Concordia University
Engineering and Computer Science

COMP 352: Data Structures and Algorithms
Winter 2021 - Assignment 2
Due date and time: Sunday, March 7, 2021 by 23:55

1 Theory 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

Consider the following algorithm:

Algorithm 1: Mystery(A, n)

Input : Array A of n numbers $a_0, \cdots a_{n-1}$ with $n \geq 1$
Output: See question (a) below

```
1  if n = 1 then
2      Print the message "Mystery solved"
3  else
4      j ← 1
5      while j ≤ n − 1 do
6          if a_{j−1} > a_j then
7              swap a_{j−1} and a_j
8          Print n, j, and the array elements
9          j ← j + 1
10     Mystery(A, n-1)
```

(a) Using A = (9,5,11,3,2) as the input array, hand-trace Mystery, showing the output.

(b) Determine the running time $T(n)$ as a function of the number of comparisons made on line 6, and then indicate its time and space efficiency (i.e., $\mathcal{O}$ and Ω bounds).

(c) Determine the running time $T(n)$ as a function of the number of swaps made on line 7, and indicate its time and space efficiency.

(d) Suggest an improvement, or a better algorithm altogether, and indicate its time and space efficiency. If you cannot do it, explain why it cannot be done?

(e) Can Mystery be converted into an iterative algorithm? If it cannot be done, explain why; otherwise, do it and determine its running time complexity in terms of the number of comparisons of the array elements.

Question 2

For each of the following pairs of functions, either $f(n)$ is $\mathcal{O}(g(n))$, $f(n)$ is $\Omega(g(n))$, or $f(n)$ is $\Theta(g(n))$. For each pair, determine which relationship is correct. Justify your answer.

#	$f(n)$	$g(n)$
(a)	$4n \log n + n^2$	$\log n$
(b)	$8 \log n^2$	$(\log n)^2$
(c)	$\log n^2 + n^3$	$\log n + 3$
(d)	$n\sqrt{n} + \log n$	$\log n^2$
(e)	$2^n + 10^n$	$10n^2$
(g)	$\log^2 n$	$\log n$
(h)	n	$\log^2 n$
(i)	$\sqrt{n}$	$\log n$
(j)	4^n	5^n
(k)	3^n	n^n

2 Programming Problem (50 marks)

The researchers at a medical center wish they had taken a computer programming course way back when they were at medical school. These days, in addition to attending their research, they find themselves entangled with having to learn the R and Python languages in order to wrangle, visualize and model data collected from patients.[1,2] They simply have no time and no patience for R or Python! So they have decided to hire you as a programmer to assist, providing you with a bit of background information and a clear description of your task.

The researchers have discovered that, when certain cells are introduced into a new environment, the cells quickly form colonies. To study the effectiveness of a drug they are developing, they need to figure out the size and number of the colonies in the environment. Fortunately, they can map the environment onto a grid of **0**s and **1**s that might, for example, look something like this:

0	0	0	1	1	0	1	1	0	0	0	1	1	1	0	1	1	1	0	1
1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	1	1	1	1
0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1
1	1	1	0	0	1	0	1	0	0	0	0	1	0	1	1	0	1	1	0
0	1	1	1	0	1	1	1	0	1	0	0	1	0	1	0	1	1	0	1

The locations on the grid each have a row and a column coordinates. A **1** or **0** at a location indicates, respectively, the presence or absence of a cell at that location. Colonies are formed by neighboring cells. Two cells are considered neighbors, if their locations are adjacent either horizontally, vertically, or diagonally. Thus, the maximum number of neighbors a cell can have is 3 for a corner cell, 5 for a cell on a single border, and 8 for a cell not on the borders. Given this background information, you are tasked with developing a `ColonyExplorer` program that explores and labels the colonies (if any). For example, using the grid above, your `ColonyExplorer` should produce the following information:

-	-	-	A	A	-	A	A	-	-	-	B	B	B	-	C	C	C	-	C
D	-	-	-	-	A	-	-	-	A	-	-	-	-	-	C	C	C	C	C
-	D	-	-	A	-	-	-	A	-	A	-	-	-	-	-	-	C	C	C
D	D	D	-	-	A	-	A	-	-	-	-	E	-	C	C	-	C	C	-
-	D	D	D	-	A	A	A	-	F	-	-	E	-	C	-	C	C	-	C

```
A: 14
B: 3
C: 20
D: 8
E: 2
F: 1
```

The information shows that there are six colonies labeled A, B, C, D, E, and F on the grid with size 14, 3, 20, 8, 2, and 1, respectively.

[1] Why being a programmer will make me a better doctor
[2] 5 Reasons Some Doctors are Learning to Code

2.1 Requirements

(a) In this programming assignment, you will be developing your `ColonyExplorer` program (i.e., class) in two versions.

Both versions must implement an algorithm named `ExploreAndLabelColony`.

In version 1 of `ColonyExplorer`, the `ExploreAndLabelColony` algorithm is recursive.

In version 2 of `ColonyExplorer`, the `ExploreAndLabelColony` algorithm is non-recursive. This version is allowed to use a linear data structure such as a stack, queue, list, or vector.

Both versions are required to be designed in pseudo code and implemented in Java.

(b) Starting from a given location on the grid, `ExploreAndLabelColony` explores the grid, expanding and labeling all of the cells in a single colony originating from the starting location. Taking as input a grid, the coordinates of a starting location, and a label, `ExploreAndLabelColony` will either label that location and all its neighbors or do nothing, depending on the presence or absence of a cell at the given location, respectively; either way, `ExploreAndLabelColony` is required to return the size of the labeled colony.

(c) `ColonyExplorer` is responsible for ensuring that `ExploreAndLabelColony` is called starting at every location on the grid storing a **1**. The reason is that `ExploreAndLabelColony` locates only a single colony.

For example, using the first grid above and calling `ExploreAndLabelColony` for the first time on the location at the top row and forth column will modify the grid only at the locations labeled A, leaving all the other locations intact.

Once every grid location storing a **1** has been colonized, `ColonyExplorer` freezes the grid by replacing the **0**s on the grid with –s (dashes) .

(d) `ColonyExplorer` is responsible for creating an initial grid of random number of rows and columns in the range [5-20], and for filling the grid randomly with **0**s and **1**s.

(e) `ColonyExplorer` is responsible for generating the labels for `ExploreAndLabelColony` to use during exploration. Use the alphabet letters $A \cdots Z$ and $a \cdots z$ as labels (in that order). For simplicity, cycle through and reuse the same labels if necessary.

(f) Briefly explain the time and memory complexity for both versions of `ExploreAndLabelColony`. Write your answers in a separate file and submit it together with the other programming submissions.

For version 1, describe the type of recursion used in your implementation and its time and memory complexity? Justify your answer. Also explain whether a tail-recursive version is possible. If yes, you can earn bonus marks by submitting such a version.

For version 2, justify your choice of the particular data structure and why you chose it over the other available structures (e.g., why you chose a stack and not a queue, etc.). Explain whether your choice has an impact on the time and memory complexity.

(g) For each version provide test logs for at least 20 different initial grid configurations.

3 Deliverables

In this assignment, the programming question can be done in a team of two, if you wish. However, the theory questions must be completed and submitted individually.

3.1 Answers to Theory Questions

For all questions, including the programming questions, the parts that require written answers, pseudo-code, graphs, etc., you can express your answers into any number of files of any format, including image files, plain text, hand-writing, Word, Excel, PowerPoint, etc.

However, no matter what program you are using, you must convert each and every file into a PDF before submitting. This is to ensure the markers and you have exact same view of your work regardless of the original file formats.

Whether or not you are in a team, you must each submit a copy of your written answers.

3.2 Solutions to Programming Question

Developing your programming work using a Java IDE such as NetBeans, Eclipse, etc., you are required to submit the project folder(s) created by your Java IDE, together with any input files used and output files produced by your program(s).

If you are in a team, you must to submit only ONE copy of your program project folders(s) and input/output files.

3.3 What and How to Submit

1. Letting the # character denote the assignment number, create a folder as follows:
   (a) If you are working individually, name your folder `A#_studentID`, where `studentID` denotes your student ID. Then, copy the PDF files you prepared for the theory questions together with your project folder(s) and input/output files to the `A#_studentID` folder.
   (b) If you are working in a team, name your folder `A#_studentID1_studentID2`, where `studentID1` denotes the ID number of the student who is responsible for submitting the programming solutions for both of you, and `studentID2` denotes the ID number of the other team member. Then, the student whose ID number is `studentID1` must follow item (a), as if she/he completed the programming question individually. The other team member must also follow item (a) above, but she/he must NOT include the team's programming solutions.
2. Compress the folder you created in (1) into a zip file with the same name as that of the folder being compressed.
3. Upload the compressed file you created in item (2) to Moodle.

3.4 Last but not Least

To receive credit for your programming solutions, you must demo your program to your marker, who will schedule the date and time for the demo for you (please refer to the course outline for full details). If working in a team, both members of the team must be present during the demo. Please notice that failing to demo your programming solution will result in zero mark regardless of your submission.