CS 204, Program #3: Binary Tree Performance
Final version

Due Wednesday, Sept. 30th by 11:59 p.m.

Summary: Implement the Binary Search Tree from the text. Modify their algorithm so that it doesn't enter duplicate values. Add a method that tells how many things are currently in the tree. Evaluate the performance of this tree under 3 different situations from totally random to pretty well structured. Compare the performance of such trees.

The Problem:

1) We enter 500 randomly chosen numbers in the range of 1..1000. Then we search for each of the numbers in the range 1..1000, half of those searches being successful, and half of them being unsuccessful. Count the total number of "nodes inspected" that you need to follow to execute all of the searches. (That's every time you have to look at "current.contents" or "current.getContents().) Print out the *average* number of such steps it took to do a search.

2) Enter 500 randomly chosen numbers, but do them in the following order: 50 randomly chosen numbers between 1 & 100; 50 randomly chosen numbers between 101 & 200; etc. Do the same experiment as above.

3) Enter 500 randomly chosen numbers, but do them in the following order: 5 randomly chosen numbers between 1 & 10; 5 randomly chosen numbers between 11 & 20; etc. Do the same experiment as above.

Recall that if we entered *all* of the 500 numbers in sequential order from smallest to largest, we wouldn't get a tree at all, but would rather get a linked list. In this linked list, the worst-case number of node-accessing steps can be calculated directly, and would be 375.25 steps; the average-case would be 250.25 steps. If we had a complete tree, the value can also be calculated directly, and is 8.48 steps. These should be viewed as the upper and lower bounds, against which the data you collect should be compared.

Programming Requirements:

1) You must use the "TreeNode" class posted on Moodle, with no changes. You must use the "MyRandom" class posted on Moodle. Your program should work correctly with the provided Driver public methods, although you are welcome to test your code with variations of that code, and you need not use the same private methods that I suggest in the driver template.
2) Except for the edits implied by the description above, you must use (variations of) the textbook code for the methods listed above. (It's easy to find dozens of versions of basic linked lists code online, but those aren't appropriate for this assignment.)
3) Your BinaryTree class must use the following private object variable:

```
private TreeNode root;
private int numberOfNodesChecked = 0;
private int sizeOfTree = 0;
```

The "numberOfNodesChecked" keeps track of how many times some node was inspected when doing the searches, and the "sizeOfTree" keeps track of how many nodes are currently in the tree.

4) You need setters and getters for the two integer variables of requirement #3. You shouldn't have such methods for "root". You need a setter for the random number generator in the Driver class, which should be named "setRandomNum".

5) Your BinaryTree class must have the methods:

```
/** Search to see if any node contains the specified key.
  *  If so, return a pointer to that node.
  */
public TreeNode search( long key )

/** Given a node with a defined value in its contents, insert it into the Binary
  * Search Tree in the appropriate location. Modify so it doesn't insert duplicate keys.
  */
public void insert(TreeNode node)
```

6) See the posted template for a driver for some suggestions as to how to construct your design program. Your code must work with the "main" method shown there. In order for that code to work, the other public methods must also work as described. You need not use the same "private" methods as I did, although I think it's a good suggestion.

7) The numbers generated by your random number class must all be inserted into the tree in the order in which they're generated (that's the only way to get the results the JUnit tests expect).

8) Your code must pass the JUnit tests provided. These should be posted to Moodle on Sunday.

9) Your output must be in the format shown in the sample runs below.

10) Your code must meet my standard "clean code" specifications: Good variable names, good formatting, full method comments for all methods, and no methods longer than 9 lines of code. If a method has more than 6 lines, it should be broken into 2-3 blocks, where blocks are separated by blank lines, and including a "block comment" for each block.

11) At the beginning of your code, you must include an author specification, which should include your full name (or name preference), in the following format:

```
/** Binary search tree performance evaluation experiment
  * @author John Jacob Jingleheimer Schmidt
  */
```

Sample Run #1:

```
The number of nodes inspected, on average, was 11.48.
The number of nodes inspected, on average, was 25.02.
The number of nodes inspected, on average, was 121.90.
```

Sample Run #2, replacing the random seeds with 101, 1001, and 10001:

```
The number of nodes inspected, on average, was 11.39.
The number of nodes inspected, on average, was 25.75.
The number of nodes inspected, on average, was 118.42.
```

Sample Run #3, replacing all random seeds with zero:

```
The number of nodes inspected, on average, was 11.41.
The number of nodes inspected, on average, was 25.32.
The number of nodes inspected, on average, was 119.26.
```