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

BURAK KOCAUSTA
1901042605

1. SYSTEM REQUIREMENTS

There are 4 different packages. One of them is for BinarySearchTree implementations. Inside this package, there is BinaryTree implementation, and BinarySearchTree implementation exists. convertToSearchTree() is in BinaryTree class, and convertToAVLTree() method is in BinarySearchTree class. Both method are defined statically.

They must be called like this:

```
BinarySearchTree<Integer> bst1 = BinaryTree.convertToSearchTree(bTree1, arr1);
```

```
BinarySearchTree.convertToAVLTree(bst1);
```

Other package is for CustomSkipList, it can be constructed this way.

```
CustomSkipList<Integer> skip1 = new CustomSkipList<Integer>();
```

For convertToSearchTree() method array size and binary tree sizes are required to be same, and array's elements must be unique. Method does not modify the calling object, it takes BinaryTree as parameter, modifies the BinaryTree as binary search tree, also returns it's BinarySearchTree version.

For convertToAVLTree(), it is also static method and modifies the BinarySearchTree input with rotate operations. Input must be BinarySearchTree.

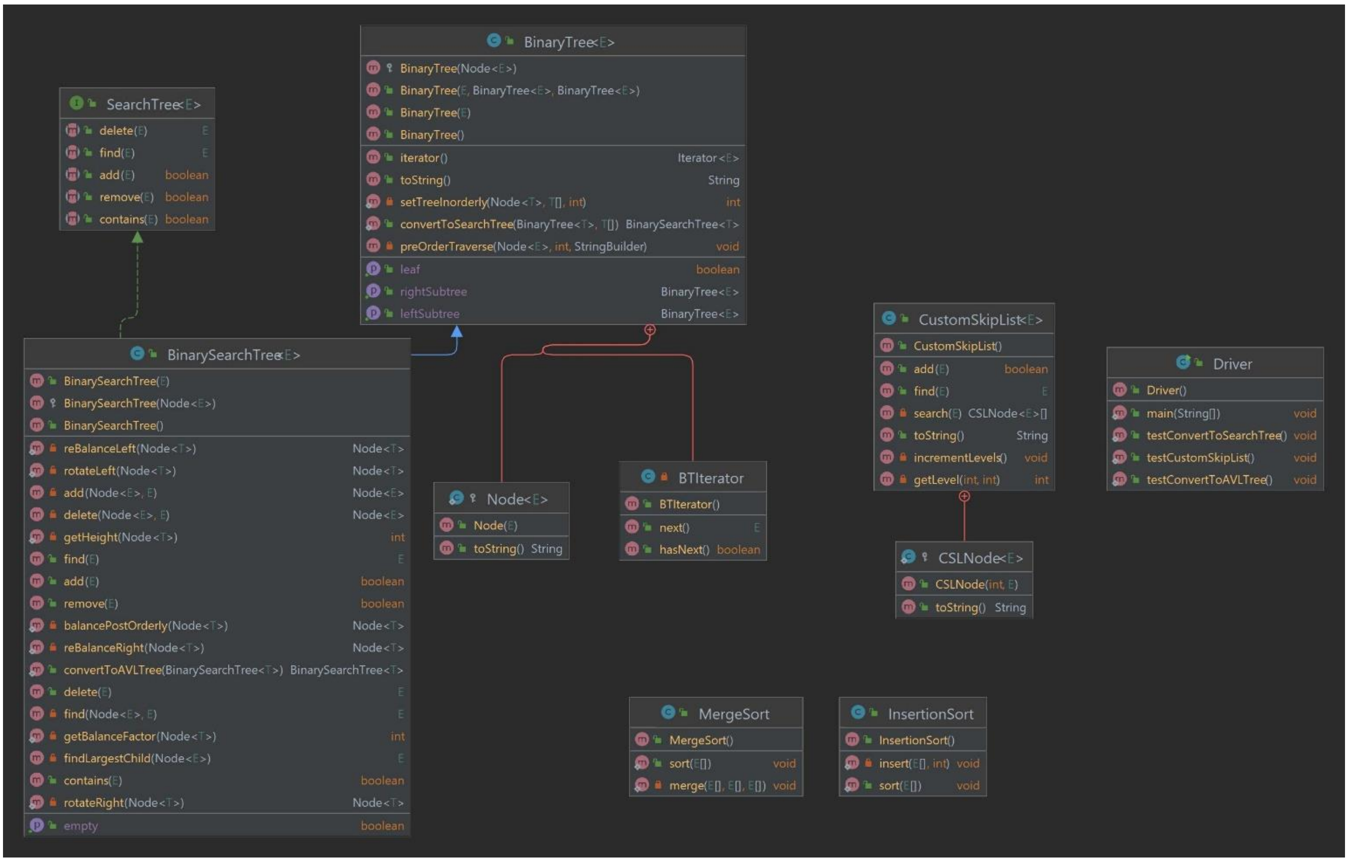
CustomSkipList class has only requirements that is, its type must be comparable.

```
public class CustomSkipList<E extends Comparable<E>>
```

All these implementations are inside these packages, therefore this import statement is required.

```
import BinarySearchTreeGTU.*;
import CustomSkipListGTU.*;
```

2. CLASS DIAGRAM



3. PROBLEM SOLUTION APPROACH

Q1

In question 1, it wanted to convert binary tree to binary search tree with given inputs in an array. First problem I encountered is, accessing the nodes of binary tree. One of the possible solution is making binary tree's node class public. I don't want to do that, because I don't want to modify binary tree's current structure, it must be the same accessibility state as before. Then, I decided to define public static method which takes binary tree and converts it to binary search tree. It is static because it can access the static node class. Binary Tree's node class is defined statically. After deciding where to implement algorithm, I decide how algorithm works. In binary tree, if data's are written inorderly, result is an inorder set. So, algorithm's first step is sorting the array. I used the merge sort algorithm for this. After sorting the array, It is assumed that array size and tree size is equal, binary tree is traversed inorderly each node is set to the array's index. Problem I encountered in this traversing is, tracking the array index. Traversing is made recursively, because of this tracking array index is a little bit difficult. Solution is returning the current array index. In this way, index is less for the inorder predecessor. It increases with left root right way. After setting the binary tree, it is wanted to return binary search tree. So I move iterator implemented in binary search tree in previous homework, to binary tree. Using this iterator, I created the binary search tree, and returned it.

Q2

In question 2, it is wanted to convert binary search tree to AVL tree with using rotation operations. There are many solutions to solve this algorithm, but as I understand, binary search tree is wanted to be modified with rotations, because of that I returned the AVL version of binary search tree input, but its type is not AVLTree class BinarySearchTree class, it is just converted to AVLTree type structure with rotations. I have the same accessibility problems as in question 1, but the solution is same with it. I defined the static methods inside BinarySearchTree class. After deciding where to implement, input and return types, the only problem left is how to implement. There are two ways to balance the tree directionally, it can be from root, or it can be from leaves. It is known that from AVLTree implementation, rotations are made from leaves to root. Some problems are occurred while constructing the algorithm, one of them is how do I know if root is balanced. Because binary search tree's node has not balance factor in its fields. So, It is impossible to know if node is balanced or not constantly. I don't want to modify binary search tree's node because of the reason I explained for question 1, so that I decided an inefficient solution for this problem. Algorithm must work for regular SearchTree's, if I add, height information for nodes, it becomes an AVL tree not binary search tree. Solution is simple, to check each nodes balancing state, count its left and right child's height recursively. I decided to balance the tree from leaves, therefore it is obvious that tree must be traversed post orderly. While traversing check each nodes balance factor, make rotation according to that balance factor. This rotation operation is similar with AVL tree's operations. Every case in AVL tree's balancing which are left-left, left-right,

right-right, and right-left must be checked in this also. After balancing a node’s left subtree, its right subtree needs to be checked, because they might become unbalanced after rotations. So, every node is checked for balancing factor. In this way tree is balanced from leaves to root.

Q3

In question 3, It is wanted to define a custom skip list structure which has insertion operation that have differences from regular skip list implementation. The differences occur when reallocating the links and determining the level of new added node. Other difference is, when a new level is added, all tall items must go to one upper level. In homework pdf it is said that, reallocate happens according to the powers of 10, but in PS It is also said that, it will happens 10 by 10. I decided to do it according to the PS. After deciding how to do it, thought about how to count left and right nodes between left tall or right tall item and inserted item. I decided to go from pred array which returned from private search method. pred[1] will always upper item(except there is no tall item in the skip list), with this way I can access right tall and left tall. I made all count operations with pred[1] information. It gave the possibility to access left tall and right tall items. After counting the number of left and right, implementing the probability calculation is left. Point is making at least one tall item for 10 elements. So I used an array of 10, and insert 1’s to that array to the number of left+right. Then I randomly choose its element till it comes to max level or fails. My last step is implementing the reallocation part which is incrementing the tall items when level is increased. I thought this part to come from last level to first level. Incrementing each level in one tour of the loop. So outer loop iterates max level – 2 times, inner loop iterates according to number of elements for that level. Firstly I thought it might be done without outer loop, but it gets complicated. With this way problem is solved easily.

4. Complexity Analysis

Q1

```
public static <T extends Comparable<T>> BinarySearchTree<T> convertToSearchTree(BinaryTree<T> binaryTree, T[] arr) {

    // sort the array
    MergeSort.sort(arr);
    setTreeInorderly(binaryTree.root, arr, 0);
    BinarySearchTree<T> bst = new BinarySearchTree<T>();

    // traverse with iterator for the bst class version
    for(T data: binaryTree) {
        bst.add(data);
    }
    return bst;
}
```

$\Theta(nlogn)$

$\Theta(n)$

$\Theta(nlogn)$ (average)

```
private static <T extends Comparable<T>> int setTreeInorderly(Node<T> localRoot, T[] arr, int cur) {

    if ( cur >= arr.length || localRoot == null )
        return cur;

    cur = setTreeInorderly(localRoot.left, arr, cur);
    localRoot.data = arr[cur];
    return setTreeInorderly(localRoot.right, arr, cur+1);
}
```

$\Theta(1)$

$\Theta(n)$

Time complexity of this operation is **Θ(nlogn)** averagely because of the merge sort and creating bst class. It could be **O(n^2)** in the worst case.(more explanation of algorithm is at the problem solution approach part)

$T_{av}(n)$ = **Θ(nlogn),** $T_w(n)$ = **O(n^2)**

Q2

```
public static <T extends Comparable<T>> BinarySearchTree<T> convertToAVLTree ( BinarySearchTree<T> bst ) {

    bst.root = balancePostOrderly(bst.root);
    return bst;
}
```

```java
private static <T extends Comparable<T>> Node<T> balancePostOrderly ( Node<T> localRoot ) {

    if ( localRoot == null )
        return null;

    localRoot.left = balancePostOrderly(localRoot.left);      } 2T(n/2)
    localRoot.right = balancePostOrderly(localRoot.right);    }

    int balance = getBalanceFactor(localRoot);  → Θ(n)

    // left is heavy
    if ( balance < -1 ) {
        localRoot = reBalanceLeft(localRoot);  → Θ(n)
        localRoot.right = balancePostOrderly(localRoot.right);  → T(n/2)
    }

    // right is heavy
    if( balance > 1 ) {
        localRoot = reBalanceRight(localRoot);  → Θ(n)
        localRoot.left = balancePostOrderly(localRoot.left);  → T(n/2)
    }

    return localRoot;  → Θ(1)
}
```

```java
private static <T extends Comparable<T>> int getBalanceFactor ( Node<T> localRoot ) {

    if ( localRoot == null )    } Θ(1)
        return 0;               }

    int left = getHeight(localRoot.left);     } → Θ(n)
    int right = getHeight(localRoot.right);   }
    return right - left;  → Θ(1)
}
```

```java
private static <T extends Comparable<T>> int getHeight ( Node<T> localRoot ) {

    if ( localRoot == null )    } Θ(1)
        return 0;               }

    int left = getHeight(localRoot.left);     } → Θ(n)
    int right = getHeight(localRoot.right);   }
    return 1 + ((right >= left) ? right : left);  → Θ(1)
}
```

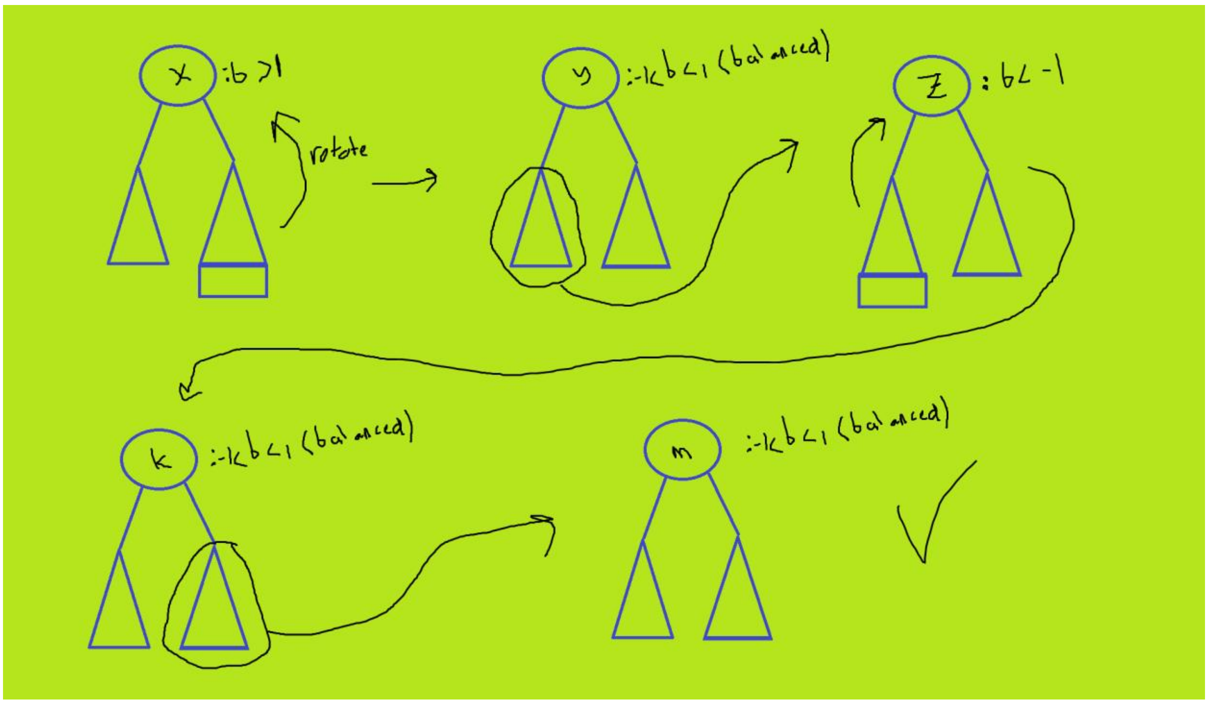
```java
private static <T extends Comparable<T>> Node<T> reBalanceLeft ( Node<T> localRoot ) {
    int balance = getBalanceFactor(localRoot.left);  → Θ(n)

    // left right
    if ( balance > 0 )                                   } Θ(1)
        localRoot.left = rotateLeft(localRoot.left);     }


    // left left and left balanced
    return rotateRight(localRoot);  → Θ(1)
}
```

```java
private static <T extends Comparable<T>> Node<T> reBalanceRight ( Node<T> localRoot ) {
    int balance = getBalanceFactor(localRoot.right); → Θ(n)

    // right left
    if ( balance < 0 )
        localRoot.right = rotateRight(localRoot.right); } Θ(1)

    // left left and left balanced
    return rotateLeft(localRoot); → Θ(1)
}
```

Complexity is **Θ(n^2) in the best case**(if there is no rotation). First n comes from number of elements all nodes need to be checked. Other n comes from calculating the balance factor its time complexity is Θ(n). Worst case happens when rotation happens. Because, after each rotation one of the subtrees must checked again. If left subtree is heavy, after right rotation right subtree must be checked, it is opposite when right subtree is heavy. Therefore, in the **worst case it becomes averagely O(n^3).**

$T_b(n)$ = **Θ(n^2),** $T_w(n)$ = **O(n^3),** T(n) **= O(n^3)**

When rotation happens, algorithm works like this:



Q3

```java
public E find ( E target ) {
    CSLNode<E>[] pred = search(target);
    if ( pred[0].links != null && pred[0].links[0].data != null && pred[0].links[0].data.compareTo(target) == 0 )
        return pred[0].links[0].data;
    return null;
}
```

→ O(logn)

```java
private CSLNode<E>[] search ( E target ) {
    CSLNode<E>[] pred = (CSLNode<E>[]) new CSLNode[maxLevel];
    CSLNode<E> cur = head;
    for ( int i = cur.links.length - 1; i >= 0; i-- ) {

        while( cur.links[i] != null && cur.links[i].data.compareTo(target) < 0 )
            cur = cur.links[i];
        pred[i] = cur;
    }
    return pred;
}
```

O(logn)

find() and search() methods are **O(logn) averagely**, because in each traverse, almost half of the list is left out. If it is skewed skip list worst case happens an it is O(n).

T(n) = O(logn) (average)

```java
public boolean add ( E item ) {
    size++;
    CSLNode<E>[] pred = search(item);
    if ( size > maxCap ) {

        maxLevel++;
        maxCap = size + 10;
        head.links = Arrays.copyOf(head.links, maxLevel);
        pred = Arrays.copyOf(pred, maxLevel);
        pred[maxLevel - 1] = head;
        incrementLevels();  // increment the tall item's levels
    }

    int left = 0;
    int right = 0;
    CSLNode<E> cur = pred[1];

    // count left
    while ( cur != null && (cur != head && cur.data.compareTo(item) < 0) ) {
        left++;
        cur = cur.links[0];
    }

    // count right
    if ( pred[1].links[1] != null ) {
        cur = pred[0];
        while ( cur != null && (cur != head && cur.data.compareTo(pred[1].links[1].data) != 0) ) {
            right++;
            cur = cur.links[0];
        }
    }
    int level = getLevel(left, right);
    CSLNode<E> newNode = new CSLNode<E>(level, item);

    for ( int i = 0; i < newNode.links.length; i++ ) {
        newNode.links[i] = pred[i].links[i];
        pred[i].links[i] = newNode;
    }

    return true;
}
```

→ O(logn) (av)

} O(n²)

} O(1)

m < n

} O(m)

→ O(k) k < n

→ O(1)

} → O(1)

```java
private int getLevel ( int left, int right ) {
    if ( size == 1 )
        return 2;
    if (left == 0 && right == 0)
        return 1;

    int level = 1;

    int[] arr = new int[10];

    int chance = left + right;
    int i = 0;
    while ( i < chance && i < arr.length ) {
        arr[i] = 1;
        ++i;
    }

    while ( level < maxLevel ) {
        Random rand = new Random();
        int index = rand.nextInt(arr.length);
        if ( arr[index] == 1 )
            level++;
        else
            return level;
    }
    return level;
}
```

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$ (av)

```java
private void incrementLevels ( ) {

    for ( int i = maxLevel - 2; i >= 1; --i ) {
        CSLNode<E> pre = head;
        CSLNode<E> pos = head.links[i];


        while ( pos != null ) {

            if ( pos.links.length == i + 1 ) {
                pos.links = Arrays.copyOf(pos.links, pos.links.length + 1);
                pos.links[i+1] = pre.links[i+1];
                pre.links[i+1] = pos;
            }
            pos = pos.links[i];
            pre = pre.links[i+1];
        }
    }
}
```

→ m times

$\Theta(n^2)$ (av)

$n \rightarrow$ num of element

Add operation's time complexity is **O(n²)** because of the incrementLevels() method. It is averagely traverses whole nodes to increment (except first level nodes), and makes copy of the arrays. Normally it could be amortized, but reallocation happens 10 by 10 so it is not amortized. Counting left and right nodes could be thought as constant because it is averagely traverses very small portion of the list. Calculating probability is also thought as constant time because it can iterates at most max level times. Max level is very small compared to the size of the list.(more explanation of the algorithm is at the problem solution approach.)

T(n) = O(n²)

5. TEST CASES

```java
public static void testConvertToSearchTree ( ) {

    System.out.println("\n____Testing convertToSearchTree()____\n");
    System.out.println("\nBinary Tree:");
    BinaryTree<Integer> bTree1 = new BinaryTree<Integer>(15
                                                         , new BinaryTree<Integer>(23
                                                             , new BinaryTree<Integer>(45)
                                                             , new BinaryTree<Integer>(11))
                                                         , new BinaryTree<Integer>(51
                                                             , new BinaryTree<Integer>(12), new BinaryTree<Integer>(13
                                                                 , new BinaryTree<Integer>(16, new BinaryTree<Integer>(12), null)
                                                         , new BinaryTree<Integer>(11)))
                                                         );

    System.out.println(bTree1);

    System.out.print("array: ");
    Integer[] arr1 = new Integer[]{3,7,1,21,15,2,6,8,93,72};
    for ( int i = 0; i < arr1.length; ++i )
        System.out.print(arr1[i] + " ");
    System.out.print("\n");
    BinarySearchTree<Integer> bst1 = BinaryTree.convertToSearchTree(bTree1, arr1);

    System.out.println("\nBinary Search Tree: \n" + bst1 + "-------------");
```

```java
System.out.println("\nBinary Tree:");
BinaryTree<Integer> bTree2 = new BinaryTree<Integer>(15
                                                     , new BinaryTree<Integer>(3
                                                         , new BinaryTree<Integer>(4, new BinaryTree<Integer>(1), null)
                                                         , new BinaryTree<Integer>(11, null, new BinaryTree<Integer>(5)))
                                                     , new BinaryTree<Integer>(1
                                                         , new BinaryTree<Integer>(12,
                                                             new BinaryTree<Integer>(1, new BinaryTree<Integer>(4),
                                                                 new BinaryTree<Integer>(1)), null)
                                                         , new BinaryTree<Integer>(12, new BinaryTree<Integer>(6,
                                                             new BinaryTree<Integer>(2), null)
                                                     , new BinaryTree<Integer>(1)))
                                                     );

System.out.println(bTree2);

System.out.print("array: ");
Integer[] arr2 = new Integer[]{8, 4, 11, 5, 2, 9, 16, 21, 31, 3, 95, 51, 99, 71, 64};
for ( int i = 0; i < arr2.length; ++i )
    System.out.print(arr2[i] + " ");
System.out.print("\n");
BinarySearchTree<Integer> bst2 = BinaryTree.convertToSearchTree(bTree2, arr2);

System.out.println("\nBinary Search Tree: \n" + bst2 + "-------------");
```

```java
public static void testConvertToAVLTree ( ) {

    System.out.println("\n____Testing convertToAVLTree()____\n");
    BinarySearchTree<Integer> bst1 = new BinarySearchTree<Integer>();

    for ( int i = 0; i < 4; ++i ) {
        bst1.add(i+1);
    }

    System.out.println( "Binary Search Tree:\n" + bst1);

    BinarySearchTree.convertToAVLTree(bst1);

    System.out.println( "After converting to AVL Tree:\n" + bst1 + "------------");


    BinarySearchTree<Integer> bst2 = new BinarySearchTree<Integer>();

    for ( int i = 0; i < 10; ++i ) {
        bst2.add((i+1)*3);
    }

    System.out.println( "Binary Search Tree:\n" + bst2);

    BinarySearchTree.convertToAVLTree(bst2);

    System.out.println( "After converting to AVL Tree:\n" + bst2 + "------------");
```

```java
BinarySearchTree<Integer> bst3 = new BinarySearchTree<Integer>();

for ( int i = 10; i > 0; --i ) {
    bst3.add((i+1)*3);
}

System.out.println( "Binary Search Tree:\n" + bst3);

BinarySearchTree.convertToAVLTree(bst3);

System.out.println( "After converting to AVL Tree:\n" + bst3 + "------------");

BinarySearchTree<Integer> bst4 = new BinarySearchTree<Integer>();
bst4.add(10);
bst4.add(7);
bst4.add(5);
bst4.add(1);
bst4.add(15);
bst4.add(25);
bst4.add(31);

System.out.println( "Binary Search Tree:\n" + bst4);

BinarySearchTree.convertToAVLTree(bst4);

System.out.println( "After converting to AVL Tree:\n" + bst4 + "------------");
```

```java
public static void testCustomSkipList() {

    System.out.println("\n____Testing CustomSkipList____\n");
    CustomSkipList<Integer> skip1 = new CustomSkipList<Integer>();

    System.out.println("Creating an empty skip list, and inserting elements one by one\n" + skip1 + "\n------------\n");

    skip1.add(7);
    System.out.println(skip1 + "\n------------\n");

    skip1.add(3);
    System.out.println(skip1 + "\n------------\n");

    skip1.add(15);
    System.out.println(skip1 + "\n------------\n");

    skip1.add(11);
    System.out.println(skip1 + "\n------------\n");

    skip1.add(81);
    System.out.println(skip1 + "\n------------\n");

    skip1.add(54);
    System.out.println(skip1 + "\n------------\n");

    skip1.add(37);
    System.out.println(skip1 + "\n------------\n");

    skip1.add(19);
    System.out.println(skip1 + "\n------------\n");

    skip1.add(74);
    System.out.println(skip1 + "\n------------\n");
```

```java
skip1.add(13);
System.out.println(skip1 + "\n------------\n");

skip1.add(14);
System.out.println(skip1 + "\n------------\n");

skip1.add(63);
System.out.println( "maximum level increased, tall item's are appended one level upper list.\n" + skip1 + "\n------------\n");

skip1.add(71);
System.out.println(skip1 + "\n------------\n");

skip1.add(68);
System.out.println(skip1 + "\n------------\n");

skip1.add(82);
System.out.println(skip1 + "\n------------\n");

skip1.add(1);
System.out.println(skip1 + "\n------------\n");

skip1.add(4);
System.out.println(skip1 + "\n------------\n");

skip1.add(8);
System.out.println(skip1 + "\n------------\n");

skip1.add(12);
System.out.println(skip1 + "\n------------\n");

skip1.add(99);
System.out.println(skip1 + "\n------------\n");
```

```java
skip1.add(101);
System.out.println(skip1 + "\n------------\n");

skip1.add(69);
System.out.println( "maximum level increased, tall item's are appended one level upper list.\n" + skip1 + "\n------------\n");

skip1.add(58);
System.out.println(skip1 + "\n------------\n");

skip1.add(16);
System.out.println(skip1 + "\n------------\n");


System.out.println("skip1.find(99) = " + skip1.find(99));
System.out.println("skip1.find(11) = " + skip1.find(11));
System.out.println("skip1.find(4) = " + skip1.find(4));
System.out.println("(not exist)skip1.find(2) = " + skip1.find(2));
System.out.println("skip1.find(15) = " + skip1.find(15));
System.out.println("(not exist)skip1.find(24) = " + skip1.find(24));
System.out.println("(not exist)skip1.find(66) = " + skip1.find(66));
```

6. RUNNING AND RESULTS

```
____Testing convertToSearchTree()____


Binary Tree:
15
  23
    45
      null
      null
    11
      null
      null
  51
    12
      null
      null
    13
      16
        12
          null
          null
        null
      11
        null
        null

array: 3 7 1 21 15 2 6 8 93 72
```

```
Binary Search Tree:
6
  2
    1
      null
      null
    3
      null
      null
  8
    7
      null
      null
    72
      21
        15
          null
          null
        null
      93
        null
        null
--------------
```

```
Binary Tree:
15
  3
    4
      1
        null
        null
      null
    11
      null
      5
        null
        null
  1
    12
      1
        4
          null
          null
        1
          null
          null
      null
    12
      6
        2
          null
          null
        null
      1
        null
        null

array: 8 4 11 5 2 9 16 21 31 3 95 51 99 71 64
```

```
Binary Search Tree:
9
  4
    3
      2
        null
        null
      null
    5
      null
      8
        null
        null
  51
    31
      16
        11
          null
          null
        21
          null
          null
      null
    95
      71
        64
          null
          null
        null
      99
        null
        null
--------------
```

```
____Testing convertToAVLTree()____

Binary Search Tree:
1
  null
  2
    null
    3
      null
      4
        null
        null

After converting to AVL Tree:
3
  1
    null
    2
      null
      null
  4
    null
    null
------------
```

```
Binary Search Tree:
3
  null
  6
    null
    9
      null
      12
        null
        15
          null
          18
            null
            21
              null
              24
                null
                27
                  null
                  30
                    null
                    null

After converting to AVL Tree:
15
  9
    3
      null
      6
        null
        null
    12
      null
      null
  21
    18
      null
      null
    27
      24
        null
        null
      30
        null
        null
------------
```

```
Binary Search Tree:
33
  30
    27
      24
        21
          18
            15
              12
                9
                  6
                    null
                    null
                  null
                null
              null
            null
          null
        null
      null
    null
  null

After converting to AVL Tree:
21
  15
    9
      6
        null
        null
      12
        null
        null
    18
      null
      null
  27
    24
      null
      null
    33
      30
        null
        null
      null
------------
```

```
Binary Search Tree:
10
  7
    5
      1
        null
        null
      null
    null
  15
    null
    25
      null
      31
        null
        null

After converting to AVL Tree:
10
  5
    1
      null
      null
    7
      null
      null
  25
    15
      null
      null
    31
      null
      null
------------
```

```
____Testing CustomSkipList____

Creating an empty skip list, and inserting elements one by one
Empty Skip List
------------

Maximum Level = 4, size = 1
, (Level = 2, data = [7])
------------

Maximum Level = 4, size = 2
, (Level = 1, data = [3]), (Level = 2, data = [7])
------------

Maximum Level = 4, size = 3
, (Level = 1, data = [3]), (Level = 2, data = [7]), (Level = 1, data = [15])

------------

Maximum Level = 4, size = 4
, (Level = 1, data = [3]), (Level = 2, data = [7]), (Level = 1, data = [11])
, (Level = 1, data = [15])
------------

Maximum Level = 4, size = 5
, (Level = 1, data = [3]), (Level = 2, data = [7]), (Level = 1, data = [11])
, (Level = 1, data = [15]), (Level = 2, data = [81])
------------

Maximum Level = 4, size = 6
, (Level = 1, data = [3]), (Level = 2, data = [7]), (Level = 1, data = [11])
, (Level = 1, data = [15]), (Level = 1, data = [54]), (Level = 2, data = [81])

------------

Maximum Level = 4, size = 7
, (Level = 1, data = [3]), (Level = 2, data = [7]), (Level = 1, data = [11])
, (Level = 1, data = [15]), (Level = 1, data = [37]), (Level = 1, data = [54])
, (Level = 2, data = [81])
------------

Maximum Level = 4, size = 8
, (Level = 1, data = [3]), (Level = 2, data = [7]), (Level = 1, data = [11])
, (Level = 1, data = [15]), (Level = 4, data = [19]), (Level = 1, data = [37])
, (Level = 1, data = [54]), (Level = 2, data = [81])
------------
```

```
Maximum Level = 4, size = 9
, (Level = 1, data = [3]), (Level = 2, data = [7]), (Level = 1, data = [11])
, (Level = 1, data = [15]), (Level = 4, data = [19]), (Level = 1, data = [37])
, (Level = 1, data = [54]), (Level = 2, data = [74]), (Level = 2, data = [81])

-----------

Maximum Level = 4, size = 10
, (Level = 1, data = [3]), (Level = 2, data = [7]), (Level = 1, data = [11])
, (Level = 4, data = [13]), (Level = 1, data = [15]), (Level = 4, data = [19])
, (Level = 1, data = [37]), (Level = 1, data = [54]), (Level = 2, data = [74])
, (Level = 2, data = [81])
-----------

Maximum Level = 5, size = 11
, (Level = 1, data = [3]), (Level = 3, data = [7]), (Level = 1, data = [11])
, (Level = 5, data = [13]), (Level = 1, data = [14]), (Level = 1, data = [15])
, (Level = 5, data = [19]), (Level = 1, data = [37]), (Level = 1, data = [54])
, (Level = 3, data = [74]), (Level = 3, data = [81])
-----------

maximum level increased, tall item's are appended one level upper list.
Maximum Level = 5, size = 12
, (Level = 1, data = [3]), (Level = 3, data = [7]), (Level = 1, data = [11])
, (Level = 5, data = [13]), (Level = 1, data = [14]), (Level = 1, data = [15])
, (Level = 5, data = [19]), (Level = 1, data = [37]), (Level = 1, data = [54])
, (Level = 1, data = [63]), (Level = 3, data = [74]), (Level = 3, data = [81])

-----------

Maximum Level = 5, size = 13
, (Level = 1, data = [3]), (Level = 3, data = [7]), (Level = 1, data = [11])
, (Level = 5, data = [13]), (Level = 1, data = [14]), (Level = 1, data = [15])
, (Level = 5, data = [19]), (Level = 1, data = [37]), (Level = 1, data = [54])
, (Level = 1, data = [63]), (Level = 1, data = [71]), (Level = 3, data = [74])
, (Level = 3, data = [81])
-----------

Maximum Level = 5, size = 14
, (Level = 1, data = [3]), (Level = 3, data = [7]), (Level = 1, data = [11])
, (Level = 5, data = [13]), (Level = 1, data = [14]), (Level = 1, data = [15])
, (Level = 5, data = [19]), (Level = 1, data = [37]), (Level = 1, data = [54])
, (Level = 1, data = [63]), (Level = 1, data = [68]), (Level = 1, data = [71])
, (Level = 3, data = [74]), (Level = 3, data = [81])
-----------
```

```
Maximum Level = 5, size = 15
, (Level = 1, data = [3]), (Level = 3, data = [7]), (Level = 1, data = [11])
, (Level = 5, data = [13]), (Level = 1, data = [14]), (Level = 1, data = [15])
, (Level = 5, data = [19]), (Level = 1, data = [37]), (Level = 1, data = [54])
, (Level = 1, data = [63]), (Level = 1, data = [68]), (Level = 1, data = [71])
, (Level = 3, data = [74]), (Level = 3, data = [81]), (Level = 1, data = [82])

-----------

Maximum Level = 5, size = 16
, (Level = 1, data = [1]), (Level = 1, data = [3]), (Level = 3, data = [7])
, (Level = 1, data = [11]), (Level = 5, data = [13]), (Level = 1, data = [14])
, (Level = 1, data = [15]), (Level = 5, data = [19]), (Level = 1, data = [37])
, (Level = 1, data = [54]), (Level = 1, data = [63]), (Level = 1, data = [68])
, (Level = 1, data = [71]), (Level = 3, data = [74]), (Level = 3, data = [81])
, (Level = 1, data = [82])
-----------

Maximum Level = 5, size = 17
, (Level = 1, data = [1]), (Level = 1, data = [3]), (Level = 1, data = [4])
, (Level = 3, data = [7]), (Level = 1, data = [11]), (Level = 5, data = [13])
, (Level = 1, data = [14]), (Level = 1, data = [15]), (Level = 5, data = [19])
, (Level = 1, data = [37]), (Level = 1, data = [54]), (Level = 1, data = [63])
, (Level = 1, data = [68]), (Level = 1, data = [71]), (Level = 3, data = [74])
, (Level = 3, data = [81]), (Level = 1, data = [82])
-----------

Maximum Level = 5, size = 18
, (Level = 1, data = [1]), (Level = 1, data = [3]), (Level = 1, data = [4])
, (Level = 3, data = [7]), (Level = 3, data = [8]), (Level = 1, data = [11])
, (Level = 5, data = [13]), (Level = 1, data = [14]), (Level = 1, data = [15])
, (Level = 5, data = [19]), (Level = 1, data = [37]), (Level = 1, data = [54])
, (Level = 1, data = [63]), (Level = 1, data = [68]), (Level = 1, data = [71])
, (Level = 3, data = [74]), (Level = 3, data = [81]), (Level = 1, data = [82])

-----------

Maximum Level = 5, size = 19
, (Level = 1, data = [1]), (Level = 1, data = [3]), (Level = 1, data = [4])
, (Level = 3, data = [7]), (Level = 3, data = [8]), (Level = 1, data = [11])
, (Level = 1, data = [12]), (Level = 5, data = [13]), (Level = 1, data = [14])
, (Level = 1, data = [15]), (Level = 5, data = [19]), (Level = 1, data = [37])
, (Level = 1, data = [54]), (Level = 1, data = [63]), (Level = 1, data = [68])
, (Level = 1, data = [71]), (Level = 3, data = [74]), (Level = 3, data = [81])
, (Level = 1, data = [82])
-----------
```

```
Maximum Level = 5, size = 20
, (Level = 1, data = [1]), (Level = 1, data = [3]), (Level = 1, data = [4])
, (Level = 3, data = [7]), (Level = 3, data = [8]), (Level = 1, data = [11])
, (Level = 1, data = [12]), (Level = 5, data = [13]), (Level = 1, data = [14])
, (Level = 1, data = [15]), (Level = 5, data = [19]), (Level = 1, data = [37])
, (Level = 1, data = [54]), (Level = 1, data = [63]), (Level = 1, data = [68])
, (Level = 1, data = [71]), (Level = 3, data = [74]), (Level = 3, data = [81])
, (Level = 1, data = [82]), (Level = 1, data = [99])
-----------

Maximum Level = 5, size = 21
, (Level = 1, data = [1]), (Level = 1, data = [3]), (Level = 1, data = [4])
, (Level = 3, data = [7]), (Level = 3, data = [8]), (Level = 1, data = [11])
, (Level = 1, data = [12]), (Level = 5, data = [13]), (Level = 1, data = [14])
, (Level = 1, data = [15]), (Level = 5, data = [19]), (Level = 1, data = [37])
, (Level = 1, data = [54]), (Level = 1, data = [63]), (Level = 1, data = [68])
, (Level = 1, data = [71]), (Level = 3, data = [74]), (Level = 3, data = [81])
, (Level = 1, data = [82]), (Level = 1, data = [99]), (Level = 1, data = [101])

-----------

maximum level increased, tall item's are appended one level upper list.
Maximum Level = 6, size = 22
, (Level = 1, data = [1]), (Level = 1, data = [3]), (Level = 1, data = [4])
, (Level = 4, data = [7]), (Level = 4, data = [8]), (Level = 1, data = [11])
, (Level = 1, data = [12]), (Level = 6, data = [13]), (Level = 1, data = [14])
, (Level = 1, data = [15]), (Level = 6, data = [19]), (Level = 1, data = [37])
, (Level = 1, data = [54]), (Level = 1, data = [63]), (Level = 1, data = [68])
, (Level = 5, data = [69]), (Level = 1, data = [71]), (Level = 4, data = [74])
, (Level = 4, data = [81]), (Level = 1, data = [82]), (Level = 1, data = [99])
, (Level = 1, data = [101])
-----------

Maximum Level = 6, size = 23
, (Level = 1, data = [1]), (Level = 1, data = [3]), (Level = 1, data = [4])
, (Level = 4, data = [7]), (Level = 4, data = [8]), (Level = 1, data = [11])
, (Level = 1, data = [12]), (Level = 6, data = [13]), (Level = 1, data = [14])
, (Level = 1, data = [15]), (Level = 6, data = [19]), (Level = 1, data = [37])
, (Level = 1, data = [54]), (Level = 2, data = [58]), (Level = 1, data = [63])
, (Level = 1, data = [68]), (Level = 5, data = [69]), (Level = 1, data = [71])
, (Level = 4, data = [74]), (Level = 4, data = [81]), (Level = 1, data = [82])
, (Level = 1, data = [99]), (Level = 1, data = [101])
-----------
```

```
Maximum Level = 6, size = 24
, (Level = 1, data = [1]), (Level = 1, data = [3]), (Level = 1, data = [4])
, (Level = 4, data = [7]), (Level = 4, data = [8]), (Level = 1, data = [11])
, (Level = 1, data = [12]), (Level = 6, data = [13]), (Level = 1, data = [14])
, (Level = 1, data = [15]), (Level = 1, data = [16]), (Level = 6, data = [19])
, (Level = 1, data = [37]), (Level = 1, data = [54]), (Level = 2, data = [58])
, (Level = 1, data = [63]), (Level = 1, data = [68]), (Level = 5, data = [69])
, (Level = 1, data = [71]), (Level = 4, data = [74]), (Level = 4, data = [81])
, (Level = 1, data = [82]), (Level = 1, data = [99]), (Level = 1, data = [101])

-----------

skip1.find(99) = 99
skip1.find(11) = 11
skip1.find(4) = 4
(not exist)skip1.find(2) = null
skip1.find(15) = 15
(not exist)skip1.find(24) = null
(not exist)skip1.find(66) = null
```

“make” command compiles, and “make run” command runs the program.