Department of Computer Science and Software Engineering
Concordia University

COMP 352: Data Structure and Algorithms

Fall 2018

Assignment 2

Due date and time: Monday October 22nd, 2018 by midnight

Written Questions (50 marks):

Question 1

1) Develop well-documented pseudo code that finds all the elements of a given array (of any size *n)* that are multiple of *x*. The code must display the indices and the values of these elements. For instance, given an array *A* :

   (22, 61,-10, 21, 0, 9, 50, 17, 35, 81,-46, 19, 5, 77) with *x* as 5, 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):

   The elements of the array *A* that are multiple of 5 are:
   Index 2 with value -10
   Index 6 with value 50
   Index 8 with value 35
   Index 12 with value 5

   a. Briefly justify the motive(s) behind your design.
   b. What is the Big-O complexity of your solution? Explain clearly how you obtained such complexity.
   c. What is the Big-Ω complexity of your solution? Explain clearly how you obtained such complexity.
   d. What is the Big-O *space* complexity of your solution?

2) Develop a well-documented pseudo that solves the problem stated in part 1), using either a stack *S* or a queue *Q* to perform what is needed.
   a. Briefly justify the motive(s) behind your design.
   b. What is the Big-O complexity of your solution? Explain clearly how you obtained such complexity.
   c. What is the Big-Ω complexity of your solution? Explain clearly how you obtained such complexity.
   d. What is the Big-O *space* complexity of the utilized stack or queue? Explain your answer.

Question 2

Consider the algorithm *MySolution* below:

```
Algorithm MySolution (A, n)
 Input: Array A of integer containing n elements
 Output:  Array B of integer containing n elements
1.  for i=0 to n-1 do
2.     Res[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]≤A[j] then
7.             Res [j]= Res [j]+1
8.         else
9.             Res[i]= Res[i]+1
10.        end if
11.    end for
12. end for
13. for i=0 to n-1 do
14.     B[Res [i]]= A[i]
15. end for
16. Return B
```

a) What is the big-O (*O( )*) and big-Omega (*Ω( )*) time complexity for algorithm *MySolution* in terms of *n*? Show all necessary steps.
b) Trace (hand-run) *MySolution* for an array A = (88, 12, 94, 17, 2, 36, 69). What is the resulting B?
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?

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)	$f(n) = \log^3 n$;	$g(n) = \sqrt{n}$.
ii)	$f(n) = n\sqrt{n} + \log n$;	$g(n)=\log n^2$.
iii)	$f(n) = n$;	$g(n) = \log^2 n$.
iv)	$f(n) = \sqrt{n}$;	$g(n) = 2^{\sqrt{\log n}}$.
v)	$f(n) = 2^{n!}$;	$g(n) = 3^n$.
vi)	$f(n) = 2^{10n}$;	$g(n) = n^n$.
vii)	$f(n) = (n^n)^5$;	$g(n) = n^{(n^5)}$.

Note: You must submit the answers to all the questions above. However, only one or more questions, possibly chosen at random, will be corrected and will be evaluated to the full 50 marks.

Programming Questions (50 marks):

In these programming questions you will evaluate two implementations of *List* interface in terms of their performance for different operations.

List[1] interface is an ordered collection (also known as a *sequence*). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.
The List interface provides four methods for index **access** to list elements, two methods to **search** for a specific object, and two methods to efficiently **insert** and **remove** multiple elements at an arbitrary location in the list. Note that the speed of these operations may depend on the implementation (e.g. Array or LinkedList).

You are required to write two implementations of *`List`* interface, one that uses array, and one that uses doubly-linked list. Then, you will have to test the performance of several operations when using your implementations.

Question 1:

Implement the following methods in the two implementations (called *`MyArrayList`* and `MyLinkedList`) of `List` interface:

```
Boolean   add(E e)               // Appends the specified element to the end of this list
void add(int index,E element)    // Inserts the specified element at the specified position in this list
void  clear()                    // Removes all of the elements from this list
E  remove(int index)             // Removes the element at the specified position in this list
Boolean remove(Object o)         // Removes the first occurrence of the specified element from this list
String  toString()               // Returns a string representation of this list
int size()                       // Returns the number of elements in this list
```

Define your classes to be generics. The array implementation should have dynamic resizing (double the size when growing and halve the size when less than 25 % of the capacity is used); and the linked list implementation should use doubly linked list. Also, the behavior of these methods should be equivalent to that of Java Standard Library's classes `ArrayList` or `LinkedList`. Please refer to the corresponding descriptions online[2]/[3].

For the rest of the methods of the `List` interface, you may just throw an exception:

```
      public type someUnneededMethod() {
             throw new UnsupportedOperationException();
      }
```

Question 2:

Write your driver class `ListTester`. Use both of your list implementations and compare them to the corresponding Java library implementations (*`ArrayList`* and *`LinkedList`*)
For numbers *N* = {10, 100, 1000, 10000, 100000, 1000000}

a) Starting with *empty* lists of Number-s; measure how long it takes to insert N integer numbers (`int`, or `Integer`) with random values ranging from 0 to 2N into the lists, when inserting them at the *beginning*, at the *end*, and into a *random location* of the list (use indices to indicate where to do the insertion (e.g., `list.add (randomLocation, number)`).

[1] https://docs.oracle.com/javase/7/docs/api/java/util/List.html
[2] https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html
[3] https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

b) Starting with *non-empty* lists of N items (e.g., from part *a*), measure how long it takes to remove N numbers from the lists when removing them from the beginning, from the end, and from a random location of the list (use indices to indicate the location).
c) Starting with *non-empty* lists of N items (same as part *b*), measure how long it takes to remove N random numbers (with values between 0 and 2N) from the four lists (some values might not exist in the list!).

- ➢ Produce the following table (the timing values below are just an illustration and do not relate to any real measurements):

N = 10	**Insert@**start(ms)	**Insert@**end (ms)	**Insert@**random (ms)
MyArrayList	15	201	603
ArrayList	15	201	603
MyLinkedList	15	201	603
LinkedList	15	201	603

N = 10	**Remove@**start(ms)	**Remove@**end(ms)	**Remove@**random(ms)	**Remove** byvalue (ms)
MyArrayList	15	201	603	121212
ArrayList	15	201	603	121212
MyLinkedList	15	201	603	121212
LinkedList	15	201	603	121212

- ➢ Repeat for all values of N = 100; N = 1000; ….etc.

- Save the result of your program execution in a file `testrun.txt` and submit it together with your other files.

Important Requirements:

1. Make sure you reset the timer (or save the intermediate time before the next measurement); i.e., make sure your measured time contains only the time to perform one set of operations that was supposed to be timed.
2. In case the operations for big N numbers take too long (e.g., more than 50s) you may reduce the number to a smaller one or eliminate it (so that you will have a range from, say, 1 to 100000).
3. Do not use any java abstract data type or packages when writing `MyArraylist` and `MyLinkedList` in these programming questions of your assignment.
4. Your code should handle boundary cases and error conditions. It is also imperative that you test your classes.
5. For these programming questions, you are required to submit the commented Java source files, the compiled files (.class files), and the test run text files.

Submission Guidelines

- 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 part, submit all your answers in PDF. You need to be concise and brief for each question. Submit the theory part of the assignment under Theory_A#2 Dropbox on Moodle or "Theory assignment 2" via EAS.
- For the programming part: you must submit the Java programs as the source files together with the compiled files as well as any required associated files. The solutions to all the questions should be zipped together into one .zip file and submitted via Moodle under Programming_A#2 Dropbox for Moodle submission, or "Programming Assignment 2" via EAS. You must upload at most one file. That is, if you are working in a group of 2 students, only one student will submit the programming part. Do not upload 2 copies.
  - o Create **one** zip file, containing the necessary files (.java, .html, etc.). 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.

Important Notes

- The assignment must be submitted in the right DropBox for Moodle submission and right Folder for EAS submission. **Assignments uploaded to an incorrect DropBox/Folder will result in a zero mark. No resubmissions will be allowed.**
- For the programming part of the assignment, a demo is required (please refer to the course 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.