Problem:
The “size” of a map should only include the number of leafs in the map. The size in the example includes every internal node as part of the size. The proper way is to count only the leaf nodes. Given the size you have calculated, subtract 1 and divide by 2 to get the actual size.

Client.java

```
import java.util.Random; // for generating random numbers

/**
 * @author joey
 */
public class Client {
    public static final int N = 8800;

    public static void main(String[] args) {
        // reallly wishing for typedef stuff right here:
        TreeMap<Integer, Integer> unbalancedSearchTree  = new TreeMap<>();
        AVLTreeMap<Integer, Integer> balancedSearchTree = new AVLTreeMap<>();

        testBalancedTree(balancedSearchTree);
        System.out.println("");
        testUnbalancedTree(unbalancedSearchTree);
    }

    /**
     * @param tm TreeMap to find height of
     * @return height of given tree (assume unbalanced)
     * wrapper over recursive height function
     */
    public static int getHeightOfTree(TreeMap<Integer, Integer> tm) {

        Position<Entry<Integer,Integer>> proot = tm.root();
        int height = findHeight(proot, tm, 0);

        return height;

    }

    /**
     * @param p position to calculate height from
     * @param tm TreeMap that p belongs to
     * @param current_height current tracked height
     * @return height of tree from this p
     * recursively finds height of (assumed) unbalanced tree
     * balanced tree requires additional calculation afterwards
     * see testBalancedTree() for example
     */
    public static int findHeight(
            Position<Entry<Integer,Integer>> p,
```

```
        TreeMap<Integer, Integer> tm,
        int current_height) {

    // track every left child first
    if(tm.left(p) != null) {
        int tmp_height = findHeight(tm.left(p), tm, current_height+1);

        if(tmp_height > current_height)
            current_height = tmp_height;
    }

    if(tm.right(p) != null) {
        int tmp_height = findHeight(tm.right(p), tm, current_height+1);

        // only care if it is greater
        if(tmp_height > current_height)
            current_height = tmp_height;
    }

    return current_height;
}

/**
 * @param ubst Unbalanced Search Tree
 */
public static void testUnbalancedTree(TreeMap<Integer, Integer> ubst) {
    Random r = new Random();

    System.out.println("== Unbalanced Tree ==");
    System.out.println("Before any insertions: ");
    System.out.println("Tree map size:   " + ubst.size());
    System.out.println("Tree map height: " + getHeightOfTree(ubst));

    for(int i = 0; i < 8800; i++)
        ubst.put(i, r.nextInt()); // unique key each time

    System.out.println("After " + N + " insertions: ");
    System.out.println("Tree map size:   " + ubst.size());
    System.out.println("Tree map height: " + getHeightOfTree(ubst));
}

/**
 * @param bst Balanced Search Tree
 */
public static void testBalancedTree(AVLTreeMap<Integer, Integer> bst) {
    Random r = new Random();

    System.out.println("== Balanced Tree ==");
    System.out.println("Before any insertions: ");
```

```
        System.out.println("Tree map size:   " + bst.size());

        // height of balanced search tree is log base-2 of height of purely unbalanced tree
        // which is what recursive findHeight function returns
        // log base-n of x can be calcuated as (ln(x)/ln(n)), ln = natural logarithm
        int treeHeight = getHeightOfTree(bst);
        if(treeHeight != 0) // ln doesnt play nice with input of zero
            treeHeight = (int)(Math.log((double)treeHeight) / Math.log(2.0));
                        // ^^ because Java doesnt have log base-n function
        System.out.println("Tree map height: " + treeHeight);

        // key,value insertion routine
        for(int i = 0; i < 8800; i++)
            bst.put(i, r.nextInt()); // unique key each time

        System.out.println("After " + N + " insertions: ");
        System.out.println("Tree map size:   " + bst.size());
        treeHeight = (int)(Math.log((double)getHeightOfTree(bst)) / Math.log(2.0));
        System.out.println("Tree map height: " + treeHeight);

    }

}
```

Screenshot of output: