

## CMSC 132 Quiz 5 Worksheet

The fifth quiz for the course will be on Thursday, Nov 3. The following list provides more information about the quiz:

- The quiz will be a written quiz (no computer).
- Closed book, closed notes quiz.
- Answers must be neat and legible. **You must use pencil.**
- Check the information available at <http://www.cs.umd.edu/~nelson/classes/examRules.html>

The following exercises cover the material to be included in this quiz. Solutions to these exercises will not be provided, but you are welcome to discuss your solutions with the TA or instructor during office hours. **We strongly recommend you do not use Eclipse to write the code associated with these exercises.** Try to answer the exercises in a piece of paper and then use Eclipse to verify your solutions. This approach will better prepare you for the quiz.

Some recursion problems require an auxiliary method. For example, a recursive implementation for the tree size() method may use an auxiliary method that takes as parameter a reference to a Node. Keep this in mind while solving the problems below.

The following Java class definition for a binary search tree will be used to answer the questions that follow. Unlike project #3, the following tree is not polymorphic and we use null to represent an empty tree. For example, an empty BinarySearchTree has a null root, and a leaf node has null left and right fields. You may not add any variables (instance or static) to the class in order to answer the questions below.

```
public class BinarySearchTree <K extends Comparable<K>, V> {
    private class Node {
        private K key;
        private V data;
        private Node left, right;
        public Node(K key, V data) {
            this.key = key;
            this.data = data;
        }
    }
    private Node root;
}
```

1. Define a constructor that creates an empty tree.
2. Define a recursive method **add(K key, V data)** that adds a key,value pair to the proper location in the tree.
3. Define a recursive method **size()** that returns the number of entries in the tree.
4. Define a **non-recursive** method **max()** that returns the data associated with the maximum key value in the tree.
5. Define a **recursive** method **max()** that returns the data associated with maximum key value in the tree.
6. Define a recursive method **min()** that returns the data associated with minimum key value in the tree.
7. Define a recursive method named **postOrderTraversal()** which returns a string representing a post-order traversal of the tree.
8. Define a recursive method **getNumInteriorNodes()** that returns the number of non-leaf nodes in the tree.
9. Define a recursive method **getHeight()** that returns the height of the tree.
10. Define a recursive method **leaves(ArrayList<K> L)** that adds to the ArrayList L the key value of leaf nodes.
11. Define a recursive method **isDegenerate()** that returns true if a binary search tree is a degenerate tree and false otherwise. A degenerate tree is one where every node has only one child.
12. Define a recursive method **getDecreasingOrderList()** that returns an ArrayList with the data elements of the tree inserted into the list based on decreasing key order.