Department Computer Science and Software Engineering
Concordia University

COMP 352: Data Structures and Algorithms
Fall 2020 - Assignment 1

Due date and time: Friday September 25th, 2020 by midnight

Written Questions (50 marks): Please read carefully: You must submit the answers to all the questions below. However, only one or more questions, possibly chosen at random, will be corrected and will be evaluated to the full 50 marks.

Question 1

Given an array of integers of any size, $n \geq 1$, write an algorithm in **pseudo code** (not Java or any other programming language) that would read each element of the array and if the element is odd it moves it to the back of the array, whereas if it is even, it stays intact. For example, given the following array, [51, 88, 3, 70, 96, 38, 47] your solution should return [88, 70, 96, 38, 47, 51, 3].
(Notice that this is just an example. Your solution must not refer to this particular example). Your algorithm **cannot** use any auxiliary storage such as 'extra' arrays to perform what is needed.

a) What is the time complexity of your algorithm, in terms of Big-O?
b) What is the space complexity of your algorithm, in terms of Big-O?

Question 2

Develop well-documented **pseudo code** that finds the largest and smallest product of two elements in an array of n integers, $n \geq 1$. The code must display the values and the indices of these elements, and the value of that product. For instance, given the following array [42, 21, 10, 31, 7] your code should find and display something similar to the following (notice that this is just an example. Your solution must not refer to this particular example):

- The two indices with largest product between their values are: index 0 and index 3, storing values 42 and 31, and the value of their product is 1302.
- The two indices with smallest product between their values are: index 2 and index 4, storing values 10 and 7 and the value of their product is 70.

In case of multiple occurrences of the smallest or largest product, the code should display the first found occurrence.

a) Briefly justify the motive(s) behind your design.
b) What is the time complexity of your solution? You must specify such complexity using the Big-O notation. Explain clearly how you obtained such complexity.

Question 3

Prove or disprove the following statements, using the relationship among typical growth-rate functions seen in class.

a) $8000000n^2 \log n + n^3$ is $O(n^3 \log n)$
b) $10^6n^2 + 3n^7 + 5n^3$ is $\Theta(n^3)$
c) $0.1n^3 + 0.0000005n^6$ is $\Theta(n^3)$
d) $n^4 + 0.0000001n^3$ is $\Omega(n^3)$
e) $n!$ is $\Theta(2^n)$
f) n^n is $\Omega(n!)$

Programming Questions (50 marks):

In this programming assignment, you will design in pseudo code and implement in Java two versions of a program that takes as input a string of any length of random number of binary characters "0 " and "1" and a masked "*" character at some positions, and find all possible sequences of binary strings that can be construct by replacing the masked"*" character by either 0 or 1.

Version 1:

In your first version, you must write a **recursive** method called ***RevealStr***, which takes a string, (and any other parameters; e.g. length, start index and end index, etc., if needed) and generates ALL possible combinations of that string without the mask "*" characters.

For example; given the string

- "1011*00*10", the method ***RevealStr*** will display something like:

  1011000010
  1011000110
  1011100010
  1011100110

- "1011*00*1011*0**" the method ***RevealStr*** will display something like:

1011000010110000	1011000110110010	1011100010111000
1011000010110001	1011000110110011	1011100010111001
1011000010110010	1011000110111000	1011100010111010
1011000010110011	1011000110111001	1011100010111011
1011000010111000	1011000110111010	1011100110110000
1011000010111001	1011000110111011	1011100110110001
1011000010111010	1011100010110000	1011100110110010
1011000010111011	1011100010110001	1011100110110011
1011000110110000	1011100010110010	1011100110111000
1011000110110001	1011100010110011	1011100110111001
		1011100110111010
		1011100110111011

You will need to run the program multiple times. With each run, you will need to provide a random generated string size with a mask "*" character in an incremented number from 2, 4, 6, up to 100 (or higher value if required for your timing measurement) and measure the corresponding run time for each run. You can use Java's built-in time function for finding the execution time. You should redirect the output of each program to an *out.txt* file. You should write about your observations on timing measurements in a separate text file. You are required to submit the two fully commented Java source files, the compiled executables, and the text files.

Briefly explain what is the complexity of your algorithm. More specifically, has your solution has an acceptable complexity; is it scalable enough; etc. If not, what are the reasons behind that?

Version 2:

In this version, you will need to provide and alternative/different solution to solve the same exact problem as above). This second solution must be **iterative and not recursive**, and can use any linear data structure such as array, stack, queue, etc.

a) Explain the details of your algorithm, and provide its time and space complexity. You must clearly justify how you estimated the complexity of your solution.
b) Compare the complexities between version 1 and version 2

b) Submit both the pseudo code and the Java program, together with your experimental results. Keep in mind that Java code is **not** pseudo code. See the full details of submission details below.

The written part must be done individually (no groups are permitted). The programming part can be done in groups of two students (maximum.

For the written questions, submit all your answers in PDF (no scans of handwriting; this will result in your answer being discarded) or text formats only. Please be concise and brief (less than ¼ of a page for each question) in your answers. Submit the assignment under Theory Assignment 1 directory in EAS or the correct Dropbox/folder in Moodle (depending on your section).

For the Java programs, you must submit the source files together with the compiled files. The solutions to all the questions should be zipped together into one .zip or .tar.gz file and submitted via Moodle/EAS under Programming 1 directory or under the correct Dropbox/folder. You must upload at most one file (even if working in a team; please read below). In specific, here is what you need to do:

1) Create **one** zip file, containing the necessary files (.java and .html). Please name your file following this convention:
If the work is done by 1 student: Your file should be called *a#_studentID*, where # is the number of the assignment *studentID* is your student ID number.
If the work is done by 2 students: The zip file should be called *a#_studentID1_studentID2*, where # is the number of the assignment *studentID1* and *studentID2* are the student ID numbers of each student.

2) If working in a group, only one of the team members can submit the programming part. Do not upload 2 copies.

Very Important: Again, the assignment must be submitted in the right folder of the assignments. Depending on your section, you will either upload to Moodle or EAS (your instructor will indicate which one to use). **Assignments uploaded to an incorrect folder will not be marked and result in a zero mark. No resubmissions will be allowed.**

- ⇨ Additionally, for the programming part of the assignment, a demo is required (please refer to the courser outline for full details). The marker will inform you about the demo times. **Please notice that failing to demo your assignment will result in zero mark regardless of your submission.** If working in a team, both members of the team must be present during the demo.