Department Computer Science and Software Engineering
Concordia University

COMP 352: Data Structures and Algorithms
Fall 2021 - Assignment 2

Due date and time: Monday November 1, 2021 @ 11:59 PM

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

I) Develop well-documented pseudo code that finds all consecutive similar elements of a given array (of any size *n)*. The code must display the start indices where the values start to repeat, as well as the values of these elements. For instance, given the following array *A*:
(22, 9, 61,61, 61, 21, 0, 9, 9, 9, 9, 35, 81,81, 9, 5, 5), 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):

```
Value 61 is repeated 3 times starting at Index 2
Value 9 is repeated 4 times starting at Index 7
Value 81 is repeated 2 times starting at Index 12
Value 5 is repeated 2 times starting at Index 15
```

a. Briefly justify the motive(s) behind your design.
b. What is the Big-O time complexity of your solution? Explain clearly how you obtained such complexity.
c. What is the Big-Ω time complexity of your solution? Explain clearly how you obtained such complexity.
d. What is the Big-O *space* complexity of your solution?

II) Develop a well-documented pseudo code that solves the problem stated in part 1), using either a stack *S* or a queue *Q* to perform what is needed.
a. Briefly justify the motive(s) behind your design.
b. What is the Big-O time complexity of your solution? Explain clearly how you obtained such complexity.
c. What is the Big-Ω time complexity of your solution? Explain clearly how you obtained such complexity.
d. What is the Big-O *space* complexity of the utilized stack or queue? Explain your answer.

Question 2

I) Develop well-documented pseudo code that finds all two elements (non-negative numbers) of a given array that modulo up exactly to a given value x. The code must display the indices and the values of these elements. For instance, given the following array (123,73,39,12,14,9,113,93,203,22,25,10) and *x* as 3, 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). Notice that the output should show all possible pairs:

All pairs of elements of the array that modulo to a value of 3 are:
Indices 0 & 3 with values 123 & 12 (e.g., 123 % 12 returns 3)
Indices 1 & 4 with values 73 & 14
Indices 2 & 5 with values 39 & 9
Indices 6 & 9 with values 113 & 22
etc.

II) Briefly justify the motive(s) behind your design.

III) What is the Big-O time and space complexity of your solution? Explain clearly how you obtained such complexity.

IV) What is the Big-Ω time and space complexity of your solution? Explain clearly how you obtained such complexity.

Question 3

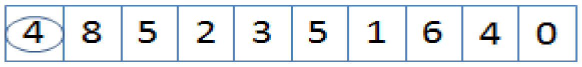
For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) is θ(g(n)). For each pair, determine which relationship is correct. Justify your answer.

i) $f(n) = \log^3 n$; $g(n) = \sqrt{n}$.
ii) $f(n) = n\sqrt{n} + \log n$; $g(n)=\log n^2$.
iii) $f(n) = n$; $g(n) = \log^2 n$.
iv) $f(n) = \sqrt{n}$; $g(n) = 2^{\sqrt{logn}}$.
v) $f(n) = 2^{n!}$; $g(n) = 3^n$.
vi) $f(n) = 2^{10n}$; $g(n) = n^n$.
vii) $f(n) = (n^n)^5$; $g(n) = n^{(n^5)}$.

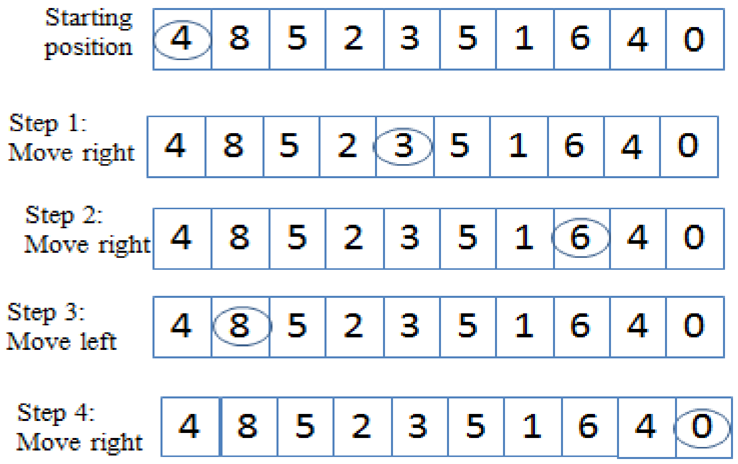
Programming Questions (50 marks):

In this programming part you are asked to implement a game called ***HitZero*.**

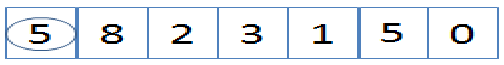
HitZero is a 1-player game consisting of a row of squares of any size each containing an integer, like this:



The rules of the game are simple. The circle on the initial square is a marker that can move to other squares along the row**.** At each step in the game, you may move the marker the number of squares indicated by the integer in the square it currently occupies. The marker may move either left or right along the row but may not move past either end**.** For example, in the above array, the only legal first move is to move the marker four squares to the right because there is no room to move four spaces to the left.
The goal of the game is to move the marker to the far right index, which always contains "0"**.** Considering the above given array as an example, you can solve the game by making the following set of moves:



Though the *HitZero* game may be solvable, and possibly with more than one solution, some configurations of this form may be impossible to solve, such as the following one:



In this configuration, you will bounce between the two 5's, but you cannot reach any other square. Other configurations may attempt all possible moves, however none of them would allow to hit the "0" at the far right end.

Requirements:

1. In this programming assignment, you will design using pseudo code and implement in java code **two versions** of the *HitZero* game**.**
   - The first version will be completely based on recursion.
   - The second one will be based on a stack, a queue, or a list.

2. Your solution takes a starting position of the marker along with the list of squares. The starting position can be any index between 0 and N-2 (where N is the size of the array)**.** Your solution should return *true* if it is possible to solve the game from the starting point, and *false* if a solution is impossible. Your solution should work for any size of the game's row (any array size), and a random positive value (>0) in each square, except for the last entry which always has "0".

3. At the end of the game, the values of the elements in the list must be the same after calling your solution as they are beforehand, (that is, if you change them during processing, you need to change them back.).

   a) Briefly explain the *time* and *space* complexity for both versions of your game. You can write your answer in a separate file and submit it together with your submissions.
   b) For the first version of your solution, describe the type of recursion used in your implementation. Does the particular type of recursion have an impact on the time and memory complexity? Justify your answer.
   c) For the second part of your solution, justify why you choose that particular data structure (e.g. why you choose a stack and not a queue, etc.)
   d) Provide test logs for at least 20 different game configurations. The test cases should sufficiently show the correctness of your solution with various game sizes and input values.
   e) If possible, explain how one can detect unsolvable array configurations and whether there exists a way to speed up the execution time. Answering this question is optional and you can earn bonus marks by submitting a good solution.

   You are required to submit the two fully commented Java source files, the compiled files (.class files), and the text files.

   You will need to submit both the pseudo code and the Java program, together with your experimental results. Keep in mind that Java code is **not** pseudo code.

Deliverables

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 2 directory on Moodle.

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 under Programming 1 directory on Moodle. 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 source-code files (.java, .c, etc.)

You must name your file using the following 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 of the group members.

2) Assignments must be submitted in the right folder/dropbox on the course Moodle page. Assignments uploaded to an incorrect folder will not be marked and result in a **zero mark**. **No resubmissions will be allowed.**

Demo

A demo is needed for this assignment and your lab instructors will communicate the available demo times to you, where you **must** register a time-slot for the demo, and you must prepare your assignment and be ready to demo at the start of your time-slot. If the assignment is done by 2 members, then **both members must** be present for the demo. During your presentation, you are expected to demo the functionality of the application, explain some parts of your implementation, and answer any questions that the lab instructor may ask in relation to the assignment and your work. Different marks may be assigned to the two members of the team if needed. **Demos are mandatory. Failure to demo your assignment will entail a mark of zero for the assignment regardless of your submission. Please read the following carefully!**

- **If you book a demo time, and do not show up, for whatever reason, you will be allowed to reschedule a second demo but a penalty of 50% will be applied.**
- **Failing to demo at the second appointment will result in zero marks and no more chances will be given under any conditions.**