COMP202 **Programming Assignment**
Complexity of Algorithms **27 March 2020**

Title: Longest Bubble Problem

Submission deadline: 27 April 2020 (Monday, 2pm)

Submissions should be made via the Departmental Submission Server:
`https://sam.csc.liv.ac.uk/COMP/Submissions.pl`

Notes:

1. This assessment is worth 10% of your overall course grade.
2. Standard late penalties apply, as per university policy.
3. Learning outcomes covered by this CA task will also be covered within resit exam: this prevents the need for explicit reassessment of this component. The resit exam will replace the CA component in case this is failed.

Learning outcomes

The purpose of this exercise is for you to demonstrate the following learning outcomes and for me to assess your achievement of them.

1. To demonstrate how the study of algorithmics has been applied in a number of different domains.
2. To introduce formal concepts of measures of complexity and algorithms analysis.
3. To introduce fundamental methods in data structures and algorithms design.

Note: You will be provided with a collection of sample inputs together with correct answers to these inputs. You should aim at submitting your final program only if it produces correct answers to all these inputs.

Academic integrity

The work that you submit should be the product of yourself (alone), and not that with any other student or outside help. Obviously I am providing a source code framework within which you will provide your own method for solving this problem, but the code that you write within this framework should be your own code, not that obtained in collaboration with other students or other outside assistance or sources.

Problem Description

Let us first define a notion of a *bubble*. Any sequence of $k \geq 3$ positive integers $b_1, b_2, \ldots, b_k$ is called a *bubble* if it is initially strictly increasing up to a certain index and then it is strictly decreasing from that index till its end. Formally, there exists an index $j \in \{2, 3, \ldots, k-1\}$ such that $b_1 < b_2 < \cdots < b_j$ and $b_j > b_{j+1} > \cdots > b_k$. Note that we insist that a bubble has length at least 3.
I introduce here a problem which I will call the Longest Bubble problem. You are given as input a sequence of n *positive* integers $a_1, a_2, \ldots, a_n$. We do not assume that the input sequence is sorted.

Your task is to find the length of the longest *subsequence* of the input sequence $a_1, a_2, \ldots, a_n$ that is a bubble or output 0 if there is no such subsequence (that is a bubble) in this sequence. That is, you need to fnd the length k of a subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ for some integer indices $1 \leq i_1 < i_2 < \cdots < i_k \leq n$, such that the subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ is the longest bubble; or output 0 if there is not such subsequence in the input sequence. Note that whereas a subsequence keeps the order of elements, it is allowed to skip some of the elements from the sequence $a_1, a_2, \ldots, a_n$.
Suppose, for instance, that $n = 9$ and that the input sequence is:

2 1 4 7 5 4 6 3 1,

that is, $a_1 = 2, a_2 = 1, a_3 = 4, a_4 = 7, a_5 = 5, a_6 = 4, a_7 = 6, a_8 = 3, a_9 = 1$. Then, for example, a_3, a_4, a_5, a_6, a_8 is a bubble of length 5; the longest bubble, of length 7, is $a_2, a_3, a_4, a_5, a_6, a_8, a_9$. Another bubble of length 7 is $a_1, a_3, a_4, a_5, a_6, a_8, a_9$. Thus, the answer in this case will be 7.

Observe that the input sequence $a_1, \ldots, a_n$ can have the same consecutive integers. For instance, if the sequence $a_1, a_2, \ldots, a_n$ is non-decreasing, that is, $a_1 \leq a_2 \leq \ldots \leq a_n$, then there is no bubble and the answer is 0. Similarly, the answer is 0 in case when the input sequence is non-increasing.

For further examples of inputs together with answers, see the text file dataTwo.txt that I provide (see explanation of the data format below). In fact the example sequence 2 1 4 7 5 4 6 3 1 above, is the first instance in dataOne.txt and in dataTwo.txt (dataTwo.txt contains also solutions).

You should write a procedure that for any given input sequence of n positive integers (multiple indentical numbers allowed) finds the length of the longest bubble (or 0 is there is none). Your procedure should only output the value of the longest bubble or 0.

Additionally, you should include a brief idea of your solution in the *commented* text in your code and you should also include a short analysis and justification of the running time of your procedure. These descriptions are part of the assessment of your solution.

Hints

You are supposed to solve the Longest Bubble problem by dynamic programming. This problem is related to one of the exercises from the exercise list on dynamic programming. In your solution (described as part of the assessment) you should come up with appropriate recurrence solution to the problem which then you should translate into a sequential solution that fills in dynamic programming tables in an appropriate way in your implementation.

Programming Language

You will be using Java as the programming language.

Program framework description

I provide a template program called `Main.java` that you will use (without altering anything but the place to put your code) to include the code of your procedure to solve the Longest Bubble problem. Note that you may add additional procedures outside of the procedure LongestBubble if needed.

To use this template, after you write your code inside of procedure called LongestBubble, you must include in the *current directory* the input text files dataOne.txt and dataTwo.txt. Note, however, that if you need any additional procedures, you may include them outside of the text of the procedure LongestBubble.

To compile your program in command line (under Linux/Unix) use something like this (this may differ within your system):

`javac Main.java`

Then, you can run your program from command line like this

`java Main -opt1`

which will run the program with dataOne.txt as input file and will output answers (that is the values of the longest bubble or 0) to all instances in order they appear in dataOne.txt. You may use your own dataOne.txt in the format (see below) to experiment with your program. Input file dataOne.txt may contain any number of correct input sequences.

Now, if you run the program with

`java Main -opt2`

this will run the program with dataTwo.txt as input file. In this case, the output will be the indices of inputs from dataTwo.txt that were solved incorrectly by your program together with percentage of correcly solved instances. If all instances are solved correctly, the output should be 100%. File dataTwo.txt contains the same instances as dataOne.txt, but, in addition dataTwo.txt also contains the correct, that is values of the longest bubbles or 0, answers to these instances. You may use dataTwo.txt to test the correctness of your program.

Description of the input data format:

Input data text file dataOne.txt has the following format (this example has 2 inputs, each input ends with A; note the number 0 is the part of the input format, but not part of the input sequences):

$$
\begin{array}{c}
0 \\
a_1 \\
a_2 \\
... \\
... \\
a_n \\
A \\
0 \\
a_1 \\
a_2 \\
... \\
... \\
a_n \\
A
\end{array}
$$

In general file dataOne.txt can have an arbitrary number of distinct inputs of arbitrary, varying lengths. File dataOne.txt contains 31 different instances of the problem. Observe that n is not explicitly given as part of the instance. Also 0 which starts each instance does not have any particular purpose; it is just format of the input data.

Input data text file dataTwo.txt has the following format (this example has again 2 inputs, each input ends with A):

$$
\begin{array}{c}
0 \\
ans_1 \\
a_1 \\
a_2 \\
... \\
... \\
a_n \\
A \\
0 \\
ans_2 \\
a_1 \\
a_2 \\
... \\
... \\
a_n \\
A
\end{array}
$$

There ans_1 (ans_2, respectively) is the value of the longest bubble for the first (second, respectively) instance or 0 if there is no bubble in those instances.

File dataTwo.txt contains the same 31 instances of the problem as in file dataOne.txt but in addition has the answers. This data can be used to test the correctness of your procedure.

Again, in general file dataTwo.txt can have an arbitrary number of distinct input sequences of arbitrary, varying lengths. It is provided by me with correct answers to these instances.

The solutions shown in dataTwo.txt are (at least) the claimed solutions for each sample input, computed by my program. Recall that your solution should print out the value of the longest bubble in the given sequence for the given instance, or 0 in case if this instance contains no bubble. Note, that your program does not need to output this longest bubble.

Program submission

You must submit a **single** Java source code in a single file that must be called `Main.java` (not the byte code) to the Departmental Submission Server:

`https://sam.csc.liv.ac.uk/COMP/Submissions.pl?strModule=COMP202`

Your source file `Main.java` must have the (unaltered) text of the template provided by me, containig the text of your procedure inside the LongestBubble method. You are allowed to include additional procedures outside the LongestBubble method if needed. In addition, within commented parts of method LongestBubble, you should describe your recursive solution and how you implement it sequentially. Moreover, you should also describe a short running time analysis of your sequential implementation in terms of n and big-O notation.

You are responsible that your source code program `Main.java` can be compiled correctly with Java compiler and executed on (any) one of the computers in the Computer Science Department's that runs Java installation under Linux, where I will test your programs. You may also remotely connect to any Linux computer in the Department to compile/test your program. Programs that will not correctly compile on Departmental Linux Java installation will automatically receive mark 0 for the correctness part.

Assessment

Marking on this assessment will be performed on the following basis:

- Accuracy of solution (e.g., does it give correct answers for all of the test cases, and is it a general solution method for instances of this problem?): 60%
- Clarity of solution (Is your program easy to follow? Is it commented to help me see your solution method?): 10%
- Correctness of time complexity (in big-O notation of the problem size n) description of your procedure and description of your solution.

(Have you provided an analysis of the (aymptotic) running time of your method, and is that analysis correct? Is the description of your solution (recursion and sequential implementation) correct and clrearly written?: 20%

- Optimality of solution (Do you have the "best", i.e. quickest, solution possible in terms of the asymptotic runtime?): 10%