COP 3530
Data Structures

Midsemester Exam

Name: _____

June 25, 2007

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 the number of elements in both c1 and c2
// Assumes no duplicates in either list.
public static int intersect( List<Integer> c1, List<Integer> c2 )
{
    int count = 0;

    for( int i = 0; i < c1.size( ); i++ )
    {
        int item1 = c1.get( i );
        for( int j = 0; j < c2.size( ); j++ )
        {
            if( c2.get( j ) == item1 )
            {
                count++;
                break;
            }
        }
    }

    return count;
}
```

Assume that the lists have N items each.

(a) What is the running time of `intersect`, as written above if both lists are `ArrayList`s?

(b) What is the running time of `intersect`, as written above if both lists are `LinkedList`s?

(c) Suppose it takes 4 seconds to run `intersect` on two 1,000 item `ArrayList`s. How long will it take to run `intersect` on two 3,000 item `ArrayList`s?

(d) Does rewriting the two loops using the enhanced for loop (i.e. `for( int x : c1 )`) make `intersect` more efficient? Provide an explanation of your reasoning.

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.

   (a) Below you will implement `toString`, `removeLast`, and `addFirst`. 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 `removeLast` 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 removeLast( )
{









}
```

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

```
public void addFirst( 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. The map represents words and the line numbers on which they occur.

   Write a routine, `linesToWords`, that returns a `Map` in which the keys are line numbers, and the values are lists of `String`s representing the words on the corresponding line numbers.

   For instance, if the map contains the four key/value pairs shown here:

   ```
   { hello=[2,3], good=[1,2], this=[1,5], if=[1,2,3] }
   ```

   then the map returned by `linesToWords` is

   ```
   { 1=["good","this","if"], 2=["hello","good","if"], 3=["hello","if"],5=["this"] }
   ```

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

   (b) Assuming that both maps are `TreeMap`s, provide the Big-Oh running time of your routine.

4. [**50 points**] Write routine `getAllWords` that takes as parameter a word and returns a `Set` containing all the substrings of the word. **THE SUBSTRINGS DO NOT NEED TO BE REAL WORDS, NOR CONTIGUOUS, BUT THE LETTERS IN THE SUBSTRINGS MUST RETAIN THE SAME ORDER AS IN THE WORD**. For instance, if the word is `cabb`, words that would be in the `Set` produced by `getAllWords` would be `"", "b", "bb", "a", "ab", "abb", "c", "cb", "cbb", "ca", "cab", "cabb"` .

   Implement `getAllWords`, starting below. You may add a `private` routine that is recursive and have the public routine invoke it.

```
// Returns all substrings (NOT NECESSARILY CONTIGUOUS) of word
public static Set<String> getAllWords( String word )
{
```