Department Computer Science and Software Engineering
Concordia University

COMP 352: Data Structures and Algorithms
Fall 2022 - Assignment 2

Due date and time: Sunday October 30, 2022 by midnight
[FIRM Date – Demos Start Monday October 31]

This assignment has two sections: a **theory section** and a **programming section**. These sections are completely independent from each other except for the due date and time.

Part I: Theory Section (50 marks): You must submit the answers to all the questions below. However, only a partial question, one question, or more questions, possibly chosen at random, will be corrected and will be evaluated to the full 50 marks.

For the question asking to find big-O or big-omega, it is implicit that you should find the tightest possible bound (i.e. while n+1 is both O(n^2) and O(n), for the purpose of this assignment, only O(n) will be considered the correct answer).

Question 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.

    i. 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).

a. Write the pseudocode of the max() method.
b. What is the Big-O complexity of your solution? Explain clearly how you obtained such complexity.
c. 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

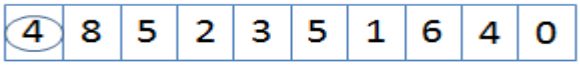
For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) is θ(g(n)). For each pair, determine which relationship is correct. Justify your answer.

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

Part II: Programming Section (50 marks):

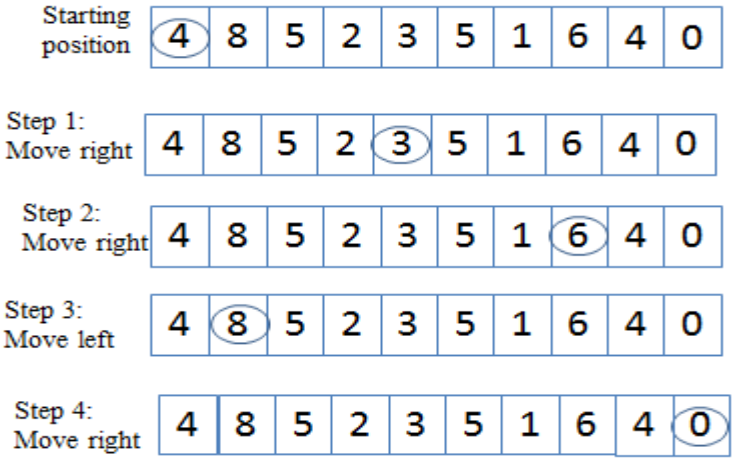
In this programming section you are asked to implement a solution for a game called ***StrikeEdgeZero*.**

StrikeEdgeZero is a 1-player game consisting of a sequence of integers, like this:

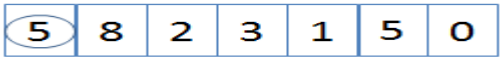


The rules of the game are simple. The circle on the initial square is an **initial marker** that can move to other positions along the sequence**.** At each step in the game, you may move the marker the number of positions indicated by the integer in the square it currently occupies. The marker may move 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 (the Edge), which always contains "0"**.** Considering the above given sequence as an example, you can solve the game by making the following set of moves:



Though the *StrikeEdgeZero* 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 or strike "0" at the edge.

Requirements:

1. In this programming assignment, you will design and implement two algorithms (i.e. pseudocode + java code) for finding a solution of the *StrikeEdgeZero* game**.**

   - The first version (A) will be completely based on recursion. Implement this in a file called Part2_A.java
   - The second one (B) will be based on a stack, a queue, or a list. Implement this in a file called Part2_B.java

2. The input is specified in a text file that should be the first argument of the program. The output should be placed in a text file that should be specified as the second argument of the program. For example, the command "java Part2_A in.txt out.txt" should take the input from the file *in.txt* and place the output in the file *out.txt*.

   The input file has on its first line one integer *M* that specifies the number of datasets in the file. The following *M* lines each specifies *M* inputs (one per line) as space separated integers. The first one is *N*, the size of the sequence including the final 0. This is followed by an integer that denotes the position of the initial marker between 0 and *N*-2 (inclusive). They are followed by *N*-1 integer values denoting the values of the sequence excluding the last 0 entry.

   The output file contains a line for each input with only one integer: 1 (if a solution exists), or 0 otherwise. No end of line after the last line. Your solution should work for any size of the game's row (i.e. any array size).

   Here is an example of input file and output file for the two examples shown above.
   Input:
   2
   10 0 4 8 5 2 3 5 1 6 4
   7 0 5 8 2 3 1 5
   Output:
   1
   0

Questions:

a) Briefly explain the *time* and *space* complexity for both versions of your game.
b) For the first version of your solution, (A), describe the type of recursion used in your implementation. Does this particular type of recursion have an impact on the time and memory complexity? Justify your answer.
c) For the second part of your solution, (B), justify why you choose that particular data structure (e.g. why you choose a stack and not a queue, etc.)
d) If possible, explain how one can detect unsolvable array configurations and whether there exists a way to speed up the execution time.

Submission

Part I:

- **Part I must be submitted individually (No groups are allowed) under the submission folder: Assignment 2 – Part I. 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.**

Part II:

- **For Part II, a group of 2 (maximum) is allowed. No additional marks are given for working alone.**
- **If working alone, you need to zip (see below) and submit your zipped file under the submission folder: Assignment 2 - Part II.**
- **If working in a group, only one submission is to be made by either of the two members (do not submit twice). You need to zip (see below) and submit your zipped file, under the submission folder: Assignment 2 - Part II.**

Submission format: All assignment-related submissions that have more than one document must be adequately archived in a ZIP file using your ID(s) and last name(s) as file name. The submission itself must also contain your name(s) and student ID(s). Use your "official" name only - no abbreviations or nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. If working in a group, the file name must include both IDs and last names.

IMPORTANT: For Part II of the assignment, a demo for about 5 to 10 minutes will take place with the marker. You (or **both** members if working in a group) **must** attend the demo and be able to explain their program to the marker. Different marks may be assigned to teammates based on this demo. The schedule of the demos will be determined and announced by the markers, and students must reserve a time slot for the demo (only one time-slot per group; or zero mark is given).

Now, please read very carefully:

- **If you fail to demo, a zero mark is assigned regardless of your submission.**
- **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.**