COMP352 S20 Assignment 2

Data Structures and Algorithms (Concordia University)

Department Computer Science and Software Engineering
Concordia University

COMP 352: Data Structure and Algorithms
Summer 2020
Assignment 2
Due date and time: Friday May 29, 2020 by midnight

Written Questions (50 marks):

Question 1

You need to provide a new design for the Stack data type. Besides the usual push() and pop() operations, you need to add a method called **max()**, which returns the current maximum value 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 2

2) 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 using that array. 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 answers to these questions; but you certainly need to think about them!

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) $f(n) = \log^3 n$; $g(n) = \sqrt{n} \log n$.
ii) $f(n) = n\sqrt{n} + \log n$; $g(n) = \log n^4$.
iii) $f(n) = 2n$; $g(n) = \log^2 n$.
iv) $f(n) = \sqrt{n}$; $g(n) = 2^{\sqrt{\log n}}$.
v) $f(n) = 2^n$; $g(n) = n^n$.
vi) $f(n) = 50$; $g(n) = \log 60$.

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 the lectures, we discussed about Evaluating Arithmetic Expressions (refer to your slides and the text book).

In this programming assignment, you will design, using pseudo code, and implement, in Java, two versions of *arithmetic* calculators. The first version will be based on the pseudo code in the lecture notes that uses two different stacks. The second version must be completely based on recursion. This means that you have to replace the explicit use of stacks (in the first version) by using recursion (that implicitly uses the JVM runtime stack). Your *arithmetic calculators* must read lines of text from a text file, where each line contains a syntactically correct arithmetic expression. Your output file must repeat the input line and print the computed result on the next line. Your calculators must support the following operators on integers or reals and observe the standard operator precedence as shown below (1-8, highest to lowest; same precedence evaluated left to right).

- 1. Parentheses (possibly nested ones): (,)
- unary operators
  2. factorial: !
  3. minus: -
- binary operators
  4. power function: x^y
  5. operators: *, /
  6. operators: +, -
  7. operators: >, ≥, ≤, <
  8. operators: ==, !=

For the implementation of factorial and power function you have to use your own code (refer to your slides and the text book) that should be as efficient as possible.

a) Briefly explain the time and memory complexity for both versions of your calculator. You can write your answer in a separate file and submit it together with the other submissions.
b) For the second version of your calculator 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) Provide test logs for at least 20 different and sufficiently complex arithmetic expressions that use all types of operators (including parentheses) in varying combinations.

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.

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. 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. **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. 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.