COMP 352: Data Structures and Algorithms
Winter 2021 - Assignment 1

Concordia University
Engineering and Computer Science

Due date and time: Friday February 5th, 2021 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

Consider an array A[1..n] of random length and random positive integer values. We define a subarray of A as a contiguous segment of A. We denote the subarray from position k to position l (both included) as A[k..l]. The subarray A[k..l] is an *ascent* if A[j] ≤ A[j + 1] for all j where k ≤ j < l. In other words, an ascent is a nondecreasing segment of A.

Develop well-documented **pseudo code** (not java code) that compute the maximal length of an ascent in A. For instance, given an array A =[5; 3; 6; 4; 6; 6; 7; 5], your algorithm should display:

The maximal length of an ascent would be 4, and A[4..7] = [4; 6; 6; 7] is the longest ascent in the array A.

(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) Briefly justify the motive(s) behind your design.
b) What is the time complexity of your algorithm, in terms of Big-O?
c) What is the space complexity of your algorithm, in terms of Big-O?

Question 2

Develop well-documented **pseudo code** (not java code) that finds the largest and smallest sum 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 sum. For instance, given the following array [13;9;47;35;6;29;69;12] 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 sum between their values are: index 2 and index 6, storing values 47 and 69, and the value of their sum is 116.
- The two indices with smallest sum between their values are: index 1 and index 4, storing values 9 and 6 and the value of their sum is 15.

In case of multiple occurrences of the smallest or largest sum, 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 and explain clearly how you obtained such complexity.
c) What is the space complexity of your algorithm?

Question 3

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

a) $n^6 \log n + n^4$ is $O(n^4 \log n)$

b) $10^6n^6 + 5n^3 + 6000000n^2 + n$ is $\Theta(n^3)$
c) $6n^n$ is Ω (n!)
d) $0.5n^8 + 700000n^5$ is $O(n^8)$
e) $n^{13} + 0.0008n^6$ is $\Omega(n^{12})$
f) n! is $O(5^n)$

Programming Questions (50 marks):

In this programming part you are asked to implement phase-1 of a game called #tictactoe. #tictactoe game consists of one row of any number of squares, where some of the squares store noughts and crosses (i.e., Os and Xs), and the remaining squares store the character "#". The squares storing the "#" character each hide their actual content which must be either an X or an O.

In this programming assignment, you will design in pseudo code and implement in Java two versions of #tictactoe game phase-1. A program that takes as input your game row of any length of random number of squares with X and O, and hidden squares with "#", and finds all possible rows of noughts and crosses that can be constructed by replacing the hidden squares (storing "#") with either X or O.

Version 1:
In your first version, you must write a **recursive** method called ***UnHide*** which reads the row (and any other parameters; e.g., length, start index and end index of row, etc., if needed) and generates ALL possible combinations of that rows without the hidden "#" square.

For example; given:
a) A row [XOXX#OO#XO], the method *UnHide* will display something like:

XOXXOOOOXO
XOXXOOOXXO
XOXXXOOOXO
XOXXXOOXXO

b) A row [XOXX#OO#XOXX#O##] the method *UnHide* will display something like:

XOXXOOOOXOXXOOOO	XOXXOOOXXOXXOOXO	XOXXXOOOXOXXXOOO
XOXXOOOOXOXXOOOX	XOXXOOOXXOXXOOXX	XOXXXOOOXOXXXOOX
XOXXOOOOXOXXOOXO	XOXXOOOXXOXXXOOO	XOXXXOOOXOXXXOXO
XOXXOOOOXOXXOOXX	XOXXOOOXXOXXXOOX	XOXXXOOOXOXXXOXX
XOXXOOOOXOXXXOOO	XOXXOOOXXOXXXOXO	XOXXXOOXXOXXOOOO
XOXXOOOOXOXXXOOX	XOXXOOOXXOXXXOXX	XOXXXOOXXOXXOOOX
XOXXOOOOXOXXXOXO	XOXXXOOOXOXXOOOO	XOXXXOOXXOXXOOXO
XOXXOOOOXOXXXOXX	XOXXXOOOXOXXOOOX	XOXXXOOXXOXXOOXX
XOXXOOOXXOXXOOOO	XOXXXOOOXOXXOOXO	XOXXXOOXXOXXXOOO
XOXXOOOXXOXXOOOX	XOXXXOOOXOXXOOXX	XOXXXOOXXOXXXOOX
		XOXXXOOXXOXXXOXO
		XOXXXOOXXOXXXOXX

You will need to run the program multiple times. With each run, you will need to provide a random generated row size with a hidden" #" tail 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 an 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, list. 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

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 questions 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 (text document format or clear handwriting: if your answer is not clearly written this will result in your answer being discarded). 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 programming part, 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:
- 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.