CS242 - Fall 2018 - Assignment #2

Assigned: October 9th, 2018
Due: October 18th, 2018
No late submissions. You will be able to complete your submission later if appropriate, but if you do not submit a significant part of the assignment by the due date you will not be able to submit afterwards. Extra credits do not have a deadline.

Collaboration policy: The goal of homework is to give you practice in mastering the course material. Consequently, you are encouraged to collaborate with others. In fact, students who form study groups generally do better on exams than do students who work alone. If you do work in a study group, however, you owe it to yourself and your group to be prepared for your study group meeting. Specifically, you should spend at least 30–45 minutes trying to solve each problem beforehand. If your study group is unable to solve a problem, it is your responsibility to get help from the instructor before the assignment is due.

For this assignment, you can form a team of up to three members. Each team must write up each problem solution and/or code any programming assignment without external assistance, even if you collaborate with others outside your team for discussions. You are asked to identify your collaborators outside your team. **If you did not work with anyone outside your team, you must write "Collaborators: none."** If you obtain a solution through research (e.g., on the web), acknowledge your source, but write up the solution in your own words. **It is a violation of this policy to submit a problem solution that any member of the team cannot orally explain to the instructor.** No other student or team may use your solutions; this includes your writing, code, tests, documentation, etc. It is a violation of this policy to permit anyone other than the instructor and yourself read-access to the location where you keep your solutions.

Submission Guidelines: Your team has to submit your work on Blackboard by the due date. **Only one submission per team is necessary.** For each of the programming assignments you must use the header template provided in Blackboard. Make sure that you identify your team members in the header, and any collaborators outside your team, if none, write "none". Your answers to questions that do not require coding must be included in this header as well. Your code must follow the Java formatting standards posted in Blackboard. Format will also be part of your grade. To complete your submission, you have to upload two files to Blackboard: your source file and your class file. **The submission will not be accepted in any other format.**

Style and Correctness: Keep in mind that your goal is to communicate. Full credit will be given only to the correct solution which is described **clearly**. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Assignment #2

Programming Assignment Grading Rubric:

The following rubric applies only to programming assignments.

Program characteristic	Program feature	Credit possible		
Design 30%	Algorithm	30%		
Functionality 30%	Program runs without errors	20%		
	Correct result given	10%		
Input 15%	User friendly, typos, spacing	10%		
	Values read in correctly	5%		
Output 15%	Output provided	10%		
	Proper spelling, spacing, user friendly	5%		
Format 10%	Comments, name	5%		
	Indentation	5%		
	TOTAL	100%		

1(20)	2(40)	3(20)	4(20)	**TOTAL**(100)	EC

Assignment:

We have studied a variety of sorting algorithms assuming that the input does not have repetitions for simplicity. Under such assumption, we have computed upper bounds on the running time of each algorithm for the worst-case input. But how good/bad are these algorithms if we know that the input has a lot of repetitions? The purpose of this assignment is to evaluate experimentally the time performance of two sorting algorithms on inputs that have many repeated items. Specifically, do the following.

1. (20 points) Implement a main method that calls 2 methods, measuring the running time of each. Specifically, your program must prompt the user to enter the size of the input n and the expected number of repetitions r. Then, create TWO arrays of n doubles containing the SAME numbers chosen uniformly at random from the interval $[0, 1]$. Also, for each number, choose at random how many repetitions in the range $[1, 2r]$. Then, call one sorting method in each array, and output the running time taken by each.

   Notice that it is crucial to create two separate arrays with the same content. Indeed, if they had different content the comparison could be unfair, and if we had only one array we would be sorting a sorted array when we call the second method (recall that arrays are passed by reference).

2. (40 points) Implement two sorting methods: Quick Sort, and Bucket Sort. Each method must receive the array produced by the main method as a parameter and sort it. The pseudocode for Quick Sort and Bucket Sort is attached.

3. (20 points) Test the algorithms with different values of n and r and fill TWO tables like the following with the running times measured (put the table in the code header).

	n_1	n_2	n_3	n_4	n_5	n_6
r_1						
r_2						
r_3						
r_4						
r_5						
r_6						

To choose your input values n and r, stretch your code with values of n and r as large as possible, before you run out of memory or the execution takes just too long. Then, come down to smaller values. You should have a combination of small/large n's and small/large r's. If possible, you should try to find values for which you can distinguish better performance for each.

4. (20 points) Based on the running times observed, draw conclusions about the experimental performance. Are both algorithms always similar? Do the repetitions have an impact? Justify based on your measurements, not in the theoretical upper bound. Provide your answers in the remarks section of the code header.

5. (Extra Credit) Modify the method `Partition()` in Quick Sort to have the pivot chosen at random.