Department Computer Science and Software Engineering
Concordia University

COMP 352: Data Structures and Algorithms
Fall 2019 - Assignment 1

Due date and time: Friday September 27, 2019 by midnight

<u>Written Questions (50 marks):</u> Please read carefully: You must submit the answers to <u>all</u> 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.

<u>Question 1</u>

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?

<u>Question 2</u>

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 $O(n^7 \log n)$
b) $10^7n^5 + 5n^4 + 9000000n^2 + n$ is $\Theta(n^3)$
c) n^n is $\Omega(n!)$
d) $0.01n^9 + 800000n^7$ is $\Theta(n^9)$
e) $n^8 + 0.0000001n^5$ is $\Omega(n^{13})$
f) $n!$ is $O(3^n)$

<u>Question 3</u>

a) What is the Big-O (*O(n)*) and Big-Omega (*Ω(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
```

<u>Programming Questions (50 marks):</u>

In class, we discussed 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 in pseudo code and implement in Java two **recursive** versions (linear and multiple) of Tetranacci calculators and experimentally compare their runtime performances. Tetranacci numbers are a more general version of Fibonacci numbers and start with four predetermined terms, each term afterwards being the sum of the preceding four terms. The first few Tetranacci numbers are:

0, 0, 0, 1, 1, 2, 4, 8, 15, 29, 56, 108, 208, 401, 773, 1490, …

For that, with each implemented version you will calculate Tetranacci(5), Tetranacci(10), etc. in increments of 5 up to Tetranacci (100) (or higher value if required for your timing measurement) and measure the corresponding run times (for instance, Tetranacci(10) returns 56). You can use Java's built-in time function for finding the execution time. You should redirect the output of each program to an *out.txt* file. You should write about your observations on 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.

b) 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.

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-recursion version of Tetranacci 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. See full details of submission details below.

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 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 and .html). 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 *studentID1* and *studentID2* are the student 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.