COP 3530
Data Structures

Midsemester Exam Version A

Name: ______________________________

October 19, 2004

This exam has 4 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**] Static method `toString` returns the `String` representation of an array. The representation is surrounded by brackets, and includes each array entry separated by a single space. `toString` is shown below:

```
public static String toString( Object [ ] arr )
{
    String result = "[";

    for( int i = 0; i < arr.length; i++ )
        result += " " + arr[ i ];
    result += " ]";

    return result;
}
```

   (a) This algorithm is not efficient. Explain why and provide the Big-Oh running time of `toString`.

   (b) If `toString` takes 8 milliseconds for an array of 1000 items, approximately how long would it take for an array of 3000 items?

   (c) Rewrite `toString` so that it is efficient.

2. [**50 points**] This question requires that you implement some methods for a class that represents a doubly-linked list. In this question, **both a beginMarker and endMarker are used**; however, there is no field used to keep track of the size. You may assume an appropriate declared nested class `Node`. You may assume that the list does not store `null` values. You should only be following links; your solutions shuld not create or use any iterator classes.

   (a) Implement `contains` and **PROVIDE ITS BIG-OH RUNNING TIME**.

   ```
   public boolean contains( Object x )
   {









   }
   ```

   (b) Implement the private helper method `remove` in the space shown below:

   ```
   private void remove( Node p )
   {









   }
   ```

   (c) Implement `removeLast` in the space shown below. You may assume code written by you in the previous part works. You must throw an exception if appropriate.

   ```
   public void removeLast( )
   {
   ```

3. [**50 points**] Assume that you have a `java.util.Map` in which the keys are `String`s and the values are `String`s.

(a) Write a routine, `removeSomeEntries` that removes from the map all entries that have the same key and value. For instance, if the map contains the six key/value pairs shown here:

```
{ hello=world, good=bad, this=this, if=who, bad=nice, nice=nice }
```

then after the call to `removeSomeEntries`, the map contains

```
{ hello=world, good=bad, if=who, bad=nice }
```

```
public static void removeSomeEntries( Map m )
{
```

4. [**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( Object [ ] arr, Comparator 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( Object [ ] arr, Comparator cmp )
{
```