CS W3137, Assignment 1, DUE: 2/6/14 in class

1 Non-Programming Problems, 5 points each

1. Exercise 2.6 (Exercises refer to the Weiss Textbook Exercises)
2. Exercise 2.7 (part a only)
3. Exercise 2.11
4. Exercise 2.14 (part a only)
5. Exercise 2.15
6. Exercise 2.22
7. Exercise 2.27
8. Prove by induction:
$$\sum_{i=0}^{n} i^3 = \frac{n^2(n+1)^2}{4}$$
9. Consider the function $f(n) = 3n^2 - n + 4$. Show that $f(n) = O(N^2)$

Extra Credit: (4 points)

An evil king has a cellar containing N bottles of expensive wine, and his guards have just caught a spy trying to poison the king's wine. Fortunately, the guards caught the spy after he succeeded in poisoning only one bottle of wine. Unfortunately, they don't know which bottle! To make matters worse, the poison the spy used was very deadly; just one drop diluted even a billion to one will still kill someone. Even so, the poison works slowly; it takes a full month for the person to die. The king could assign N of his prisoners to each drink from a different bottle and see who dies, but he has many, many, many more bottles of wine than prisoners. Design a scheme that allows the evil king to determine exactly which one of his wine bottles was poisoned in just one month's time while risking the life of at most $O(log\, N)$ of his prisoner taste testers.

2 Programming Problems

1. (20 points) Write a Java program that uses a **recursive** method to determine the number of **distinct** ways in which a given amount of money in cents can be changed into quarters, dimes or nickels. For example:

```
java MakeChange 45

Change for 45 = 25 10 10
Change for 45 = 25 10 5 5
Change for 45 = 25 5 5 5 5
Change for 45 = 10 10 10 10 5
Change for 45 = 10 10 10 5 5 5
Change for 45 = 10 10 5 5 5 5 5
Change for 45 = 10 5 5 5 5 5 5 5
Change for 45 = 5 5 5 5 5 5 5 5 5
```

   Your program should also output a message if the amount can't be changed using quarters, dimes or nickels:

```
java MakeChange 13
>
13 can't be changed
```

   No credit for non-recursive solutions!

2. (35 points) Exercise 2.8.c in the Weiss text, with the following changes: Write a single Java applet to time each of the three random permutation methods and use the Java 2D *Graph* package to graph the timings of each algorithm for given input sequences on the same graph. For each random permutation method, begin with an input value $n = 100$, and keep doubling the value of n until the time of execution for a run is greater than 30 seconds or the value of $n > 7,000,000$. Figure 1 is a sample graph.

   The *Graph* package, with examples, is available for download at:

   `http://www1.cs.columbia.edu/~allen/S14/graph/Top.html`

   You can also grab a simple applet on the class web page at:

   `www.cs.columbia.edu/~allen/S14/src/graphexample/example1.html`

   The source for the applet and Java jar file graph.jar with all the graph classes can also be downloaded from this directory:

   `www.cs.columbia.edu/~allen/S14/src/graphexample/`

3 Notes

1. Programming problem 2 requires you to generate random integers in Java. Below is how its done:

```
import java.util.Random;
...
   Random generator= new Random(); //allocates instance of Java class Random()
   x= generator.nextInt(MAX); // generates an integer between 0 and MAX-1
```

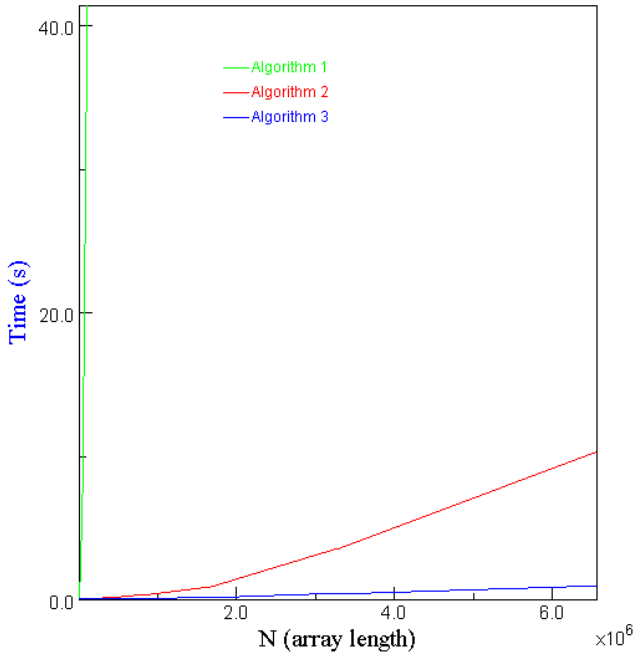

Figure 1: Plot of time vs. generating random permutations.

2. Programming problem 2 also asks you to generate timings of your program code. This is one way to do it in Java:

```
/**
  * A timing utility class useful for timing code segments
*/
public class TimeInterval {
    private long startTime, endTime;
    private long elapsedTime; // Time interval in milliseconds
// Commands
    public void startTiming() {
        elapsedTime = 0;
        startTime = System.currentTimeMillis();
    }
    public void endTiming() {
        endTime = System.currentTimeMillis();
        elapsedTime = endTime - startTime;
    }
    // Queries
    public double getElapsedTime() {
        return (double) elapsedTime / 1000.0;
    }
}
```