CS608 - Fall 2020 - Assignment #4

Assigned: October 17th, 2020
Due: October 25th, 2020 (Extra credit: December 1st, 2020)
NO LATE SUBMISSIONS.
Non-coding answers may be resubmitted once, after receiving feedback on the first attempt. **That is, if you do not submit anything in the first attempt, you cannot resubmit.**

Collaboration policy: The goal of assignment 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 (no email) by the due date. Only one submission per team is necessary. For each class in 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 code must follow the Java formatting standards posted in Blackboard. Format will also be part of your grade. To complete the submission, you have to upload two files to Blackboard: your source file and your class file. Your answers to questions that do not require coding must be included in the remarks section of the header. **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 #4 Grading Rubric

Coding:

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%		

Non-coding:
Embedded in questions.

1(10)	2(20)	3(10)	4(30)	5(30)	**TOTAL**(100)	EC

Assignment:

Quick Sort is a popular sorting algorithm because it usually achieves optimal $O(n \log n)$ running time in practice. However, in the worst case, the running time can be quadratic. The method to choose the pivot is crucial to avoid the worst cases. The purpose of this homework is to evaluate experimentally the performance of Quick Sort for different methods of choosing the pivot, and compare with Insertion Sort, which is simpler but quadratic.

1. **(10 points)** Write a method that implements Insertion Sort.
2. **(20 points)** Write methods for two versions of Quick Sort, depending on how the pivot is chosen from each subarray to be sorted, as follows.
   (a) Choose the pivot from the first position of the subarray.
   (b) Pick three indexes of the subarray at random, and make the pivot equal to the median of the content of those three indexes.
3. **(10 points)** Write a test program that measures the running time of the above three methods while sorting $n = 1000000$ numbers (adjust n if needed, but it has to be large). Use three types of inputs: already sorted in increasing order, already sorted in decreasing order, and an input with the numbers generated at random.

   Fill the following chart with the running times measured.

version	increasing order	decreasing order	random
1			
2a			
2b			

4. **(30 points)**

   Fill the following chart with the big-O running times expected.

version	increasing order	decreasing order	random
1			
2a			
2b			

Rubric: Partial credit for partially correct table. Only if all cases are correct the following question will be graded in the first round.

5. **(30 points)** Draw conclusions from the values measured. Are the relations between the measurements in various cases consistent with your big-O conjectures? Why or why not?

   Rubric: For each column, you have to compare row to row, and for each row, column to column. For instance, for the increasing order column, if your big-O for version 1*, grows faster than for version* 2*a, the running time measured for version* 1 *must be larger than for version* 2*a. If not, you must notice it. Full credit even if you are not able to explain why not. Partial credit if you do not explain why yes. Partial credit if you do not compare all cases.*

   Extra Credit: Implement Library Sort which is a version of Insertion Sort with gaps to speed up the computation. If the pseudocode in Wikipedia (`https://en.wikipedia.org/wiki/Library_sort`) is not enough, you can download the research paper from (`https://arxiv.org/pdf/cs/0407003.pdf`).