**Department Computer Science and Software Engineering**
**Concordia University**

**COMP 352: Data Structure and Algorithms**
**Fall 2019**
**Assignment 2**
**Due date and time: Monday October 28th, 2019 by midnight**

## Written Questions (50 marks):

### Question 1

1) Assume that we have one single array A of size N, where N is an even value, and that the array is not expandable. We need to implement 2 stacks. Develop well-documented pseudo code that shows how these two stacks can be implemented using this single array. There are two cases you must consider:

   **Case I:** Fairness in space allocation to the two stacks is required. In that sense, if Stack1 for instance use all its allocated space, while Stack2 still has some space; insertion into Stack1 cannot be made, even though there are still some empty elements in the array;

   **Case II:** Space is critical; so you should use all available elements in the array if needed. In other words, the two stacks may not finally get the same exact amount of allocation, as one of them may consume more elements (if many push() operations are performed for instance into that stack first).

   For **each** of the two cases:

   a. Briefly describe your algorithm;
   b. Write, in pseudocode, the implementation of the following methods, for each of the stacks: push(), pop(), size(), isEmpty() and isFull();
   c. What is the Big-O complexity for the methods in your solution? Explain clearly how you obtained such complexity.
   d. What is the Big-Ω complexity of your solution? Explain clearly how you obtained such complexity.

      ⇨ Is it possible to solve the same problem, especially for Case II, if three stacks were required? If so, do you think the time complexity will change from your solution above? You do not need to provide the answer to these questions; but you certainly need to think about it!

### Question 2

An Antique dealer needs to store records of the items in the inventory into a stack (the dealer believes that keeping items longer will be beneficial; nonetheless, the business logic here is not important!). Besides the usual push() and pop(); the dealer needs to always know what is the most expensive item in the inventory/stack. You are hired to implement a new method, called **max()**, which returns the current maximum value for any item in the stack. Notice that the stack grows and shrinks dynamically as elements are added or removed from the stack (i.e. the contents of the stack are not fixed).

   e. Write the pseudocode of the max() method.
   f. What is the Big-O complexity of your solution? Explain clearly how you obtained such complexity.
   g. Is it possible to have **all** three methods (push(), pop() and max()) be designed to have a complexity of **O(1)**? If no; explain why this is impossible. If yes; provide the pseudocode of the algorithm that would allow O(1) for all three methods (this time, you do not only need to think about it, but to actually give the pseudocode if you believe a solution is feasible!)

**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. Notice that no formal proof is required; just clear justification of your determination.

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) $f(n) = \sqrt{n}$; $g(n) = 2^{\sqrt{\log n}}$.
v) $f(n) = 2^n$; $g(n) = 3^n$.

**Question 4**

Develop well-documented pseudo code that accepts an array of integers, A, of any size, then finds and removes all duplicate values in the array. For instance, given an array A as shown below:

**[22, 61,-10, 61, 10, 9, 9, 21, 35, 22,-10, 19, 5, 77, 5, 92, 85, 21, 35, 12, 9, 61]**,

your code should find and remove all duplicate values, resulting in the array looking "exactly" as follows:

**[22, 61,-10, 10, 9, 21, 35, 19, 5, 77, 92, 85, 35, 12]**.

Notice the change of size. Also keep in mind that this is just an example; your solution must not refer to this particular example; rather it must work for any given array.
Additionally, you are **only** allowed to use Stacks, Queues, or Double-ended Queues (DQ) to support your solution, *if needed*. You are **NOT** allowed to use any other ADTs.

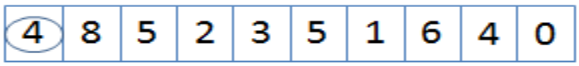
a. Briefly justify the motive(s) behind your algorithm.
b. What is the Big-O complexity of your solution? Explain clearly how you obtained such complexity.
c. What is the Big-Ω complexity of your solution? Explain clearly how you obtained such complexity.
d. What is the Big-O *space* complexity of your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

**Note:** You must submit the answers to all the questions above. However, only one or more questions, possibly chosen at random, will be corrected and will be evaluated to the full 50 marks.

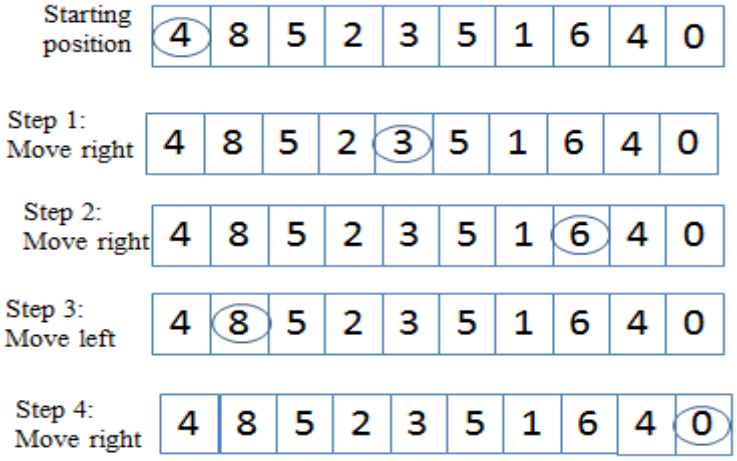
**Programming Questions (50 marks):**

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

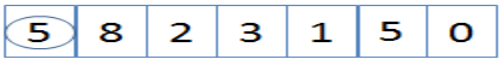
*Score FarRight* is a 1-player game consisting of a row of squares of any size each containing an integer, like this:




The rules of the game are simple. The circle on the initial square (which can initially be anywhere in the array) is a marker that can move to other squares along the row**.** At each step in the game, you may move the marker right or left by 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, 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 cave, the "0" at the far right end of the row**.** In this configuration, you can solve the game by making the following set of moves:




Though the *Score FarRight* game is solvable, actually with more than one solution—some configurations of this form may be impossible, such as the following one:




In this configuration, you will bounce between the two 5's, but you cannot reach any other square.

**Requirements:**

1. In this programming assignment, you will design, using pseudo code, and implement, in Java code, **a recursive** solution of the *Score FarRight* game**.**
2. Your solution takes a starting position of the marker along with the list of squares**.** Your solution should return *true* if it is possible to solve the game from the starting configuration and *false* if it is impossible. Your solution also should work for any size of the game's row, and a random value in each square. You may assume all the integers in the row are positive values >= 1; except for the last entry, the goal square, which is always 0. At the end of the game, the values of the

elements in the array 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 your solution. You can write your answer in a separate file and submit it together with the other submissions.
b) Describe the type of recursion used in your implementation. Does the particular type of recursion have an impact on the time or space complexity? Justify your answer.
c) Provide test logs for at least 20 different game configurations, sufficiently complete to show that your solution works for various row sizes and square values.
d) 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 a fully commented Java source files, the compiled files (.class files), and the text files. You additionally need to submit the pseudo code of your program, together with your experimental results. Keep in mind that Java code is **not** pseudo code.

**Important Notes**

**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 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 EAS under Programming 1 directory or under the correct Dropbox/folder in 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 files (.java, .doc, .html, etc.). 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, and *studentID1* and *studentID2* are the 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 EAS or to Moodle (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.