CSC 212 Homework # 1
Performance Analysis
Due date: 14/10/2017

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

Problem 1

1. Show that $5n^2 + 2n + 1$ is $O(n^2)$
2. What is the Big oh of $n^2 + n\log(n)$? prove your answer.
3. Show that $2n^3 \notin O(n^2)$.
4. Assume that the expression below gives the processing time $f(n)$ spent by an algorithm for solving a problem of size n.
$$10n + 0.1n^2$$
   (a) Select the dominant term(s) having the steepest increase in n.
   (b) Specify the lowest Big-Oh complexity of the algorithm.
5. Determine whether each statement is *true* or *false* and correct the expression in the latter case:
   (a) $100n^3 + 8n^2 + 5n$ is $O(n^4)$.
   (b) $100n^3 + 8n^2 + 5n$ is $O(n^2 \log n)$.
6. Show that $\log_a(n) \in O(\log_b(n))$ for all $a, b > 0$.
7. Show that $a^n \notin O(b^n)$ if $a > b > 0$.

. . . .

Problem 2

Analyze the following code excerpts:

1.
```
int sum = 0;
for (int i = 1; i <= n ; i++)
        for (int j = 0; j <  2* i ; j++)
                sum += j;
return sum;
```

2.
```
for (int i = 0; i < n * n * n; i++) {
        System.out.println(i);
        for (int j = 2; j < n; j++)
                System.out.println(j); }
 System.out.println("End!");
```

3.
```
int k = 100, sum = 0;
for (int i = 0; i < n; i++)
        for (j = 1; j <= k; j++) {
                sum = i + j;
                System.out.println(sum);
        }
```

. . . .

Problem 3

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. Give in asymptotic notation the running time for the following algorithms:

   (a) Vector-vector addition (the vectors are of size n).

   (b) Dot product of two vectors (the vectors are of size n).

   (c) Matrix-vector multiplication (the matrix is of size $m \times n$, the vector is of size n).

   (d) Matrix addition (the two matrices are of size $m \times n$).

   (e) Matrix-Matrix multiplication (the two matrices are of size $m \times k$ and $k \times n$ respectively).

. . . .

Problem 4

For the following function:

1. Give two example inputs leading to the best and worst running time respectively.
2. Analyze the performance of the function in each case (best and worst).

```
int func1(int[] A, int n) {
        int i = 0;
        int j = n - 1;
        int sum = 0;
        while (i <= j) {
                if (A[i] > A[j]) {
                        for (int k = i; k <= j; k++) {
                                sum += A[k];
                        }
                }
                i++;
                j--;
        }
        return sum;
}
```

....

Problem 5

The space performance (or space complexity) of an algorithm is the maximum amount of memory (in bytes) used at any point of the algorithm **ignoring the input size**.

Example 5.1. *The function* `sum1` *below uses two variables (*`sum` *and* `i`*) in addition to the input* A*, so it is* $O(1)$ *in space (and* $O(n)$ *in time).*

```
int sum1(int[] A, int n) {
        int sum = 0;
        for(int i = 0; i < n; i++) {
                sum += A[i];
        }
        return sum;
}
```

On the other hand, the function `sum2` *is* $O(n)$ *in space (why?):*

```
int sum2(int[] A, int n) {
        int sum = 0;
        for(int i = 0; i < n; i++) {
                int[] B = new int[i + 1];
                for(int j = i; j <= i; j++) {
                        B[j] = A[j] - A[i];
                }
                for(int j = i; j <= i; j++) {
                        sum += B[j];
                }
        }
        return sum;
}
```

What is the space complexity of the following function? Justify your answer.

```
void func3(int[] A, int n) {
        int[][] B = new int[n][];
        for (int i = 0; i < n; i++) {
                B[i] = new int[i + 1];
                for (int j = 0; j <= i; j++) {
                        if (A[j] > A[i])
                                B[i][j] = A[j];
                        else
                                B[i][j] = 0;
                }
        }
}
```

....

Problem 6

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?

. . . .