Instructions

This exam has 2 questions for a total of 200 marks, of which you can score at most 100 marks. You may answer any subset of questions or parts of questions. All answers will be evaluated. Go through all the questions once before you start writing your answers. Use a pen to write. Answers written with a pencil will *not* be evaluated.
Do not use hash functions or hash tables/maps in your solutions. All arrays in the questions are zero-indexed. If you wish to use one-indexing in your solutions, please state this explicitly in each such solution. You may split up the pseudocode for an algorithm into two or more functions, if you wish; you don't have to write the entire pseudocode for an algorithm as one function. You may use the function call LENGTH(A) to get the length—number of elements—of array A. You may freely invoke functions that you have written as part of a different answer in the same answer sheet. You do *not* have to use loop invariants while proving the correctness of algorithms; but you must correctly explain why each loop (if there are some) does what you expect it to do.
You may assume the following when analyzing the running time of algorithms: (i) each operation on a pair of bits takes constant time; (ii) comparing a pair of numbers takes constant time; (iii) reading/writing an array element using its index takes constant time, and, (iv) creating a new array, finding the length of an array, and returning an array, each takes time linear in the length of the array. You must clearly state any other assumption that you make.
You may use the Master Theorem for solving recurrences, but you must properly state the version of the theorem that you use, before applying it.
Unstated assumptions and lack of clarity in solutions can and will be used against you during evaluation. You may freely refer to statements from the lectures in your arguments. You don't need to reprove these unless the question explicitly asks you to, but you must be precise. Please ask the invigilators if you have questions about the questions.
Warning: CMI's academic policy regarding cheating applies to this exam.

1. All integer input arguments in this problem are represented as binary arrays, with the rightmost bit being the least significant. E.g., the number written as 35 in decimal is represented as the array $[1, 0, 0, 0, 1, 1]$. Further, there are no padding zeros in the arguments; the number zero is represented as $[0]$, and the leftmost bit is 1 for all other inputs. Note that the numbers are represented as *arrays*, not Python lists; you cannot change the *length* of these arrays after creation, you can only change the contents of their cells.

   (a) Write the *complete* pseudocode for an algorithm ADD(X,Y) that takes two integers x, y as input arguments, runs in time linear in the total length of the input, and returns[1] the sum $x + y$. Make sure that your code correctly takes care of overflows, and that the array which is returned does *not* start with a zero unless [20]

[1] *Not* prints.

$x + y = 0$.

You will get the credit for this part only if your algorithm is (i) correct, and (ii) runs in worst-case linear time.

(b) Argue that your algorithm of part (a) runs in time linear in the total input size, in the worst case. [10]

(c) Write the *complete* pseudocode for an algorithm MULTIPLY(X,Y) that takes two integers x, y with n bits each as input arguments, runs in time $\mathcal{O}(n^c)$ for some constant $c < 2$, and returns the product $x \times y$. Make sure that your code correctly takes care of overflows, and that the array which is returned does not start with a zero unless $x \times y = 0$. [30]

You will get the credit for this part only if your algorithm is (i) correct, and (ii) runs in worst-case time $\mathcal{O}(n^c)$ for some constant $c < 2$.

(d) Prove that your algorithm of part(c) is correct. [20]

(e) Prove that your algorithm from part (c) runs in $\mathcal{O}(n^c)$ time in the worst case, for a constant $c < 2$. [20]

2. In this question the notation $\hat{A}$ denotes, for an array A of integers, the version of array A that is sorted in non-decreasing order. We define the ith *order statistic* of an integer array A of length n, as the integer $\hat{A}[i]$. You may assume that the following functions are available to you as black boxes:

- A function PARTITION(X, v) that takes an integer array X and an integer v as arguments, runs in time linear in the length of X, and returns a 3-tuple of arrays (L, E, G) which form a partition of X into elements that are lesser, equal to, and greater than v, respectively.
- A function SORT(X) that takes an integer array X of length n as input, runs in $\Theta(n^2)$ time, and returns the array $\hat{X}$.

(a) Write the *complete* pseudocode for an algorithm ORDERSTATISTIC(A, i) that takes an integer array A of length n and an integer $0 \leq i < n$ as inputs, runs in worst-case (deterministic) $\mathcal{O}(n)$ time, and returns the ith order statistic of A. [40]

You will get the credit for this part only if your algorithm is (i) correct, and (ii) runs within the stated time bound.

(b) Prove that your algorithm of part (a) is correct. [30]

(c) Prove that your algorithm of part (a) runs in worst-case deterministic $\mathcal{O}(n)$ time on an input array of length n. [30]