COP 3530
Data Structures

Midsemester Exam Version A

Name: _____

June 28, 2005

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 `sum` returns the sum of the items in the even positions of a `List`. One possible implementation of `sum` is shown below:

```
public static int sum( List<Integer> list )
{
    int theSum = 0;

    for( int i = 0; i < list.size( ); i += 2 )
        theSum += list.get( i );

    return theSum;
}
```

   (a) Provide the Big-Oh running time, with a one-line explanation, if `list` is an `ArrayList`.

   (b) Provide the Big-Oh running time, with a one-line explanation, if `list` is a `LinkedList`.

   (c) If `sum` takes 10 milliseconds for a `LinkedList` of 1000 items, approximately how large a `LinkedList` could be passed as a parameter, and still have the routine run in under 1 second (one second is 1000 milliseconds)?

   (d) Rewrite `sum`, using an iterator, so that it is efficient for all `Lists`.

2. [**50 points**] This question requires that you implement some methods for a class that represents a doubly-linked list. In this question, **neither a beginMarker nor an endMarker are used**; assume the first node is accessed via `first` and the last node is accessed via `last`. If the list is empty, both `first` and `last` **MUST** be `null`. 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 `toString` and **PROVIDE ITS BIG-OH RUNNING TIME**. If you invoke other methods of this class, you must implement them.

```
public String toString( )
{









}
```

   (b) Implement the public method `removeFirst` in the space shown below. You must provide extra code to handle the special cases where the list is empty. If you invoke other methods, you must write those methods too.

```
public void removeFirst( )
{









}
```

3. [**50 points**] Use a `TreeMap` to implement an algorithm to sort an array of `String`. The idea is that the `TreeMap` will contain the `String`s in the array, and a count of the number of times each `String` occurs. For instance, if the array contains

```
[ hello, world, world, good, hello, world, if ]
```

then the populated `TreeMap` is

```
{ good=1, hello=2, if=1, world=3 }
```

Your algorithm consists of constructing the `TreeMap`, populating it, and then dumping its contents back into the array.

Write this routine below, using Java 1.5.

```
public void sort( String [ ] arr )
```

4. [**50 points**]

Function `findMaxAndMin`, defined below is intended to return (in an array of length 2) the maximum and minimum item (if `arr.length` is 1, the maximum and minimum are the same):

```
// Precondition: arr.length >=1
// Postcondition: the 0th item in the return value is the maximum
//                the 1st item in the return value is the minimum
public static double [ ] findMaxAndMin( double [ ] arr )
{




}
```

Write an appropriate `private static` recursive helper routine below, and fill in the body of the `public static` nonrecursive `findMaxAndMin` Your recursive routine must split a problem into roughly two halves, but should never split into two odd-sized problems (in other words, a problem of size 10 is to be split into 4 and 6, rather than 5 and 5). To simplify your code, you may assume the existence of the following static methods:

```
public static boolean isOdd( int n );
public static double  max( double a, double b );
public static double  min( double a, double b );
```