Assignment 1: Big-O-Highscores

Write a report (pdf or word) in which you answer the questions below. For each question:

1. repeat the question you are answering;
2. explain your approach;
3. describe your answer(s);
4. show (relevant) code snippets;
5. show the output of your program.

Once your report is finished, make sure your names and student numbers are on the title page, and send it to your instructor before the deadline.

Note that you are required to program in C# or C++, and must work in pairs of two. Assignments are graded with a G (good), V (sufficient) or O (insufficient).

Consider an unordered list of 100 players. Each player is a assigned a highscore between 0 and 10000. We are interested in writing a method to check whether or not all high scores are unique.

Question 1)

Write a program that generates an array with 100 random numbers (high scores between 0 and 10000) and then checks if the numbers are unique - output is yes or no. (hint: you can use a loop inside another loop).

Question 2)

Execute the program above 5000 times, and report how many times the array contains at least one duplicate. What is the probability that a high score list contains at least one duplicate value?

Question 3)

Write a program that initializes an array of length 10000 with all values at 0. Then, add +1 to a random element, and repeat 100 times. Now check if the array contains a value greater than 1 (output is a boolean).

Question 4)

Execute the program above 5000 times, and report how often at least one duplicate value is found. Again, what is the probability that a high score list contains at least one duplicate value?

Question 5)

Are the probabilities found under 2) and 4) equal? Explain why (not).

```
The time complexity, e.g.  O(n), of an algorithm can be
determined in two ways: by looking at the code, or by measuring
the running time. For questions 6 and 7 you'll be finding the
Big-O of the algorithms you created earlier (using the number
of players as n), by analyzing the code. Questions 8 and 9 ask
you to find out the Big-O by measuring the running time.
```

Question 6)

Consider the program you created to answer question 1). What is the time complexity (in terms of Big-O) of this program? Motivate your answer.

Question 7)

Consider again the program you created to answer question 3). What is the time complexity (in terms of Big-O) of this program? Motivate your answer.

Question 8)

Taking n as the number of players, fill in the following table for your solution of 1). Make sure you take the average time over at least 1000 execution runs. Do these results confirm your analysis reported in question 6)?

n	execution time (in milliseconds)
100	
200	
400	
800	

(hint for C# users: use the Stopwatch class. See http://www.dotnetperls.com/stopwatch)

Question 9)

Taking n as the number of players, fill in the following table for your solution of 3). Make sure you take the average time over at least 1000 execution runs. Do these results confirm your analysis reported in question 6)?

n	execution time (in milliseconds)
100	
200	
400	
800	