Department Computer Science and Software Engineering
Concordia University

COMP 352: Data Structures and Algorithms Winter 2020 - Assignment 1

Due date and time: Friday January 31, 2020 by midnight

<u>Written Questions (50 marks):</u> Please read carefully: You must submit the answers to <u>all</u> 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.

<u>Question 1</u>

Given a string of random length and random contents of characters, that do not include digits (0-9), write an algorithm, **using pseudo code** that will shorten the representation of that string by adding the number of consecutive characters. For instance, given *str* as "`gggN@@@@@KKeeeejjdsmmu`" the algorithm should return the following representation of the string: "`g3N@5K2e4j2dsm2u`".

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

<u>Question 2</u>

i) Develop a well-documented pseudo code that finds the two consecutive elements in the array with the smallest difference in between, and the two consecutive elements with the biggest difference in between. The code must display the values and the indices of these elements. For instance, given the following array [20, 52,400, 3, 30, 70, 72, 47, 28, 38, 41, 53, 20] 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 two conductive indices with smallest difference between their values are: index 5 and index 6, storing values 70 and 73.
> The two conductive indices with largest difference between their values are: index 2 and index 3, storing values 400 and 3.

In case of multiple occurrences of the smallest or largest differences, the code should display the first found occurrence.

ii) Briefly justify the motive(s) behind your design.

iii) What is the time complexity of your solution? You must specify such complexity using the Big-O notation. Explain clearly how you obtained such complexity.

iv) What is the maximum size of stack growth of your algorithm? Explain clearly.

Question 3

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

a) $n^{15}\log n + n^9$ is $O(n^9 \log n)$
b) $15^7 n^5 + 5n^4 + 8000000n^2 + n$ is $\Theta(n^3)$
c) n^n is $\Omega(n!)$
d) $0.01n^9 + 800000n^7$ is $O(n^9)$
e) $n^{14} + 0.0000001n^5$ is $\Omega(n^{13})$
f) $n!$ is $O(3^n)$

Programming Questions (50 marks):

In this programming assignment, you will design in pseudo code and implement in Java two versions of a program that deals with two strings, referred to as *shortStr* and *longStr*. The program attempts to find if any/all possible permutation of shortStr exists within longStr, and if so, at which location on longStr.

Version 1:

In your first version, you must write a **recursive** method called ***permu***, which accepts shortStr (and any other needed parameters; i.e. start index and end index) and generates ALL possible permutation of that string. For each possible permutation value, the method should find out (possibly by calling another method), whether that permutation value is a substring of longStr. Your code must show all permutations and for each of them should indicate whether or not this value exists in longStr.

For example; given longStr as "`hhhlajkjgabckkkkcbakkdfjknbbca`",and shortStr as "`abc`", the method should show display something like:

```
abc
Found one match: abc is in hhhlajkjgabckkkkcbakkdfjknbbca at location 9
acb
bac
bca
Found one match: bca is in hhhlajkjgabckkkkcbakkdfjknbbca at location 27
cba
Found one match: cba is in hhhlajkjgabckkkkcbakkdfjknbbca at location 16
cab
```

Given shortStr as "`ckkk`", the method should show display something like:

```
ckkk
Found one match: ckkk is in hhhlajkjgabckkkkcbakkdfjknbbca at location 11
ckkk
Found one match: ckkk is in hhhlajkjgabckkkkcbakkdfjknbbca at location 11
ckkk
Found one match: ckkk is in hhhlajkjgabckkkkcbakkdfjknbbca at location 11
ckkk
Found one match: ckkk is in hhhlajkjgabckkkkcbakkdfjknbbca at location 11
ckkk
Found one match: ckkk is in hhhlajkjgabckkkkcbakkdfjknbbca at location 11
ckkk
Found one match: ckkk is in hhhlajkjgabckkkkcbakkdfjknbbca at location 11
```

```
kckk
kckk
kkck
kkkc
Found one match: kkkc is in hhhlajkjgabckkkkcbakkdfjknbbca at location 13
kkkc
Found one match: kkkc is in hhhlajkjgabckkkkcbakkdfjknbbca at location 13
kkck
kkck
kkkc
Found one match: kkkc is in hhhlajkjgabckkkkcbakkdfjknbbca at location 13
kckk
kckk
kkck
kkkc
Found one match: kkkc is in hhhlajkjgabckkkkcbakkdfjknbbca at location 13
kkkc
Found one match: kkkc is in hhhlajkjgabckkkkcbakkdfjknbbca at location 13
kkck
kkkc
Found one match: kkkc is in hhhlajkjgabckkkkcbakkdfjknbbca at location 13
kkck
kckk
kckk
```

Note: While our interest is to find if any occurrence of a permutation value exists in longStr (and not all of them; for instance, if abc exists twice in longStr, it is sufficient for the algorithm to find only the first occurrence), you should observe the output carefully, since the characters of shortStr may not be unique, as in the second run!

You will need to run the program multiple times. With each run, you will need to provide a shortStr with an incremented length of 5. That is, you need to run with shortStr of 5 characters, 10 characters, 15 characters, etc. for up to length of 200 characters (or higher value if required for your timing measurement) and measure the corresponding run time for each run. You can use Java's built-in time function for finding the execution time. In each run, you need to provide a longStr that is longer in length than shortStr. The contents of the strings are not important; however, your values should show that you program work correctly. You should redirect the output of each program to an *out.txt* file. You should write about your observations on timing measurements in a separate text file. You are required to submit the two fully commented Java source files, the compiled executables, and the text files.

Briefly explain what is the complexity of your algorithm. More specifically, is your solution has acceptable complexity; is it scalable enough; etc. If not, what are the reasons behind that?

Version 2:

In this second version, you will need to provide and alternative/smarter solution (to solve the same exact problem as above) however this second solution must significantly have lower time complexity than you first version. The solution may or may not use recursion; this is up to you.

a) Explain the details of your algorithm, and provide its time complexity. You must clearly justify how you estimated the complexity of your solution.

b) Submit both the pseudo code and the Java program, together with your experimental results. Keep in mind that Java code is not pseudo code. See full details of submission details below.

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 questions, submit all your answers in PDF (<u>no scans of handwriting; this will result in your answer being discarded</u>) or text formats only. Please be concise and brief (less than ¼ of a page for each question) in your answers. Submit the assignment under Theory Assignment 1 directory in EAS or the correct Dropbox/folder in Moodle (depending on your section).

For the Java programs, you must submit the source files together with the compiled files. The solutions to all the questions should be zipped together into one .zip or .tar.gz file and submitted via EAS under Programming 1 directory or under the correct Dropbox/folder in Moodle. You must upload at most one file (even if working in a team; please read below). In specific, here is what you need to do:

1) Create **one** zip file, containing the necessary files (.java and .html). 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.

2) If working in a group, only one of the team members can submit the programming part. Do not upload 2 copies.

<u>Very Important:</u> Again, the assignment must be submitted in the right folder of the assignments. Depending on your section, you will either upload to EAS or to Moodle (your instructor will indicate which one to use). **Assignments uploaded to an incorrect folder will not be marked and result in a zero mark. No resubmissions will be allowed.**

⇨ Additionally, for the programming part of the assignment, a demo is required (please refer to the courser 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.