Data Structures and Algorithms

Assignment #1

Programming part Due: Wednesday April 5, 11:00 am
Paper part Due: Wednesday April 5 at beginning of class

The programming part of this assignment is meant to (re)familiarize yourself with the Java computing environment. What you are asked to do has been covered in CSE 142 and CSE 143. As a refresher, look in your book Chapter 3, Sections 1 and 2. Alternatively, you can go back and look at Chapters 6 and 7 of Reges and Stepp for file processing and arrays
http://www.cs.washington.edu/education/courses/142/06wi/chapters/ch6.html
http://www.cs.washington.edu/education/courses/142/06wi/chapters/ch7.html
and at the numerous hand-outs in the home page of CSE 143 that deal with singly-linked lists, for example:
http://www.cs.washington.edu/education/courses/143/06wi/handouts/07.html
and
http://www.cs.washington.edu/education/courses/143/06wi/handouts/08.html

The "paper" portion of the assignment deals mostly with the big-Oh notation (cf. Chapter 4 Sections 1 and 2 of your book).

Programming part

You are to write a Java program that reads a text file of unsorted n integers and constructs an array and a linked list of these integers sorted in ascending order. After each integer is read, the corresponding array/linked list should be sorted.

You can assume that $n < 1000$. You cannot assume that all integers are distinct but the number of duplicates will be small, i.e., there will be $O(logn)$ of them. **Duplicates should not be inserted**.

The two methods that you should use are:

1. Using an array. Insert the integer at the end of the current array and perform an insertion sort (cf. Section 3.1.2. of your book) but take advantage of the fact that the current array is already sorted.
2. Using a linked list. Traverse the list to find the position where the new node should be inserted and insert it.

The output should be a listing of the sorted integers, 10 integers per line.

A template class that opens a file, defines a list object, and has some method headers is attached. A small test file is also attached, but it does not cover all the possible test cases.

Paper Questions

1. In terms of big-Oh, what is the complexity of the array method? Explain why.
2. In terms of big-Oh, what is the complexity of the linked-list method? Explain why.

3. A friend of yours say that you can write a shorter program in the array case by using the "sort" method found in the java utilities (cf. Reges Chapter 7.7). The pseudo-code would be something like:

```
for (i = 0; i < n-1 ; i++){
    A[i+1] := new value;
    Arrays.sort(A);
}
```

   Is this correct? We are not talking about syntax, of course, but would the algorithm give the right answer? Justify your answer.

   Even if it were correct, why would this be an inefficient way to perform the task (Hint: the "sort" method uses the algorithm Quicksort that runs in average in $O(nlogn)$ time but in the worst case, when the array is sorted or almost sorted in $O(n^2)$).

4. Problem R-4.12 in your book

test1.txt:
```
23 83 12 84 39
73 44 55 83
11 48
32 58 33
12 23 12 32
56
```

output:
```
11 12 23 32 33 39 44 48 55 56
58 73 83 84
```

Assignment1.java:

```
import java.io.*;
import java.net.*;

public class Assignment1 {
    private static final String FILENAME = "test1.txt";

    private static class ListNode {
        int data;
        ListNode next;
    }

    /* File paths can be specified in two ways, absolutely or relatively.
       Absolutely is with respect to the root, so something like
       C:/Doc.../Username/Desktop or D:/MyFiles/CSE373/.  It will
       obviously be different per user (and grader).
       The alternative is relative -- relative to the current directory,
       i.e. where java is run from.  Relative works well, except in the
       case of Dr Java, where relative is to Dr Java, not your program
       (because Dr Java is itself java and spawns your code from it).
       The solution is the code below that asks the current class to
       open a file from its own directory. */
    private static Scanner scannerFromFilename(String filename) {
        URL url = Assignment1.class.getClassLoader().getResource(filename);
        if (url == null)
            throw new RuntimeException("File not found: " + filename);
        try {
            Scanner s = new Scanner(url.openStream());
            return s;
        } catch (Exception e) {
            throw new RuntimeException(e + filename +
                  " couldn't be openStreamed");
        }
    }


    // returns a reference to an array containing the parsed data
    // in sorted order and duplicates removed
    private static int[] arrayReadAndSort(Scanner s) {
        throw new UnsupportedOperationException();
    }

    // returns a reference to the front of a linked list
    // containing the parsed data in sorted order and dups removed
    private static ListNode linkedListReadAndSort(Scanner s) {
        throw new UnsupportedOperationException();
    }
    // prints the array, 10 elements per line
    private static void arrayPrint(int[] array) {
        throw new UnsupportedOperationException();
    }

    // prints the linked list starting at front, 10 elements per line
    private static void linkedListPrint(ListNode front) {
        throw new UnsupportedOperationException();
    }

    public static void main(String[] args) {
        Scanner s;
        s = scannerFromFilename(FILENAME);
        arrayPrint(arrayReadAndSort(s));
        s = scannerFromFilename(FILENAME);
        linkedListPrint(linkedListReadAndSort(s));
    }

}
```