CSC 212 Homework # 1
Performance Analysis
Due date: 16/10/2016

Guidelines:	This is an individual assignment. The homework must be **submitted electronically through LMS**. **Hard copy submissions are not accepted**.

Problem 1

1. Order the following functions by asymptotic growth rate: $4n \log n + 2n$, 2^{10}, $2^{\log n}$, $3n + 100 \log n$, $4n$, 2^n, $n^2 + 10n$, n^3, $n \log n$. (Question R-4.8 page 182 of the textbook)

2. Show that $\log n^{2n} + n^2$ is $O(n^2)$

3. Show that $\sum_{i=1}^{5} i^3$ is $O(1)$

4. Show that $\sum_{i=1}^{n} \lceil \log i \rceil$ is a $O(n \log n)$

5. Using the definition of the Big-Oh, prove that $f(n) = 10n + 5 \log n$ is a big-oh of $g(n) = n$.

Problem 2

1. Given an n-element array X, Algorithm B chooses $\log n$ elements in X at random and executes an $O(n)$-time calculation for each. What is the worst-case running time of Algorithm B? (Question R-4.30 page 184 of the textbook)

2. Given an n-element array X of integers, Algorithm C executes an $O(n)$-time computation for each even number in X, and an $O(\log n)$-time computation for each odd number in X. What are the best-case and worst-case running times of Algorithm C? (Question R-4.31 page 184 of the textbook)

3. Given an n-element array X, Algorithm D calls Algorithm E on each element $X[i]$. Algorithm E runs in $O(i)$ time when it is called on element $X[i]$. What is the worst-case running time of Algorithm D? (Question R-4.32 page 184 of the textbook)

Problem 3

Analyze the performance of the following algorithms theoretically:

```
public void func1(int n) {
        for (int i = 0; i < n * log(n); i++) {
                System.out.println(i);
                for (int j = 2; j < n; j++) {
                        System.out.println(j);
                }
        }
        System.out.println("Goodbye!");
}
```

```
public void func2(int n) {
        for (int i = 0; i < n * n; i++) {
                System.out.println(i);
                for (int j = 2 * n; j > n; j--) {
                        System.out.println(j);
                }
        }
                System.out.println("Goodbye!");
}
```

```
public void func3(int n) {
        for (int i = n; i > 0; i--) {
                System.out.println(i);
                for (int j = 0; j < i; j++) {
                        System.out.println(j);
                }
        }
        System.out.println("Goodbye!");
}
```

```
void func4(int n) {
        int m = 1;
        while( m <= n ) {
                system.out.println(m);
                i = n;
                while (i > 0 ) {
                        system.out.println(i);
                        i = i / 2;
                }
                m++;
        }
}
```

Problem 4

The class *Sort* below implements three sorting algorithms: selection sort, bubble sort and Quicksort.

```
import java.util.Arrays;

public class Sort {
        public static void selectionSort(double[] A, int n) {
                for (int i = 0; i < n - 1; i++) {
                        int min = i;
                        for (int j = i + 1; j < n; j++) {
                                if (A[j] < A[min])
                                        min = j;
                        }
                        double tmp = A[i];
                        A[i] = A[min];
                        A[min] = tmp;
                }
        }

        public static void bubbleSort(double A[], int n) {
                for (int i = 0; i < n - 1; i++) {
                        for (int j = 0; j < n - 1 - i; j++) {
                                if (A[j] < A[j + 1]) {
                                        double tmp = A[j];
                                        A[j] = A[j + 1];
                                        A[j + 1] = tmp;
                                }
                        }
                }
        }

        public static void quickSort(double A[], int n) {
                Arrays.sort(A, 0, n - 1);
        }
}
```

Conduct an experimental analysis of these three algorithms as follows:

- Use arrays of sizes ranging from 10000 to 50000 with step size 10000 (so in total you have 5 different sizes).
- Give the same input to all three algorithms.
- Fill the array with random numbers (use `Math.random()`).
- For each input repeat the execution 100 times, measure the execution in nanoseconds (use `System.nanoTime()`), and report the average time in milliseconds.

1. Write the code used for the experimental analysis.
2. Report the results as a table and as a graph.

3. Which of the three algorithms is the fastest?

4. Which of *selection sort* and *bubble sort* is faster? Which one has a larger growth rate?