COP 3530
Data Structures

Midsemester Exam

Name: ________________________________

October 13, 2008

This exam has 3 questions. Each question starts on a new page. Please answer each question on its page. You may assume `java.util` has been imported. There will be no deductions for lack of commenting. There will be no deductions for lack of import directives. There will be no deductions for minor syntax errors.

1. [**50 points**] Consider the following method, whose implementation is shown:

```
// Returns the sum of the item in list c
public static int sum( List<Integer> c )
{
    int total = 0;

    for( int i = 0; i < c.size( ); i++ )
        total += c.get( i );

    return total;
}
```

Assume that the lists have N items each.

(a) What is the running time of `sum`, as written above if both lists are `ArrayLists`?

(b) What is the running time of `sum`, as written above if both lists are `LinkedLists`?

(c) Suppose it takes 2 seconds to run `sum` on a 50,000 item `LinkedList`. How long will it take to run `sum` on a 100,000 item `LinkedList`?

(d) Explain how to modify the `sum` algorithm to run efficiently regardless of whether an `ArrayList` or `LinkedList` is passed as a parameter.

2. [**50 points**] Assume that you have a `java.util.Map` in which the keys are `String`s and the values are `List<Integer>`s.

   Write a routine, `emptyKeys`, that returns a `List` of `String`s that are keys whose corresponding values are empty lists.

   ```
   { hello=[2,3], good=[], this=[], zoo=[10,20,40] }
   ```

   then the `List` returned by `emptyKeys` is

   ```
   ["good","this"]
   ```

   (a) Write this routine below, using Java 5.

   (b) Assuming that the `Map` is a `TreeMap`, provide the Big-Oh running time of your routine.

3. **[50 pts]**

Routine `merge` takes a subarray `arr`, from indexes `low` to `high` inclusive, with the first half of the subarray already sorted, and the second half of the subarray already sorted (and merges the halves). Routine `merge` has signature:

```
    public static void merge( String [ ] arr, Comparator<String> cmp, int low, int high )
```

You may assume that `merge` has been implemented for you. Recall that the mergesort algorithm works by recursively sorting two halves, and merging the result.

Implement `mergesort`. In implementing `mergesort`, you should implement a private recursive routine, and have your public driver invoke it. Implement the public driver started below, and write a private recursive routine.

```
public static void mergesort( String [ ] arr, Comparator<String> cmp )
{
```