Department of Computer Science and Software Engineering
Concordia University
COMP 352: Data Structure and Algorithms
Fall 2017
Assignment 2
Due date and time: Monday October 23rd, 2017 by midnight

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

Question 1

a) Develop well-documented pseudo code that finds all the two elements of a given array that subtract exactly to | x | (absolute value of x). The code must display the indices and the values of these elements. For instance, given the following array (13, 1,-8, 21, 0,-9,-54, 17, 31, 81,-46) and *x* as 8, your code should find and display something similar to the following (notice that this is just an example. Your solution must not refer to this particular example):

All pairs of elements of the array that subtract exactly to absolute value of 8 are:
Indices 0 & 3 with values 13 & 21 (e.g., 13-21= | 8 |)
Indices 1 & 5 with values 1 & -9
Indices 2 & 4 with values -8 & 0
Indices 6 & 10 with values -54 & -46
Etc.

b) Briefly justify the motive(s) behind your design.

c) What is the Big-O complexity of 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.

Question 2

Given a non-sorted array *A* of *n* integers and an integer value x:

a) Similar to Question 1, develop well-documented pseudo code that finds all pairs of elements of the array that subtract to exactly to | x |. The code however must be different than the one you had in Question 1 and must use either a stack *S* or a queue *Q* to perform what is needed.

b) Briefly justify the motive(s) behind your design.

c) What is the Big-O complexity of 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.

e) What is the Big-O *space* complexity of the utilized stack or queue? Explain your answer.

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

Question 4

Consider the algorithm *DoSomething* below

Algorithm DoSomething (A, n)
Input: Array A of integer containing n elements
Output: Array M of integer containing n elements

```
for i=0 to n-1 do
  Zom[i]=0
end for
for i=0 to n-2 do
    for j=i+1 to n-1 do
      if A[i]≤A[j] then
          Zom [j]= Zom [j]+1
      else
          Zom[i]= Zom[i]+1
      end if
  end for
end for
for i=0 to n-1 do
    M[Zom [i]]= A[i]
end for
Return M
```

a) What is the big-O (*O(n)*) and big-Omega (*Ω(n)*) time complexity for algorithm *DoSomething* in terms of *n*? Show all necessary steps.
b) Trace (hand-run) *DoSomething* for an array A = (71,15,45,98,04,32). What is the resulting M?
c) What does *DoSomething* do? Explain that clearly and briefly given any arbitrary array A of n integers?
d) Can the runtime of *DoSomething* be improved easily? Explain how (i.e. re-write another solution(s) that does exactly what *DoSomething* is doing more efficiently)?
e) Can the space complexity of *DoSomething* be improved? Explain how?

Programming Questions (50 marks):

In this programming part you are asked to implement a game called *RightMagnetic Cave*.

RightMagnetic Cave is a 1-player game consisting of a row of squares of any sizes each containing an integer, like this:

8	16	10	4	6	10	2	12	8	0

The rules of the game are simple. The circle on the initial square is a marker that can move to other squares along the row. At each step in the game, you may move the marker the number of squares indicated by the integer in the square it currently occupies divided by 2 if the number is even, and moved (n/2+1) if the number is odd. The marker may move either left or right along the row but may not move past either end. For example, the only legal first move is to move the marker four squares (8/2) 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 magnetic cave, the "0" at the far end of the row. In this configuration, you can solve the game by making the following set of moves:

Starting position	8	16	10	4	6	10	2	12	8	0
Step 1: Move right	8	16	10	4	6	10	2	12	8	0
Step 2: Move right	8	16	10	4	6	10	2	12	8	0
Step 3: Move left	8	16	10	4	6	10	2	12	8	0
Step 4: Move right	8	16	10	4	6	10	2	12	8	0

Though the *RightMagnetic Cave* game is solvable, actually with more than one solution—some configurations of this form may be impossible, such as the following one:

10	16	4	6	2	10	0

In this configuration, you will bounce between the two 10's, but you cannot reach any other square.

Requirements:

1. In this programming assignment, you will design using pseudo code and implement in java code **two versions** of the *RightMagnetic* game**.**
   - The first version will be completely based on recursion.
   - The second one will be based on a stack, queue, list or vector.
2. Your solution takes a starting position of the marker along with the list of squares**.** Your solution should return *true* if it is possible to solve the game from the starting configuration and *false* if it is impossible. Your solution should also work for any size of the game's row, and a random value in each square. You may assume all the integers in the row are positive except for the last entry, the goal square. At the end of the game, the values of the elements in the list must be the same after calling your solution as they are beforehand, (that is, if you change them during processing, you need to change them back.).

   a) Briefly explain the time and space complexity for both versions of your game. You can write your answer in a separate file and submit it together with the other submissions.
   b) For the first version of your solution describe the type of recursion used in your implementation. Does the particular type of recursion have an impact on the time and space complexity? Justify your answer.
   c) For the second part of your solution, justify why you choose that particular data structure (e.g. why you choose a stack and not a queue, etc.)
   d) Provide test logs for <u>at least 20 different</u> game configurations, <u>sufficiently complete to show that your solution works for various row sizes and square values</u>.
   e) If possible, explain how one can detect unsolvable array configurations and whether there exists a way to speed up the execution time. Answering this question is optional and you can earn bonus marks by submitting a good solution.

   You are required to submit the two fully commented Java source files, the compiled files (.class files), and the text files.

   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.

Submission:

- The **<u>written questions part</u>** must be done **<u>individually</u>** and uploaded to "theory assignment 2" via EAS, name your file *A2_studentID. pdf*, where *studentID* is your ID number. For this second assignment, student 123456 would submit a file named A2_123456.pdf. All your answers to written questions must be in PDF (no scans of handwriting) or text formats only. **Please be concise and brief (about ¼ of a page for each question).**
- The **<u>programming part</u>** must be done **<u>in a team of 3 students max</u>,** For the Java programs, you must submit the source files together with the compiled files. The solutions to all the programming questions should be zipped together into one .zip or .tar.gz, name your file *A2_studentID1, studentID2, studentID3. zip,* where *studentID1, studentID2, studentID3, are the IDs of the three students who did the programming part together,* uploaded to "Programming assignment 2" via EAS.

Note: **Assignment not submitted by the due date and in the correct format will not be graded – NO EXCEPTIONS!!!!**