Department Computer Science and Software Engineering
Concordia University

COMP 352: Data Structures and Algorithms
Fall 2022 - Assignment 1

Due date and time: Friday September 30, 2022 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

a) Given an array of integers of any size, $n \geq 1$, write an algorithm as a pseudo code (not a program!) that would reverse every two consecutive elements of the left half of the array (i.e. reverse elements at index 0 &1, at index 2 & 3, etc.). Additionally, for the right half of the array, the algorithm must change the second element of every two consecutive elements to have to the sum of the two elements. For instance, if the right half starts at index 14, and the values at index 14 & 15 are 72 and 11, then the value at index 15 is changed to 83. If the given array is of an odd size, then the exact middle element should not be manipulated by the algorithm in any way. Finally, your algorithm **must** use the smallest auxiliary/additional storage to perform what is needed.

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?

Hint: Please read the question carefully!

Question 2

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

a) $n^{22}\log n + n^7$ is $\Omega(n^6 \log n)$

b) $1o^7n^5 + 5n^4 + 9000000n^2 + n$ is $\Theta(n^7)$

c) $n!$ is $\Omega(n^n)$

d) $0.01n^8 + 800000n^6$ is $\Theta(n^8)$

e) $n^9 + 8000n^7$ is $\Omega(n^{16})$

f) $n!$ is $O(4^n)$

Question 3

a) Given an ***unsorted*** array A of integers of any size, $n \geq 6$, and an integer value x, write an algorithm as a pseudo code (not a program!) that would find out if there exist **EXACTLY** 5 occurrences in the array with value x.

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?
d) Will time complexity change if *A* was given as a ***sorted*** array? If yes; give a new algorithm that achieves this better complexity (indicate the time complexity as of that algorithm). If no, explain why such new constraints/conditions cannot lead to a better time complexity.

Programming Questions (50 marks):

In class, we discussed about the two versions of Fibonacci number calculations: BinaryFib(n) and LinearFibonacci(n) (refer to your slides and the text book). The first algorithm has exponential time complexity, while the second one is linear.

a) In this programming assignment, you will design an algorithm (in pseudo code), and implement (in Java), two recursive versions of an *Oddonacci* calculator (similar to the ones of Fibonacci) and experimentally compare their runtime performances. Oddonacci numbers are inspired by Fibonacci numbers but start with three predetermined values, each value afterwards being the sum of the preceding three values. The first few Oddonacci numbers are:

   1, 1, 1, 3, 5, 9, 17, 31, 57, 105, 193, 355, 653, 1201, 2209, 4063, 7473, 13745, 25281, 46499, ...

For that, with each implemented version you will calculate Oddonacci(5), Oddonacci (10), etc. in increments of 5 up to Oddonacci(100) (or higher value if required for your timing measurement) and measure the corresponding run times. For instance, Oddonacci(10) returns 105 as value. You can use Java's built-in time function for this purpose. You should redirect the output of each program to an *OddoOut.txt* file. You should write about your observations on the timing measurements in a separate text file. You are required to submit the two fully commented Java source files, the compiled files, and the text files.

b) Briefly explain why the first algorithm is of exponential complexity and the second one is of linear complexity (more specifically, how the second algorithm resolves some specific bottleneck(s) of the first algorithm). You can write your answer in a separate file and submit it together with the other submissions.

c) Do any of the previous two algorithms use tail recursion? Why or why not? Explain your answer. If your answer is "No" then:

   Can a tail-recursive version of Oddonacci calculator be designed?

   i. If yes; provide the idea of how this can be achieved. Bonus mark will be given if you implement this tail-recursive solution.
   ii. If no, explain clearly why such tail-recursive algorithm is infeasible.

   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. See full details of submission details below.

Submission

Part I:

- **Part I must be submitted individually (No groups are allowed) under the submission folder: Assignment 1 – 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 1 - 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 1 - 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.**