Theory (20 points)

Goal: review recursion, running time, analysis, linked lists.

1) Bill and Mary are arguing about the performance of their sorting algorithms. Bill claims that his $O(n \log n)$-time is always faster than Mary’s $O(n^2)$-time algorithm.

To settle the issue, they implement and run the two algorithms on many randomly generated data sets. To Bill’s dismay, they find that if $n < 100$ the $O(n^2)$-time algorithm actually runs faster, and only when $n \geq 100$ is the $O(n \log n)$-time one better.

Explain why this scenario is possible. You may give numerical examples to illustrate your point.

2) Here is a recursive factorial function:

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

We claim all recursive programs can be written in a non-recursive fashion. Rewrite this function so that it computes the same answers but does not use recursion.
3) Arrange the following expressions by growth rate from slowest to fastest:

a. $14n^2$

b. $\log_3 n$

c. $3n$

d. $20n$

e. 2
f. $\log_2 n$

g. $n^{2/3}$

h. $n \log_2 n$

i. $n!$

4) Give an analysis of the Big-Oh running time for each of the following program fragments:

a. sum = 0;
   for (i = 0; i < 3; i++)
      for (j = 0; j < n; j++)
         sum++;

b. sum = 0;
   for (i = 0; i < n; i++)
      for(k=i; k < n; k++)   //!!!notice k is not 0
         sum++;

c. 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
      if n is even
         for (i = 0; i < n; i++)
            sum++;
   else
      sum = sum + n;
5) When I want to send you a message $M$ over the Internet (think of a movie over a p2p system). I can break $M$ into $n$ data packets, numbers the packets in order, and inject them into the network. When the packets arrive at your computer, they may be out of order, so you must assemble the sequence of $n$ packets in order before you can be sure you have the entire message $M$.
   a. Describe an efficient scheme for you to do this.
   
   b. What is the running time of your algorithm?

6) You just bought yourself a collection of $n$ very expensive bottles of wine. Unfortunately you’ve just found out the reason it was so cheap on eBay was that one of the bottles is poisoned. Even better, you don’t know which bottle is poisoned.
   a. Assume you can hire a bunch of testers who don’t mind ending up in the hospital with poison, can you think of a scheme to use the least amount of tasters? What is the run time?
Programming Section (30 points)

We will be playing with linked lists. This is a handy data structure which grows as you need it. Besides the code you will submit you will need to follow the following guidelines:

1) Include a readme.txt file which states your name, and for each class you create, briefly outlines what it does.
2) Comments are important! Sprinkle them liberally throughout the code and before methods
3) You name should be in a comment at the top of each class.

Step 1:
Create a class called SuperLinkedLists which will implement a basic linked list (you can use Java API or book code for this). That means you need to support Insert, Delete, and Find operations on the list. Assume we will be keeping lists of Integer objects. Feel free to create as many classes as you need (Example LinkedItem class). In addition create an exception class called SLException (super linked exception) for use in your code.

Step 2:
Adopt the delete method to throw an exception if the item is not in the list. Implement a print method to print out the list in a nice fashion.

Step 3:
Now adopt the insert method and the class code so that a counter is kept with each item in the list. When insert happens, and the item is already present you should increment the counter instead of inserting it again in the list. So inserting 2,1,3,4,3 will only have a list of 2,1,3,4 but 3’s count will be 2. Adopt print with the new information.

Step 4:
Implement a reverse method which flips the order of the list. Again feel free to adopt the class code for this.

Step 5:
Create a main in the HWTTest.java class. Write code to insert 100 random numbers (between 0-23) as strings. Adopt a print method so that you can print the list, and output each item along with a frequency count:

2 14.29%
1 28.57%
3 14.29%

Step 6:
In your readme, analyze the running times for each of the methods you have implemented in your class.