Homework 1

Data Structures and Algorithms in JAVA
Shlomo Herschkop
Department of Computer Science
Columbia University
Summer 2002

Due: July 18 2002, beginning of class.

Theory (45 points)

1. (6 points) The “card swipe”, is a machine sitting at every entrance of each
building on Columbia’s campus, which you need a valid Columbia card to swipe
through to gain access to some area or building. Consider the design of a program,
which runs the card swipe access authorization (CSAA) from a central location.
Assume that Columbia has about 40,000 students, and 200 machine sites. Name
three primitive operations that must be implemented on the CSAA, and give a
reasonable time constraint for each operation.

2. (5 points) Here is the recursive factorial function from the class/text:

```java
public static int factorial (int n)
{
    if (n <= 1)
        return 1;
    else
        return n * factorial (n-1);
}
```

Re-write this function so that it does not use recursion.

3. (6 points) Write a recursive method that returns the number of 1s in the binary
representation of N.
example: 23 = 10111 would return 4
      28 = 11100 would return 3

Hint: Use the fact that this is equal to the number of 1s in the representation of N/2,
plus 1 if N is odd.

4. (6 points) Write and explain how a series of JAVA statements would use the
LinkedList interface to delete all even elements in a given list.

5. (6 points) Prove, using mathematical induction, that the sum for i = 1 to n of i^2 is
(2n^3 + 3n^2 + n) / 6.

6. (4 points) Prove, using mathematical induction, that the sum for i = 1 to n of 1/(2^i)
is 1 - (1/(2^n)).
7. (3 points) Arrange the following expressions by growth rate from slowest to fastest: $14n^2$, $\log_3 n$, $3^n$, $20n$, $2$, $\log_2 n$, $n^{2/3}$, $n \log_2 n$, $n!$.

8. (2 points) Suppose that a particular algorithm has time complexity $O(n^2)$, and that executing an implementation of it on a particular machine takes $T$ seconds for $n$ inputs. Now suppose that we are presented with a machine that is 64 times as fast. How many inputs could we process on the new machine in $T$ seconds?

9. (8 points) Give an analysis of the Big-Oh running time for each of the following program fragments:

```java
sum = 0;
for (i = 0; i < 3; i++)
    for (j = 0; j < n; j++)
        sum++;

sum = 0;
for (i = 0; i < n*n; i++)
    sum++;
```

Assume the array contains $n$ values.

```java
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++)
        array[i] = random(n); // random() takes constant time
    sort(A, n); // sort takes $n \log n$ time
}
sum = 0;
if (EVEN(n)) // EVEN(n) is true iff $n$ is even
    for (i = 0; i < n; i++)
        sum++;
else
    sum = sum + n;
```
Programming (55 points)

In this assignment, you will demonstrate insertion, printing, searching, and manipulating singly linked lists. This involves designing and writing an object, text editing it into a file, compiling it in an environment that imports the needed classes, debugging it, and running it on some test data. When you are ready, submit the text of its code and a copy of your testing. Make sure that the source code is clear and its user interface is comprehensive and informative.

You can do this part of the assignment according to the following bailouts:

1. (15 points). Write a program that takes as input a non-negative integer. Insert this integer into the list, without respect to whether the list is sorted or whether the list already contains the input integer. If the input is zero, take this as a signal to print the list instead and don't insert the zero. Use a separate method for inserting into the list, and a separate method for printing the list; you can use the book's methods if you properly document that you are doing so. Thoroughly demonstrate the program on your choice of inputs, including the case where the first input is zero. For this step, negative integers and non-integer input should give an error. Nearly all of this step can be done by using the code given in the book; all that is necessary is to properly set up your paths and directories, add code to handle user input (which is somewhat tedious and tricky at the same time), and modify the object's main method. The purpose of this step is to get everyone up and running in Java.

2. (10 points). Modify an existing method to check to see if the input integer is already present. If this is the case, you don't have to make the list longer and you don't have to insert the item; this prevents duplicates. Test, by using inputs of zero liberally. This step is designed as an exercise in modifying the method part of an object, and it depends on Step 1.

3. (15 points). Modify the list node so that it includes a count field. Modify the insertion method so that if an input is positive and is recognized to be present, then the method increments the count. For example, if the list includes a 5, and the user asks to insert another 5, then the only change that occurs in the list is that it records that there are now two such 5s. Also, keep track of the total number of integers in the list. Modify the print method so that it prints each integer in the list, followed by its frequency (with two significant figures). For example, the input

```
2 1 3 5 1 4 4 0
```

would result in the following output:

```
2 14.29%
1 28.57%
```
Note that the integers will not necessarily be in sorted order in the printout. This step is designed primarily as an exercise in creating and modifying the variables part of an object, and in updating the methods appropriately. It extends step 2, so it depends on it, which in turn depends on step 1.

4. (15 points). Add a method, which is applied if the input is a negative integer, that reverses the order of the items in the list. Try to do this as efficiently (in time) as possible. The reversal should be carried out only through pointer manipulation; the values stay in their own structure, but their order in the list changes. Test your reversal routine. This step is the only part of the assignment that really exercises your understanding of the data structures, and future assignments will be more oriented to steps such as this one. This step depends only on step 1, so it can be done immediately and independently of steps 2 and 3 (i.e., you can do steps 1 and 4; or steps 1, 2, and 4; or all four steps).

NOTE: You should only submit one complete homework assignment. That is, if you do more than just step 1, you do not have to hand in a separate step 1 with separate testing; you will receive full credit for step 1 as well (if your testing shows that it does in fact work). You should clearly indicate in what you do submit which steps you did actually do, and make sure the entire combined functionality of all the steps you did do is tested.

This program is intended to make you familiar with most of the systems that this course uses, and with much of list structure manipulation. Write your program clearly, with a large block comment at the beginning describing what the program does, how it does it, and how it will be tested. To the extent possible, embed the actual values that will be used for testing in the comment so that future maintainers of the code will know how to test it.

Clear programming style will account for a substantial portion of your grade for the programming part of this assignment. A good rule of thumb here is to imitate the style of the example programs in the text.

Good Luck!