

CMSC 132 Quiz 3 Worksheet

The third quiz for the course will be on Wednesday, April 2, during your lab session. 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. We recommend that you use pencil and eraser.

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 `size()` method of a linked list calls for an auxiliary method that takes as parameter a reference to a node. Keep this in mind when solving the problems below.

A. Linked Lists

The following Java class definition for a linked list will be used to answer the questions that follow. You may not add any variables (instance or static) to the class in order to implement the questions below.

```
public class LinkedList<E> {
    private class Node {
        private E data;
        private Node next;
        private Node (E data) {
            this.data = data;
            next = null;
        }
    }
    private Node head;
}
```

1. Define a recursive method ***int size()*** that computes the size of a list.
2. Define a recursive method ***void printReverse()*** that prints the list (using `System.out.println`) in reverse order.
3. Define a recursive method ***boolean find(E elem)*** that returns true if ***elem*** is in the list and false otherwise.
4. Define a recursive method ***boolean equals(LinkedList<E> list)*** that determines whether two lists have the same corresponding elements. For example, the lists 10, 20, 30 and 10, 20, 30 are considered equals, whereas 10, 20, 30 and 30, 20, 10 are different.

B. Binary Trees

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. Null is used to indicate an empty tree. For example, an empty BinarySearchTree has a null root, and a leaf node has null left and right fields. For this problem you may not add any variables (instance or static) to the class in order to answer the questions below.

```
public class BinarySearchTree <E extends Comparable<E>> {
    private class Node {
        private E data;
        private Node left, right;
    }
    private Node root;
}
```

1. Define a constructor that creates an empty tree.
2. Define a method **void add(E val)** that adds a node to the proper location in the tree.
3. Define a recursive method **int numInteriorNodes()** that returns the number of non-leaf nodes in the tree.
4. Define a recursive method **int height()** that returns the height of the tree.
5. Define a recursive method **void leaves(ArrayList<E> L)** that adds to the ArrayList L the data component of leaf nodes.