COP 3530
Data Structures

Midsemester Exam Version D

Name: _____
Email: _____

October 14, 2003

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**] Method `containsTriplicates` returns true if the array contains three items with the same state, and false otherwise:

```
public static boolean containsTriplicates( Object [ ] arr )
{
    for( int i = 0; i < arr.length; i++ )
        for( int j = i + 1; j < arr.length; j++ )
            if( arr[ i ].equals( arr[ j ] ) )
                for( int k = j + 1; k < arr.length; k++ )
                    if( arr[ j ].equals( arr[ k ] ) )
                        return true;

    return false;
}
```

   (a) What is the Big-Oh running time of `containsTriplicates`?

   (b) If it takes 8000 milliseconds to return false for 1000 items, approximately how long would it take to return false for 100 items?

   (c) Using the Collections API, describe an algorithm (in English, no code) that is more efficient than the one above, and provide the running time of your algorithm, with a brief justification.

2. [**50 points**] This question requires that you implement some methods for a class that represents a doubly-linked list. In this question, **an endMarker is used, but a beginMarker is NOT used**, so you may assume fields named `begin` and `endMarker`. You may assume an appropriate declared nested class `Node`.

   (a) Implement the constructor.

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

```
private void addAfter( Object x, Node p )
{









}
```

   (c) Implement both `addFirst` and `addLast` in the space shown below; be careful to handle any special cases. You may assume code written by you in the previous part works.

```
public void addFirst( Object x )
{




}

public void addLast( Object x )
{




}
```

3. [**50 points**] Assume that you have a `java.util.Map` in which the keys are names of cities (stored as a `String`), and for each city, the value is a `java.util.List` of state (each state is a `String`). Some cities are in more than one state (e.g. Hollywood is in both Florida and California). Write a routine that computes the inverse map, in which the keys are the names of the states, and the values are lists of corresponding cities. The signature of your method is:

```
// In input Map
//     keys are String representing cities
//     values are List of String representing corresponding states
// In returned Map
//     keys are String representing state name
//     values are List of String representing corresponding cities
public static Map inverseMap( Map m )
{
```

4. [**50 pts**] Method `getReverse` returns a `java.util.List` with the original elements reversed. For instance, if the original list is `[3, 4, 8]`, then the returned list is `[8, 4, 3]`.

   Implement `getReverse` **recursively**. You may write an additional private routine if you find that helpful. Your algorithm must be efficient in terms of Big-Oh, regardless of whether the `List` is an `ArrayList` or `LinkedList`.