Department of Computer Science and Software Engineering
Concordia University
COMP 352: Data Structure and Algorithms
Fall 2016
Assignment 2
Due date and time: Friday October 21th, 2016 by midnight

Written Questions (50 marks):

Question 1

a) Develop well-documented pseudo code that finds all the two elements (non-negative numbers) of a given array that modulo up exactly to x. The code must display the indices and the values of these elements. For instance, given the following array (123,73,39,12,14,9,113,93,203,22,25,10) and *x* as 3, 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 modulo up to a value of 3 are:
> Indices 0 & 3 with values 123 & 12 (e.g., 123 % 12 returns 3)
> Indices 1 & 5 with values 73 & 14
> Indices 2 & 4 with values 39 & 9
> Indices 6 & 11 with values 113 & 22
> 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 modulo up 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) $f(n) = 7n \log n + n^2$; $g(n) = \log n$.
ii) $f(n) = \log n^2$; $g(n) = (\log n)^2$.
iii) $f(n) = \log n^2 + n^3$; $g(n) = \log n + 5$.
iv) $f(n) = n\sqrt{n} + \log n$; $g(n) = \log n^2$.
v) $f(n) = 2^n + 10^n$; $g(n) = 10n^2$.
vi) $f(n) = n!$; $g(n) = n \log n$.
vii) $f(n) = \log^2 n$; $g(n) = \log n$.
viii) $f(n) = n$; $g(n) = \log^2 n$.
ix) $f(n) = \sqrt{n}$; $g(n) = \log 10$.
x) $f(n) = 2^n$; $g(n) = 3^n$.
xi) $f(n) = 2^n$; $g(n) = n^n$.

Question 4

Consider the algorithm *MySolution* below

Algorithm MySolution (A, n)
Input: Array A of integer containing n elements
Output: Array S of integer containing n elements

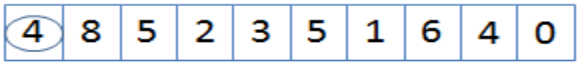
```
1.  for i=0 to n-1 do
2.    Var[i]=0
3.  end for
4.  for i=0 to n-2 do
5.    for j=i+1 to n-1 do
6.      if A[i]iA[j] then
7.        Var[j]= Var[j]+1
8.      else
9.        Var[i]= Var[i]+1
10.     end if
11.   end for
12. end for
13. for i=0 to n-1 do
14.   S[Var[i]]= A[i]
15. end for
16. Return S
```

a) What is the big-O (*O(n)*) and big-Omega (*Ω(n)*) time complexity for algorithm *MySolution* in terms of *n*? Show all necessary steps.
b) Trace (hand-run) *MySolution* for an array A = (60,35,81,98,14,47). What is the resulting A?
c) What does *MySolution* do? Explain that clearly and briefly given any arbitrary array A of n integers?
d) Can the runtime of *MySolution* be improved easily? Explain how (i.e. re-write another solution(s) that does exactly what *MySolution* is doing more efficiently)?
e) Can the space complexity of *MySolution* be improved? Explain how?

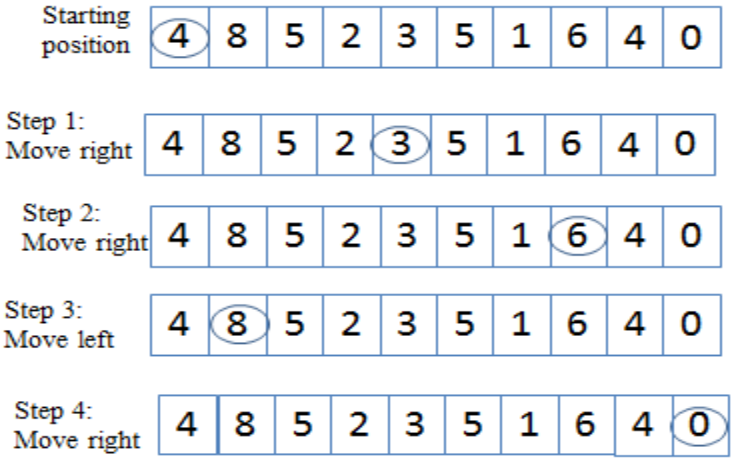
Programming Questions (50 marks):

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

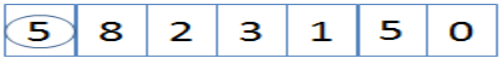
RightWing Cave is a 1-player game consisting of a row of squares of any size each containing an integer, like this:



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. 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 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 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:



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



In this configuration, you will bounce between the two 5'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 *RightWing Cave* 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 also should 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 memory 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 memory 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 at least 20 different game configurations, sufficiently complete to show that your solution works for various row sizes and square values.
   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.

The written part must be done individually and the programming part must be done in a team of 2 students. Submit all your answers to written questions in PDF (no scans of handwriting) or text formats only. Please be concise and brief (about ¼ of a page for each question) in your answers. For the Java programs, you must submit the source files together with the compiled executables. The solutions to all the questions should be zipped together into one .zip or .tar.gz file and submitted via EAS. You may upload at most one file to EAS.

For the programming component, you must make sure that you upload the assignment to the correct directory of Assignment 2 using EAS. Assignments uploaded to the wrong directory will be discarded and no resubmission will be allowed.