Concordia University
Department of Computer Science and Software Engineering
Comp 352
Data Structure and Algorithms
Assignment 2 – Summer 2021
Due date: Monday June 7th, 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 a couple of questions 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

Consider the algorithm *DoSomething* below:

```
Algorithm DoSomething (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]≤A[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 *DoSomething* in terms of *n*? Show all necessary steps.
b) Trace (hand-run) *DoSomething* for an array A = (60,35,81,98,14,47). What is the resulting A?
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?

Question 2

A company is involved in shipping large number of containers at a dock. Containers are classified into three categories:
Category 1: containers need to be accessed by their indices: *lookup*, *set*, *add* and *remove* from Category 1.
Category 2: containers need to be *add*ed and *removed* before or after certain position including the first and last position of the Category 2.
Category 3: containers need to be *added* and *removed* in a sorted alphabetical order of their destination and any new added container need to follow that order of the Category 3.

From the three linear ADTs: List, Positional and Sequence covered in class. discuss which ADT to choose and its underlying implementation for the above containers' categories.

Question 3

a) Draw a single binary tree that gave the following traversals:

Inorder: C O P Y R I G H T A B L E
Postorder: C P O R G I T H B A E L Y

b) Assume that the binary tree from the above- part (a)- is stored in an array-list as a complete binary tree as discussed in class. Specify the contents of such an array-list for this tree.

Question 4

a) Draw the min-heap that results from the bottom-up construction algorithm on the following list of values:
10,12,13,15,4,16,8,9,2, 20,7,14,6,23,19

Starting from the bottom layer, use the values from left to right as specified above. Show immediate steps and the final tree representing the min-heap.

b) Perform the operation `removeMin` two times on the heap you created in part (a) and show the resulting min-heap after each step and the final tree representing the min-heap.

Question 5

a) Given a tree T, where *n* is the number of nodes of *T*.
Give an algorithm for computing the depths of all the nodes of a tree T. What is the complexity of your algorithm in terms of Big-O?
b) We say that a node in a binary search tree is full if it has both a left and a right child.
Write an algorithm called *Count-Full-Nodes(t)* that takes a binary search tree rooted at node t, and returns the number of full nodes in the tree. What is the complexity of your solution?

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	18	110	310
ArrayList	18	110	310
MyLinkedList	18	110	310
LinkedList	18	110	310

N = 10	**Remove@**start(ms)	**Remove@**end(ms)	**Remove@**random(ms)	**Remove** byvalue (ms)
MyArrayList	10	150	200	1234567
ArrayList	10	150	200	1234567
MyLinkedList	10	150	200	1234567
LinkedList	10	150	200	1234567

- 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.

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#2 WrittenQuestions submission on Moodle, (studentID denotes your student ID, and A# is A2 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#2 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 A2 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.