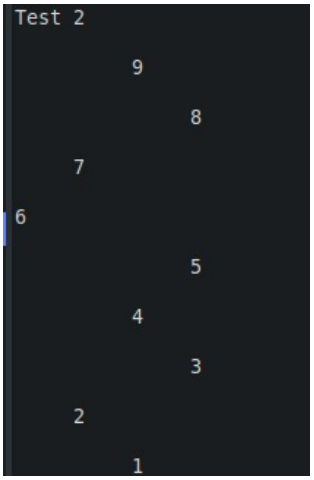
```
private void connect() {
```

Insert run time analaysis

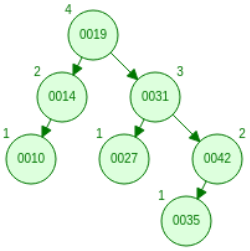
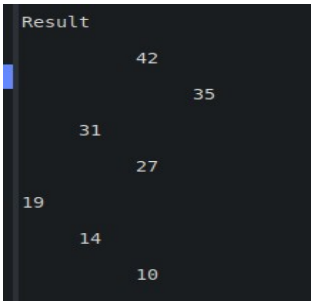
```
System.out.println("Insert run time analaysis");
long time1 = System.nanoTime();
tmp.insert(18);
long time2 = System.nanoTime();
System.out.println("Result time: " + (time2 - time1));
```

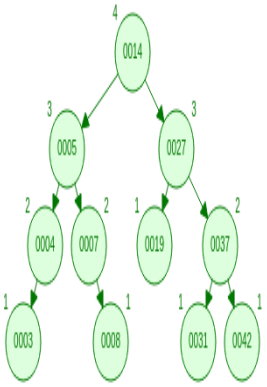
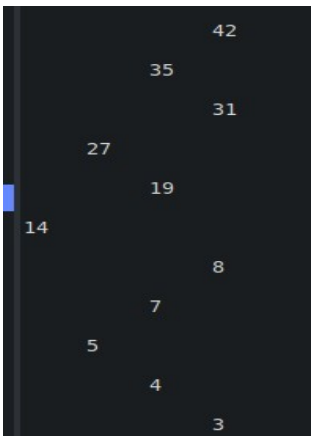
4) Test and Results

Question 1)

Test	Expected	Output	Results
`bst =q1.transformation(arr, bt1);`			Pass
`bst2 =q1.transformation(arr2, bt);`			Pass

Question 2)

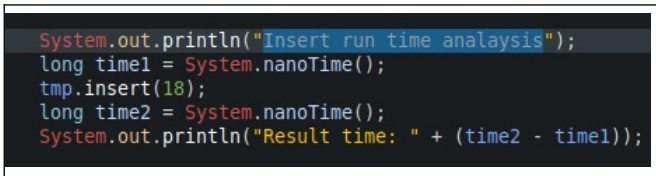
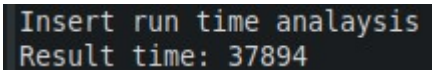
Test	Expected	Output	Results
`bst3 = q2.transformation(bst3);`			Pass

`bst4 = q2.transformation(bst4);`			Pass

Question 3)

Test	Output	Results
`System.out.println("Question #2 Test");` `SkipList<Integer> tmp = new SkipList();` `System.out.println("Adding the first node");` `tmp.insert(50);` `tmp.print();` Add first element	`Question #3 SkipList Test` `Adding the first node` `data: 50  lenght: 2`	Pass
`System.out.println("Adding to the back");` `tmp.insert(40);` `tmp.print();` Add back to the back	`Adding to the back` `data: 40  lenght: 2` `data: 50  lenght: 2`	Pass
`System.out.println("Insertion of the node");` `tmp.insert(45);` `tmp.print();` Insert between to 2 node	`Insertion of the node` `data: 40  lenght: 2` `data: 45  lenght: 1` `data: 50  lenght: 2`	Pass
`System.out.println("Adding at the end");` `tmp.insert(100);` `tmp.print();` Add back to the end	`Adding at the end` `data: 40  lenght: 2` `data: 45  lenght: 1` `data: 50  lenght: 2` `data: 100  lenght: 1`	Pass

`` if(size > (maxLevel-1)*10) { maxLevel++; increaseLevel(); connect(); } ``` Leveling up test in multiples of 10	``` After adding the 11th element data: 10 lenght: 2 data: 15 lenght: 1 data: 20 lenght: 3 data: 25 lenght: 1 data: 30 lenght: 3 data: 35 lenght: 1 data: 40 lenght: 3 data: 42 lenght: 1 data: 45 lenght: 1 data: 50 lenght: 3 data: 100 lenght: 1 ```	Pass
`` lenght = findLenght(iter); level = getLevel(lenght); SLNode<E> tmp = new SLNode(level,item); ``` Calculation of probability based on the number of elements among the largest nodes.	``` Level 2 and Upper Item Test data: 10 lenght: 2 data: 16 lenght: 2 data: 20 lenght: 4 data: 30 lenght: 4 data: 31 lenght: 2 data: 40 lenght: 4 data: 50 lenght: 4 ```	Pass
`` private void connect() { ``` Connecting nodes between same levels.	``` Level 2 and Upper Item Test data: 10 lenght: 2 data: 16 lenght: 2 data: 20 lenght: 4 data: 30 lenght: 4 data: 31 lenght: 2 data: 40 lenght: 4 data: 50 lenght: 4 ``` 2 and bigger node connect to each one	Pass
`` private void connect() { ``` Connecting nodes between same levels.	``` Level 3 and Upper Item Test data: 20 lenght: 4 data: 30 lenght: 4 data: 40 lenght: 4 data: 50 lenght: 4 ```	Pass
`` private void connect() { ``` Connecting nodes between same levels.	``` Level 3 and Upper Item Test data: 20 lenght: 4 data: 30 lenght: 4 data: 40 lenght: 4 data: 50 lenght: 4 ```	Pass

		Pass

Note

*****For question 3, I calculate probability with findLenght() and getLevel() functions. I send the length I get from findLevel() function to getLevel() function. Here I proportion the length as desired. I fill the array with one and zero, then shuffle it and calculate how many multiplex nodes will be with a random number.

5)Run Time Complexity Analysis

Question 1)

Function	Best Case	Worst Case	Average Case
`BinarySearchTree transformation(E []arr,BinaryTree<E> bt )`	θ(1)	$O(n2^n)$	O(nlogn)
`boolean add(BinaryTree.Node<E> localRoot ,E item)`	θ(1)	$O(2^n)$	O(logn)

Question 2)

Function	Best Case	Worst Case	Average Case
`BinarySearchTree transformation(BinarySearchTree bst)`	θ(1)	$θ(n^2)$	θ(nlogn)
`BinaryTree.Node<E> inOrder(BinaryTree.Node<E> root)`	θ(1)	$θ(n^2)$	θ(nlogn)
`BinaryTree.Node<E> rotation(BinaryTree.Node<E> root)`	θ(1)	$θ(n^2)$	θ(nlogn)
`BinaryTree.Node<E> bstRotation(BinaryTree.Node<E> root)`	θ(1)	θ(n)	θ(logn)

Question 3)

Function	Best Case	Worst Case	Average Case
`void insert(E item)`	θ(1)	O(n)	O(logn)
`void insert(E item)`	θ(1)	O(n)	O(logn)
`void increaseLevel()`	θ(1)	O(n)	O(logn)
`int findLenght(SLNode<E> iter)`	θ(1)	θ(n)	θ(n)