Concordia University
Department of Computer Science and Software Engineering
Comp 352
Data Structure and Algorithms
Assignment 1 – Summer 2021
Due date: Friday May 21st, 2021 by midnight

Heads-Up

- The programming questions can be done individually or in a team of two members max. However, the written questions must be completed and submitted individually.
- For the written questions, you must submit the answers to all the questions. However, only one or more questions, possibly chosen at random, will be corrected and will be evaluated to the full 50 marks.
- For the programming questions you are not allowed to use any java build-in classes/methods/algorithm unless indicated.

1. Written Questions (50 marks)

Question 1

a) Given an array A of integers of any size, $n \geq 0$, write an algorithm as a pseudo code (not a program!) that would find out the total number and their sum of even positive numbers as well as the total number and their sum of odd positive numbers in the array. For instance, assume that array A is as follows:

5	-2	4	9	-250	99	108	-62	46	13

The algorithm would find the number of positive even numbers is 3 and their sum is 4+108+46=158 and the number of positive odd numbers is 4 and their sum is 5+9+99+13=126. Your algorithm must handle possible special cases. Finally, the 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?

Question 2

Prove or disprove the following statements, using the relationship among typical growth-rate functions discussed in class:

a) $45 n^2 + 28 n + 752$ is $\Omega(n)$

b) $256 n + 8 n \log n$ is $\Theta(\log n)$

c) $n^{0.8} + \log n$ is $O(\log n)$

d) $2\,n^2 \log n + n^3$ is $\Theta(\log n)$

e) $4\,n \log^2 n + 3\,n^2 \log n$ is $O(\log n)$

f) $n^7 + 0.00000001n^6$ is $\Omega(n^6)$

Question 3

Consider the following algorithm:

```
Algorithm arraySpecialSum(A, n)

    currentMax ← A[0]
    for i ← 1 to n − 1 do
        if A[i] > currentMax then
            currentMax ← A[i]
        { increment counter i }

    CurrentMaxOccurence = 0
    for i ← 0 to n − 1 do
        if A[i] == currentMax then
            { increment currentMaxOccurence}
        { increment counter i }

    specialSum = 0
    for i ← 0 to n − 1 do
        for j ← 1 to currentMaxOccurence do
            { specialSum = specialSum + A[i] }
            { increment counter j }
        { increment counter i }

    return specialSum
```

a) Use the analysis from the course notes to determine a time complexity function f(n) for the above algorithm. Your answer must be simple and concise.

b) What is the time complexity of this algorithm, in terms of Big-O?

c) What is the space complexity of this algorithm, in terms of Big-O?

d) Can we improve this algorithm to achieve a better time complexity and still return the same `specialSum` value?

i. If yes:
   - give a new algorithm that achieves it.
   - provide a time complexity function f(n) for the modified algorithm.
   - provide the time complexity of the modified algorithm, in terms of Big-O.
   - provide your observations regarding the two algorithms with respect to time complexity.

ii. If no, provide explanations as to why it cannot be improved.

Programming Questions (50 marks)

We received a list of people that are going to participate in a clinical test. The list includes the name and date of birth of the participants which is stored in two arrays: *pName* and *pDOB*. The two arrays are of the same size. The arrays' size N is the number of participants. pName[0] hold the name of the first participant and pDOB[0] hold the date of birth of the first participant; pName[N-1] hold the name of the last participant and pDOB[N-1] hold the date of birth of the last participant. The participants' information is stored in the arrays at random. We need to separate the participants in two groups: seniors and non-seniors. A person is considered senior if she/he is 65 years of age or older. You need to provide an algorithm that takes as input the two arrays and the number of participants. Your algorithm should rearrange the participants' information in the arrays in the following manner:

- Array position zero hold the information of the oldest person in the participants.
- Array position N hold the information of the oldest non-senior participant.
- Senior participants should be stored in a decreasing order based on their age.
- Non senior participants should be stored in an increasing order based on their age.

A sample of two input arrays of size 10 and their contents after rearrangements by the algorithm is shown below:

	Algorithm inputs		Algorithm outputs	
Index	*pName*	*pDOB*	*pName*	*pDOB*
0	Linda	1-1-2003	Sam	24-2-1940
1	Sam	24-2-1940	Maria	9-5-1941
2	Roger	11-12-1995	Melissa	25-7-1945
3	Alfred	31-3-1980	Roberto	29-6-1950
4	Roberto	29-6-1950	Thomas	20-7-2004
5	Melissa	25-7-1945	Linda	1-1-2003
6	Brian	15-7-2002	Brian	15-7-2002
7	Thomas	20-7-2004	Roger	11-12-1995
8	Leslie	27-4-1990	Leslie	27-4-1990
9	Maria	9-5-1941	Alfred	31-3-1980

a) In this programming assignment, you will design an algorithm (in pseudo code), and implement (in Java), four functions as follows:
   1) *`rearrangeParticipants`*: a recursive function that take as input two arrays pName and pDOB and the number of paticipants, and returns the number of senior participants in addition to arranging the arrays as specified above.
   2) *displaySeniorsIncreasingOrder*: a recursive function that takes as input the two arrays pName and pDOB and the number of senior participants and display the name and DOB of senior participants in an increasing order based on their age.
   3) *displayNonSeniorsInreasingOrder*: a recursive function that takes as input the two arrays pName and pDOB, the number of non-senior participants and the total number of participants and display the name and DOB of non-senior participants in an increasing order based on their age.
   4) *`displayIncreasingOrder`*: this function takes as input the two arrays pName and pDOB, the number of senior participants and the total number of participants and display all participants in an increasing order based on their age.

   All your algorithms must handle possible special cases. For each implemented function you need to compare the runtime performance and measure the corresponding run times. You can use Java built-in time function for this purpose.

You can generate a file with random data and experiment your implementation against different size of participants' list size N in `{10, 100, 1000, 10000, 100000…,1000,000}`. You should redirect the output of each set of test size 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 fully commented Java source files, the compiled files, and the text files.

For each implemented function:

- Use the analysis from the course notes to determine a time complexity function f(n). Your answer must be simple and concise.
- What is the time complexity of the function in terms of Big-O?

b) For *`rearrangeParticipants`* recursive algorithm:

- Briefly explain whether your algorithm is linear or not.
- Does your algorithm use tail recursion? Why or why not? Explain your answer.
  If your answer is "No" then:
  can a tail-recursive version for this algorithm 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 not feasible.

You will need to submit both the <u>pseudo code</u> and <u>the Java program</u>, together with your experimental results. <u>Keep in mind that Java code is not pseudo code. See full details of submission details below</u>.

Guidelines

For all questions, including the programming questions, the parts that require written answers, pseudo-code, analysis, table, 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.

1. Written Questions

You must submit a copy of your written answers as A#studentID under A#1 WrittenQuestions submission on Moodle, (studentID denotes your student ID, and A# is A1 for this first assignment).

2. Programming Questions

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

You must submit a copy of your programming questions as a zip file that includes the sources java files, input/output files, written pseudo code, etc., under A#1 ProgrammingQuestions submission on Moodle,

a) If you are working individually, name your zip file A#studentID, where studentID denotes your student ID, and A# is A1 for this first assignment.
b) If you are in a team, you must submit only ONE copy of your programming part, name your zip file A#studentID1_studentID2, where studentID1 denotes the student ID of the member responsible for submitting the programming solutions for both students.

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.