COSC2658 Data Structures & Algorithms
COSC2469 Algorithms and Analysis
Semester 3, 2023

RMIT UNIVERSITY

TEST 1 (20%) – TEST QUESTIONS

Full Name:
Student ID:

Test Duration: 120 mins (+ 10 mins for submission)

Deliverables: Exactly 2 Java files for problems 1 and 2; a plain text file for problem 3. Compress all files into a single .zip file (**your_first_name_Test1.zip**) and submit the zip file to Canvas.

Note: For every custom class you are asked to implement, use <your_first_name> as the prefix of the class name. For example, if you are asked to create a class Student, and your first name is Ling, you must define your class as LingStudent (no space in between your first name and the class name).

Problem 1

You are given a string **s** containing only the parentheses **(** and **)**. String **s** is valid if any opening left parenthesis **(** has a corresponding closing right parenthesis **)**. For example, the following are valid strings: **()**, **(())**, **(()())**. The following are invalid strings:

- **(** : Missing closing parenthesis.
- **(()(()))** : Extra closing parenthesis at the end.
- **)))** : Missing opening parentheses.

Create a Java class named **`StringAnalyser`** **(remember to add your first name as the class name prefix)** with a constructor that takes a string **s** as a parameter. Implement the following three public methods in the class. You can add additional private methods as needed. Assume N is the number of characters representing the parentheses. What are the Big-Os of the three methods regarding N in the worst case? You must insert a comment before the method signature to describe the method's Big-O. A sample template is given below:

```
// method1 complexity = O(N * log N)
public void method1() { ... }
```

(a) **`boolean isValid()`**: returns **true** if the string **s** is valid according to the rules mentioned above; otherwise, returns **false**.

(b) **`int totalPar()`:** returns the total number of parentheses in the string **s**. Only call this method when the string **s** is valid.

(c) **`int totalInsert()`:** returns the minimum number of insertions needed to make string **s** balanced. Only call this method when the string **s** is invalid. For example,

- **(** : 1 insertion needed.
- **(()(()))** : 1 insertion needed.
- **)))** : 3 insertions needed.

Create a **StringAnalyser** object and use it in the main method (client code). Your client code must at least call **`isValid()`** for the following strings: **(()()), (, (()(()))),)))**. Your code for this problem must be stored in a single file <your_first_name>StringAnalyser.java.

Problem 2

A singly linked list consists of a series of positive integers separated by -1. The first node and the last node of the linked list will have the values of -1 as well. An example of the linked list is depicted as follows:

head -> -1 -> 2 -> 4 -> 3 -> -1 -> 9 -> 10 -> 8 -> 4 -> -1

Create a Java class named **`LLAnalyser`(remember to add your first name as the class name prefix).** Implement the following three public methods in the class. You can add additional private methods as needed. What are the Big-Os of the three methods regarding input size N in the worst case? You must insert a comment before the method signature to describe the method's Big-O. A sample template is given below:

```
// method1 complexity = O(N * log N)
public void method1() { ... }
```

(a) **`void traverseLL(Node head)`:** traverses the linked list and prints the node data, excluding the node with -1.

(b) **`int largest(Node node)`**: returns the largest value of all the nodes lying in between the two consecutive -1's.

(c) **`Node resultLL()`:** returns the new linked list that contains the largest value of all nodes lying in between the two consecutive -1's. The new linked list should not contain any -1's. For example, using the example linked list above, the new linked list should be as follows: **head -> 4 -> 10**, where 4 is the largest value of the first group of nodes, and 10 is the largest value of the second group of nodes.

Create a **LLAnalyser** object and use it in the main method (client code). Your client code must test the linked list example given. You have to call your **traverseLL** method to traverse your initial linked list and the new linked list. Your code for this problem must be stored in a single file <your_first_name>LLAnalyser.java.

Problem 3

An array A consists of n integers A[0], A[1], … A[n-1]. You would like to output array A to a two-dimensional n-by-n array B in which B[x][y] contains the sum of array entries A[x] through A[y] and x < y, i.e. the sum A[x] + A[x+1] + … + A[y]. For x >= y, it doesn't matter what is the output for these values. A simple algorithm to solve this problem is shown below.

```
for x = 0 to n-1
      for y = x+1 to n-1
            sum up array entries A[x] through A[y]
            store the result in B[x][y]
      end for
end for
```

(a) What is the time complexity of the above algorithm? Provide your analysis.

(b) Although the algorithm you analyzed in part (a) is the most natural way to solve the problem—after all, it just iterates through the relevant entries of the array B, filling in a value for each—it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time, and analyze its time complexity.

--- END OF PAPER ---