COMP352 F21 Assignment 1

Data Structures and Algorithms (Concordia University)

Department Computer Science and Software Engineering
Concordia University

COMP 352: Data Structures and Algorithms
Fall 2021 - Assignment 1

Due date and time: Friday October 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

a) Given an array of integers of any size, $n \geq 4$, 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 product (multiplication) of the two elements. For instance, if the right half starts at index 14, and the values at index 14 & 15 are 5 and 12, then the value at index 15 is changed to 60. If the given array is of an odd size, then the left half is considered as the ceiling value of $n/2$. 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) $n^5 \log n$ is $O(n^7)$
b) $10^8 n^2 + 5n^4 + 5000000n^3 + n$ is $\Theta(n^4)$
c) n^n is $O(n!)$
d) $0.01n^2 + 0.0000001n^4$ is $\Theta(n^2)$
e) $1000000n^2 + 0.0000001n^5$ is $\Omega(n^3)$
f) $n!$ is $\Omega(2^n)$

Question 3

a) What is the Big-O ($O(n)$) and Big-Omega ($\Omega(n)$) time complexity for algorithm *MyAlgorithm* below in terms of n? Show all necessary steps.
b) Trace (hand-run) *MyAlgorithm* for an array A = (4,11,5,3,2). What is the resulting A?
c) What does *MyAlgorithm* do? What can be asserted about its result given any arbitrary array A of n integers?
d) Can the runtime of *MyAlgorithm* be improved easily? how?
e) Write the algorithm using actual Java code, insert any additional lines of code for the sole purpose of finding out the number of executions, then run with different initial values of the array including the one given above. Do the results correspond to your above estimate of Big-O? If no, explain

clearly the reasons behind this mismatch! Provide the Java code, and the sample outputs as part of your answers.

```
Algorithm MyAlgorithm(A, n)
  Input: Array of integer containing n elements
  Output: Possibly modified Array A
    done ← true
    j ← 0
    while j ≤ n - 2 do
      if A[j] > A[j + 1] then
        swap(A[j], A[j + 1])
        done:= false
      j ← j + 1
    end while
    j ← n - 1
    while j ≥ 1 do
      if A[j] < A[j - 1] then
        swap(A[j - 1], A[j])
        done:= false
      j ← j - 1
    end while
  if ¬ done
    MyAlgorithm(A, n)
  else
    return A
```

Programming Questions (50 marks):

In class, we discussed the two versions of recursive Fibonacci number calculations: BinaryFib(n) and LinearFibonacci(n) (refer to your slides and the text book).

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 *out.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 executables, and the text files.

a) Briefly explain why the first algorithm is of exponential complexity and the second one is linear (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.

b) 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; write the corresponding pseudo code for that tail recursion algorithm and implement it in Java and repeat the same experiments as in part (a) above.
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.

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 1 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.**