COP 3530
Data Structures

Midsemester Exam

Name: _____

June 18, 2008

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]** Consider the following method, whose implementation is shown:

```
// Returns true if two numbers in c when added together sum
// to a third number in c.
public static boolean hasSpecial( List<Integer> c )
{
    for( int i = 0; i < c.size( ); i++ )
        for( int j = i + 1; j < c.size( ); j++ )
            for( int k = 0; k < c.size( ); k++ )
                if( c.get( i ) + c.get( j ) == c.get( k ) )
                    return true;

    return false;
}
```

Assume that the lists have N items each.

(a) What is the running time of `hasSpecial`, as written above if the list is an `ArrayList`?

(b) What is the running time of `hasSpecial`, as written above if the list is a `LinkedList`s?

(c) Suppose it takes 2 seconds to run `hasSpecial` on a 1,000 item `ArrayList`. How long will it take to run `hasSpecial` on a 3,000 item `ArrayList`?

(d) Suppose as a first step, the items in `c` are copied into an array and sorted by using mergesort. Provide the remaining steps that will allow you to implement `hasSpecial` faster than the original algorithm above. You do not have to write any code; just describe the key steps.

2. **[50 points]** This question requires that you implement some methods for a class that represents a doubly-linked list. In this question, a header node is used, but there is no tail node. You may assume an appropriate declared nested class `Node`. You may assume that the list does not store `null` values. You may assume that the header node in the list is accessed by `header` and the last node is accessed by `last`, and if the list is empty, then both `header.next` and `last` are null. You should only be following links; your solutions should not create or use any iterator classes. If you use another method that would logically be in the doubly-linked list class, such as a general purpose `add`, then you must write that method too.

   (a) Below you will implement `toString`, `removeFirst`, and `addLast`. Before writing the code, give the Big-Oh running time for each routine.

   (b) Implement `toString`. You may not invoke other methods of this class.

```
public String toString( )
{




}
```

   (c) Implement `removeFirst` below. You may not invoke any other methods of the class. Make sure to appropriately handle the cases of a zero-element and one-element list.

```
public boolean removeFirst( )
{




}
```

   (d) Implement `addLast`. You may not invoke any other methods of the class. Make sure you have handled the special case of an empty list.

```
public void addLast( AnyType x )
{



}
```

   DID YOU REMEMBER TO GIVE THE BIG-OH?

3. **[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, `sumsToTen`, that returns a `List` of `String`s that are keys whose corresponding values sum to exactly 10. For instance, if the map contains the four key/value pairs shown here:

   ```
   { hello=[2,3], good=[1,2,3,4], this=[10], zoo=[10,20,40] }
   ```

   then the `List` returned by `sumsToTen` 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.

4. **[50 points]** Suppose that the world is flat, and can be represented as a grid of Boolean variables. Each Boolean variable is true if it represents land, and false if it is water. The grid can be encapsulated in the following interface:

```
interface GridElement
{
    int            getRow( );
    int            getCol( );
    boolean        containsWater( );

    List<GridElement> getAdjacents( );   // returns grid elements that are adjacent
                                         // size is < 8 if at edge of world
}
```

Implement method `getWaterArea`:

```
public static int getWaterArea( GridElement pos )
```

`getWaterArea` accepts a `GridElement` as a parameter. If the `GridElement` represents land, `getWaterArea` returns 0. Otherwise it returns the number of `GridElements` that are connected to the `GridElement` parameter via water, including the parameter itself.

For example, in the following grid, in which `W` is water and `L` is land:

```
WWLWLWWLWL
LLLWLWWLLL
LLWLLWLLLL
WLLLLLLWLL
LLLWLLWLWL
```

the results of `getWaterArea` for a few positions are as follows:

```
GridElement pos     getWaterArea( pos )
===============     ===================
(0,0)                2
(0,3)                3
(0,5)                5
```

In implementing `getWaterArea`, you may create private helpers as you see fit, but you must implement them. You MAY NOT declare any additional static data members.

IMPLEMENT `getWaterArea` *ON THE FACING PAGE.*