- This exam has 2 questions for a total of 100 marks.
- Please write *legibly*. If you strike something off, please strike it off clearly. Please do *not* overwrite text that you wish me to read. (Parts of) Answers that are not easy to read, or can be interpreted in more than one way, will not be evaluated.
- Please keep in mind that I can only read what is there on your answer sheets, and not the thoughts that are in your mind when you wrote an answer (or at any other time ...). So: please *write down* whatever you wish me to know while I am evaluating the answer.
- Please follow that instructions that I sent you by Moodle, about this exam.
- Warning: CMI's academic policy regarding cheating applies to this exam.

Unstated assumptions and lack of clarity in solutions can and will be used against you during evaluation.

Please ask the invigilator if you have questions about the questions. Array indexing starts at 1—not 0—in these problem statements.

1. Obamer Thetadation is a world leader in producing the software deployed in industrial-scale sheet metal cutting machines. The typical sheet metal cutting problem has as input (i) the dimensions of a rectangular sheet of metal, and (ii) the dimensions of a collection of shapes that need to be cut out of this metal sheet. The goal is to cut as many copies of these shapes as possible, from the given sheet. These shapes are, in general, not rectangular, or nor even convex. They can be arbitrarily complex, and have "holes". Indeed, some shapes may be mostly made up of hole(s); think of the metallic beading that goes around the body of a laptop. The challenge that Obamer Thetadation has to solve over and over, is to find out how to lay the given shapes out on the given metal sheet, usually with much "nesting"—one piece laid out inside the "hole" made by another—so as to maximize the number of pieces that are cut out, and minimize wastage of material. The ability to compute this layout faster and better than the competition is critical to the company's success.

   The company has, over its many years of existence, evolved a sophisticated (so-called) *nesting strategy* that it employs in its cutting machines. This strategy employs a collection of algorithms that are executed in a particular sequence, with one algorithm taking as input some subset of the outputs of its predecessors. One of these intermediate outputs is an array $A[1 \ldots n]$ of n positive integers, where each such integer is (i) expressed in binary, and (ii) no larger than n^{12}. Here n is the best estimate that the strategy currently has about the number of pieces that can possibly be cut

out from the input sheet, and A[i] is a certain numerical "measure" that the strategy has computed for the ith piece. Further processing involves verifying whether this collection of pieces can indeed be laid out on the given input sheet; if the answer is YES, then the strategy proceeds to compute a layout of these pieces. If the answer is NO, then it goes back and tries a different, smaller value of n.

The first step in checking if the pieces can be laid out, is *sorting* these n numbers in non-decreasing order, and this is what you will help the company with, in this question. Note that n can be very large, and so it is important that the algorithm does the sorting really fast.

(a) Write the *complete* pseudocode for a procedure which takes an array A of the type described above as its argument, and returns a sorted—in non-decreasing order—version of A in $\Theta(n)$ time. You may assume that each array access and each arithmetic or logical operation involving up to two numbers, can be done in constant time. [20]

You will get the credit for this part only if your pseudocode is self-contained, correct, and runs within the required time bound.

(b) Argue that your procedure of part (a) correctly sorts the input array. [10]

(c) Argue that your procedure of part (a) runs in $\Theta(n)$ time on an input array A of the type described in the problem statement. [10]

2. The input to the *Superlative Sub-array* problem is an array $A[1 \ldots n]$ of $n \geq 1$ integers, and the output is an array $B[1 \ldots m]$ such that:

   1. $m \leq n$;
   2. B is a *sub-array* of A. That is, there is a function f from the indices of B to the indices of A such that (i) $B[i] = A[f(i)]$ holds for $1 \leq i \leq m$, and (ii) $f(i) < f(i+1)$ holds for $1 \leq i < m$. Intuitively: B can be obtained by deleting zero or more elements of A;
   3. B forms an *increasing* sequence. That is, $B[i] < B[i+1]$ holds for $1 \leq i < m$;
   4. B is a *longest* array—one with the most number of elements—satisfying the above conditions.

   In this question we derive various algorithms for *Superlative Sub-array*. For keeping the pseudocode simple, assume throughout that the array A and the number n are globally accessible.

   (a) Let LENGTH-SS-FIRST(i) denote *the length* of a superlative sub-array S of the [10]

sub-array A[i...n] such that S[1] = A[i] holds[1]. Write a *recurrence relation* for LENGTH-SS-FIRST(i). That is: express LENGTH-SS-FIRST(i) *mathematically* in terms of one or more values LENGTH-SS-FIRST(j) for some $j \neq i$. **Explain why your recurrence is correct.** (use induction)

This part does *not* ask for pseudocode, so your answer should *not* be pseudocode. You will get the credit for this part only if *both* your recurrence *and* your explanation are correct.

(b) Write pseudocode for a *recursive* function SS-STARTINGAT(i) which takes an index $1 \leq i \leq n$ as input and returns a superlative sub-array S of the sub-array A[i...n] such that S[1] = A[i] holds. Note that it is *not* enough to return just the *length* of such a sub-array; your function should return one such *sub-array* S. Also, your function must be recursive: it must make calls of the form SS-STARTINGAT(j) for some $j \neq i$. **Explain why your function is correct.** [10]

You will get the credit for this part only if *both* your pseudocode *and* your explanation are correct.

(c) Write pseudocode for a function SS() which returns a superlative sub-array of the array A. **Explain why your function is correct.** [10]

You will get the credit for this part only if *both* your pseudocode *and* your explanation are correct.

(d) Convert your function SS() from part (c) to a **non-recursive** DP algorithm SS-DP() that computes a superlative sub-array of the input array A[1...n] in time $\mathcal{O}(n^c)$ for some constant c which is independent of n. [10]

You will get the credit for this part only if your pseudocode is (i) correct, (ii) non-recursive, and (iii) runs in time polynomial in n.

(e) Argue that your algorithm of part (d) correctly computes the required solution. [10]

(f) Argue that your algorithm of part (d) runs in $\mathcal{O}(n^c)$ time on an input array A[1...n], for some constant c which is independent of n. [10]

[1] This S need not necessarily be a superlative sub-array of A[i...n]; it is just a longest one among those increasing sub-arrays of A[i...n] whose first element is A[i].