Instructions

This exam has 5 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* overwrite; instead, strike off cleanly and write in a fresh spot.

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 following functions, and assume in your analysis that they work correctly with the stated worst-case running times:

- LENGTH(A): returns the length n of array A, and runs in $\Theta(n)$ time;
- MERGESORT(A): returns the version of integer array A which is sorted in non-decreasing order, and runs in $\Theta(n \log n)$ time when A has n elements;
- BINARYSEARCH(A, v): returns the index of integer element v in a *sorted* array A, or NIL if v is not present in A. Runs in $\Theta(\log n)$ time when A has n elements.

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.

Clearly describe the meaning of any Python (or other) syntax that you use, so that I can correctly evaluate your answer. E.g., if you use the notation A[i : j] to denote a sub-array, clearly explain what you mean by this. And similarly for any other notation.

You may assume the following when analyzing the running time of algorithms: (i) comparing a pair of numbers takes constant time; (ii) reading/writing an array element using its index takes constant time, and, (iii) creating a new 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 (*except* when the question says otherwise), 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. Unroll each of the following recurrences to come up with an estimate $f(n)$ that satisfies $T(n) = \Theta(f(n))$.

   In each case, verify your estimate by induction. Note that this involves verifying *two* asymptotic bounds for each part, for the *same* function f.

   In each case, you may assume bounds of the form $T(n') \leq c'$ and $T(n'') \geq c''$ where n', n'', c', c'' are all fixed constants of your choice. That is, you may assume constant upper and lower bounds for inputs of up to some constant size.

   You will *not* get the credit for either part if you use the Master Theorem (or some such) to get at the estimate.

   (a) $T(n) = 4T(n/2) + 10$ [15]

   (b) $T(n) = 2T(n/2) + 6n$ [15]

2. Let A be an array of integers. A *subsequence* of A is any array obtained from A by deleting zero or more elements of A *without* changing the order of the remaining elements (if any). In particular, the empty array, and A itself, are subsequences of A. As a concrete example, here are two subsequences of the array $A = [41, 24, 35, 89, 0, 32, 8, 48, 89, 16]$: (i) $[24, 0, 32, 48]$, and (ii) $[41, 89, 0, 8, 48, 89, 16]$.

   An array C is a *common subsequence* of arrays A and B if C is a subsequence of A, and is a subsequence of B. A *longest common subsequence* of A and B is a common subsequence C of A and B with the maximum number of elements.

   (a) Write the *complete* pseudocode for a *recursive* algorithm LCS(A,B) that takes two integer arrays A, B as input arguments, and returns a longest common subsequence of A and B. [20]

   You may use the operation "X + Y" to append the elements of Y (in order) to the array X. Clearly describe any other operator that you use.

   Note that your algorithm must return a *subsequence*. Make sure that your algorithm handles the base case(s) correctly. You will get the credit for this part only if your algorithm is (i) recursive, and (ii) correct.

   (b) Prove using induction that your algorithm of part (a) is correct. [10]

   (c) Write a recurrence for the running time of your algorithm of part (a), and explain why this recurrence correctly captures the running time of the algorithm. Clearly state any assumptions that you make about the running times of various operations. You do *not* have to *solve* the recurrence. [10]

3. 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. For an integer array A with an even number

of elements $n \geq 2$, we define the *lower median* of A as the integer $\hat{A}[\frac{n-2}{2}]$ and the *upper median* of A as the integer $\hat{A}[\frac{n}{2}]$.

(a) Write the *complete* pseudocode for an algorithm LowerMedian(A, u) that takes (i) an integer array A of even length and (ii) its upper median u as inputs, runs in worst-case (deterministic) linear time in the length of A, takes at most a constant amount of extra space, and returns the lower median of A. [20]

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

(b) Prove that your algorithm of part (a) is correct. [10]

(c) Prove that your algorithm of part (a) runs in worst-case deterministic $\mathcal{O}(n)$ time on an input array of length n. You may assume that comparing a pair of numbers takes constant time; clearly state any other assumptions that you make. [10]

4. (a) Write the *complete* pseudocode for an algorithm CheckSum(A, T) that takes an array A of n integers and a target integer T as inputs, runs in $\mathcal{O}(n \log n)$ time, and finds if there are two indices $0 \leq i \leq j < n$ in A such that $A[i] + A[j] = T$. If there *are* two such indices (possibly, with $i = j$), then the algorithm should return "Yes". Otherwise it should return "No". [20]

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. [10]

(c) Prove that your algorithm of part (a) runs in worst-case deterministic $\mathcal{O}(n \log n)$ time on an input array of length n. Clearly state any assumptions that you make. [10]

5. Let A be an array containing n integers. For $0 \leq i \leq j < n$ we use $A[i : j]$ to denote the *sub-array* $[A[i], A[i+1], \ldots, A[j]]$. Note that this is **different** from the Python convention for lists.

The Maximum Sub-array problem has such an array A as input, and asks for the number

$$\max_{0 \leq i \leq j < n} \sum_{i \leq k \leq j} A[k].$$

That is, the goal is to compute the maximum sum, taken over all sub-arrays of A, of all the elements in a sub-array of A.

(a) Write the *complete* pseudocode for a divide-and-conquer algorithm MaxSubArray(A) that takes an array A of n integers as input, runs in $\mathcal{O}(n \log n)$ time, and solves the Maximum Sub-array problem. The algorithm should return the maximum sum. [30]

You will get the credit for this part only if your algorithm is (i) correct, (ii) uses divide-and-conquer to solve the problem, and (iii) runs within the stated time bound.

(b) Prove that your algorithm of part (a) is correct. [10]

(c) Prove that your algorithm of part (a) runs in worst-case deterministic $\mathcal{O}(n \log n)$ time on an input array of length n. Clearly state any assumptions that you make. [10]