- This exam has 6 questions for a total of 135 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.
- Warning: CMI's academic policy regarding cheating applies to this exam.

You do *not* have to use loop invariants for proving the correctness of algorithms; but you must correctly explain why each loop (if there are some) does what you expect it to do. Of course, you *may* use loop invariants if you wish.

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.

1. In this question we consider graphs that are finite and undirected. Such a graph is *simple* if has neither multiple edges nor self-loops. An *isolated* vertex in a graph is a vertex with no edge incident on it. We say that a graph is *special* if it is simple and has no isolated vertex. The notation $|S|$ denotes the cardinality of set S.

   Consider the following claim and its proof:

   Claim

   Every special graph $G = (V, E)$ that satisfies $|V| = |E| + 1 \geq 2$, is acyclic.

   Proof: By induction on $|V|$.

   - Base case: $|V| = 2, |E| = 1$: correct by inspection.
   - Inductive assumption: suppose the claim holds for all special graphs with at most k vertices, for some $k \geq 2$.
   - Inductive step: Let $G = (V, E)$ be a special graph with $|V| = k = |E| + 1$. Consider an arbitrary graph G' obtained from G by adding (i) a new vertex v, and (ii) an arbitrary edge from v to some vertex of G. Then G' (i) has $k + 1$ vertices and k edges, (ii) has no isolated vertices, and (iii) is acyclic. Hence proved.

(a) Provide a counter-example to the claim, with at most 10 vertices [5]

(b) Clearly explain what is wrong with the above proof. [5]

2. (a) Write the pseudocode for a *recursive* algorithm QUOTREM(x, y) that takes two integers $x \geq 0, y \geq 1$ as arguments, and uses *repeated subtraction* to find the quotient q and the remainder r that are obtained when x is divided by y. [10]

   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. Make sure that you handle (all) the base case(s). [10]

3. The SecondBest problem is defined as follows:

SecondBest

- Input: An integer $n \geq 1$ and an array A of n integers. Array A is indexed from 0; its elements are thus $A[0], A[1], \ldots, A[(n-1)]$.
- Output: The *second largest* number x which is present in A, or the special value NIL if there is no such number in A.

  A number x is the second largest number in A if: (i) x is present in A; (ii) the largest number present in A is y, and $y > x$ holds; and, (iii) there is no element z in A such that $x < z < y$ holds.

(a) Give an example of an input to this problem with $n = 5$, where the (correct!) output value is NIL. [5]

(b) Write the *complete* pseudocode for an algorithm SECONDBEST(n, A) that solves the SecondBest problem in $\mathcal{O}(n)$ time in the worst case. You will get the credit for this part only if your algorithm is correct, and runs within the required time bound. [10]

(c) Argue that your algorithm of part (a) correctly solves the problem. [10]

(d) Prove that your algorithm from part (a) runs in $\mathcal{O}(n)$ time in the worst case. For this you may assume that each array operation, and each comparison of a pair of numbers, take constant time. Clearly state any other assumptions that you make. [10]

4. Prove the following statement using the definition of the $\mathcal{O}()$ notation that we saw in class: [10]

Claim

Let $f(n)$, $g(n)$, and $h(n)$ be functions from non-negative integers to real numbers. If $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(h(n))$, then $f(n) = \mathcal{O}(h(n))$.

5. Prove or disprove:

   (a) $\log_2 3n = \mathcal{O}(\log_2 2n)$ [10]

   (b) $2^{3n} = \mathcal{O}(2^{2n})$ [10]

6. Consider the following problem:

Count Palindromic Substrings

- Input: An integer $n \geq 1$ and a string S of length n. String S is a list indexed from 0; its elements are thus $S[0], S[1], \ldots, S[(n-1)]$.
- Output: The number of different pairs (i, j) ; $0 \leq i < j \leq (n-1)$ such that the substring $S[i]S[i+1] \cdots S[j]$ of S is a palindrome.

   (a) Write the *complete* pseudocode for an algorithm that solves the above problem in $\mathcal{O}(n^c)$ time—in the worst case—for some constant c. You will get the credit for this part only if your algorithm is correct and complete, and runs within the required time bound. [15]

   (b) Prove that your algorithm of part(a) is correct. [15]

   (c) Prove that your algorithm from part (a) runs in $\mathcal{O}(n^c)$ time in the worst case, for a small constant c. For this you may assume that each array operation, and each comparison of a pair of characters, take constant time. Clearly state any other assumptions that you make. [10]

- This exam has 4 questions for a total of 170 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.
- Warning: CMI's academic policy regarding cheating applies to this exam.

You do *not* have to use loop invariants for proving the correctness of algorithms; but you must correctly explain why each loop (if there are some) does what you expect it to do. Of course, you *may* use loop invariants if you wish.

The arrays in this question paper are objects whose sizes are fixed, in the sense that the size cannot be changed after the array is created. In particular, these are *not* Python's lists, whose sizes can be changed using, say, append(). Also: note that Python's list.append() does *not* run in *worst-case* constant time; be mindful of this when writing your pseudocode.

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.

1. Recall the SecondBest problem from Quiz 1:

SecondBest

- Input: An integer $n \geq 1$ and an array A of n integers. Array A is indexed from 0; its elements are thus $A[0], A[1], \ldots, A[(n-1)]$.
- Output: The *second largest* number x which is present in A, or the special value None if there is no such number in A.

Each part below shows (a Python version of) the pseudocode offered as a solution to this problem by someone among you. And in each case the pseudocode is *wrong*; it produces an incorrect output for certain valid inputs.

For each part, come up with a valid input of the form (n, A) where n is *at most* 5, which the given pseudocode fails to solve correctly. Clearly explain *how and why* the pseudocode fails to correctly solve this input.

You will get the credit for each part only if (i) the input that you specified is valid, (ii) the given pseudocode produces a wrong output for this input, (iii) you have described the **actual** output that the pseudocode produces when given this input, and, (iv) you have **explained** the reason why the pseudocode produces this wrong output.

Please make sure that you write the part number correctly, since there is no other way for me to match your answer to the part number in this question.

(a) [10]

```
def secondBest(n, A):
    if n == 1:
        return None
    else:
        x = A[0]
        y = None
    for i in range(1,n):
        if A[i] > x:
            y = x
            x = A[i]
    return y
```

(b) [10]

```
def secondBest(n, A):
    if n == 1:
        return None
    largest = A[0]
    secLargest = None
    for i in range(1,n):
        if A[i] < largest:
            secLargest = A[i]
    for i in range(1,n):
        if A[i] > largest:
            secLargest = largest
            largest = A[i]
    return secLargest
```

2. Consider the following problem:

Max Pair Product

- Input: An integer $n \geq 2$ and an array A of n non-negative integers. Array A is indexed from 0; its elements are thus $A[0], A[1], \ldots, A[(n-1)]$.
- Output: The *maximum value* of the product of two elements of A that are *at distinct indices* in A. Note that the two elements of A whose product is the required output, need not be distinct as numbers; but they must appear at different *indices* in A.

(a) Write the *complete* pseudocode for an algorithm MAXPAIRPRODUCT(n, A) that solves the above problem in $\mathcal{O}(n)$ time and uses at most a constant amount of extra space, in the worst case. You will get the credit for this part only if your algorithm is correct and complete, and runs within the required time and space bounds. [15]

(b) Explain why your algorithm of part (a) is correct. [15]

(c) Prove that your algorithm from part (a) runs in $O(n)$ time and constant extra space in the worst case. For this you may assume that each array operation, and each comparison of a pair of numbers, take constant time. Clearly state any other assumptions that you make. [10]

3. Assume for the sake of this question that 5000 candidates attempted Part A of CMI's 2023 MSc DS entrance exam. Consider the following claim and its proof: [10]

Claim

All the 5000 candidates who attempted part A of this exam, scored the exact same marks for part A.

Proof

By induction on the size of subsets of candidates who attempted Part A.

- Base case: Take any subset consisting of one candidate. Clearly, all candidates in this subset scored the same marks for part A (Because there is only one candidate in the subset.).
- Inductive assumption: suppose the claim holds for all subsets with at most k candidates.
- Inductive step: Let S be a subset with $k+1$ candidates. Remove an *arbitrary* candidate x from S to get the subset S' with k candidates. By the inductive assumption, all candidates in the set S' scored the same marks—say m'—for part A. Now remove another arbitrary candidate y ; $y \neq x$ from the *original* subset S to get a subset S'' with k candidates. By the inductive assumption, all candidates in the set S'' scored the same marks—say m''—for part A.

  Since the set S'' contains candidate x, we get that x scored the same marks—namely, m''—as every other candidate in the set $(S'' \setminus \{x\})$. Similarly, the set S' contains candidate y, and so we get that y scored the same marks—namely, m'—as every other candidate in the set $(S' \setminus \{y\})$.

  But notice that $(S'' \setminus \{x\}) = (S' \setminus \{y\})$. So we get that m' is in fact equal to m'', and that all the candidates in set S scored the same marks for part A.

Hence we get, by induction, that all the candidates scored the same marks.

It is obvious (I hope ...) that the claim cannot possibly be correct. Clearly explain what is wrong with the above proof.

4. Recall that a palindrome is a string that reads the same in either direction. A *non-trivial palindrome* is a palindrome with length (number of characters) at least two. Consider the

following problem

Palindrome Sequence

- Input. An integer $n \geq 2$ and a string S of length n. String S is an array indexed from 0; its elements are thus $S[0], S[1], \ldots, S[(n-1)]$.
- Output: True if S can be obtained by concatenating one or more non-trivial palindromes, and False otherwise. Equivalently: True if S can be partitioned (that is: cut up, without dropping any element) into one or more non-trivial palindromes, and False otherwise.

Some examples with $n = 10$:

- True instances: `nnllknkoyo`, `fjbubjfhuh`, `mttmzizzcz`, `fjfyspsyqq`, `abcdeedcba`
- False instances: `cmmcmhdaba`, `azatznnzth`, `ummnurlxmv`, `xqeppajynx`, `tyjglnvmaa`

(a) Write the *complete* pseudocode for a function IsNTP(w) that returns True if string w is a non-trivial palindrome and False otherwise, and runs in *linear time* in the length of w in the worst case. You will get the credit for this part only if your algorithm is correct and runs within the required worst-case time bound. [10]

(b) Write the *complete* pseudocode for a *recursive* function IsNTPSequence(n, S) that solves Palindrome Sequence. You may use the function IsNTP() that is described in part (a) as a black box, even if you have not solved part (a). You will get the credit for this part only if your algorithm is (i) correct, and (ii) recursive. In particular, make sure that you have correctly handled all the base cases. [20]

(c) Explain why your algorithm of part (b) correctly solves Palindrome Sequence. You may assume that the function IsNTP() works correctly. [10]

(d) Write a recurrence for the *number of recursive calls* that your algorithm from part (b) makes, in terms of n. Explain why your recurrence correctly captures this number. [10]

(e) Solve your recurrence of part (d) to get an upper bound on the number of recursive calls that your algorithm makes, in terms of n. [10]

(f) Write the *complete* pseudocode for a *memoized version* of your algorithm from part (b). As in part (b), you may use IsNTP() as a black box even if you haven't solved part (a). You will get the credit for this part only if your pseudocode is a correctly memoized version of your *correct* algorithm from part (b). Make sure that you have correctly handled all the sentinel values/base cases. [20]

(g) Show that your memoized algorithm of part (f) runs in $\mathcal{O}(n^c)$ worst-case time for input strings of length n, for some fixed constant c. What is the value of c that you get? [20]

- This exam has one question for 100 marks.
- 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. 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.

A *subsequence* of an array A is any sub-array of A, obtained by deleting zero or more elements of A *without* changing the order of the remaining elements. The input to the *Subsequence Sum* problem consists of (i) an array $A[0\ldots(n-1)]$ of n positive integers, and (ii) a target integer T. The problem is to decide if there exists a subsequence B of A such that the sum of all the elements of B is exactly T. We define the sum of the *empty* subsequence (one with no elements) to be zero, and the "sum" of a subsequence with one element, to be that element.

0

1. (a) Write the *complete* pseudocode for a *non-recursive* algorithm that solves the *Subsequence Sum* problem on inputs $(A[0\ldots(n-1)], T)$ using *dynamic programming.* The algorithm should have a worst-case running time of $\mathcal{O}(n^c T^d)$ for some small constants c, d. [40]

   You will get the credit for this part only if your psuedocode is that of a non-recursive DP algorithm which correctly solves the problem within the required time bound.

   Make sure that you correctly initialize your DP table, that you check for sentinel values wherever required, and that you always compute and store the value in any cell of the table *before* you access the value in that cell for further computation.

   (b) Explain why your algorithm of part (a) correctly solves the problem. [40]

   (c) Show that the worst-case running time of your algorithm of part (a) is $\mathcal{O}(n^c T^d)$ for some small constants c, d. What are the constants that you get? [20]

- This exam has 3 questions for a total of 100 marks.
- 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. 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.

1. Recall the *Interval Scheduling* problem that we saw in class. The input is a set of n *requests* $R = \{(s(1), f(1)), (s(2), f(2)), \ldots, (s(n), f(n))\}$ where all the $s(i), f(i)$ are natural numbers, and $s(i) < f(i)$ holds for all i. The pair $(s(i), f(i))$ are the *starting time* and *finishing time* for request i. A subset of the requests is said to be *compatible* if no two of them intersect—as intervals on the real line—other than (perhaps) at their starting or finishing times. The goal is to find a largest collection of compatible intervals.

   For each part below, describe one input instance of the *Interval Scheduling* problem for which the stated greedy approach *does not* give an optimal solution. Your description of the instance should be in the form that we saw in class: intervals represented using horizontal line segments with short vertical bars at either end. Clearly mark the starting and finishing times—these must be integers—of each interval. In each case, list (i) the solution that the greedy approach computes, an (ii) an optimal solution.

   (a) Select the first available request. [10]

   (b) Select the request that takes up the least amount of time. [10]

   (c) Select the request that has conflicts with the least number of other requests. [20]

   Warning: Make sure that there is no ambiguity in your solutions. If I cannot clearly distinguish whether two intervals in your examples overlap or not, you will not get the credit for the answer.

2. Vinod plays a magical fantasy game on his mobile phone. A new version of the game will be released on Jan 1, 2025. This version has n amulets—things with special magical powers that can be used in the game—$A_1, A_2, \ldots, A_n$ for in-game purchase. The game allows a player to buy at most one amulet per week, and a player can buy a specific amulet at most once. Each amulet costs ₹100 on Jan 1, 2025, and the price of each amulet A_j increases by a multiplicative factor of $r_j > 1$ *per week*. That is: let the week of Jan 1–Jan 7, 2025 be week 0, the week of Jan 8–Jan 14 be week 1, and so

on. Then amulet A_j, for $1 \leq j \leq n$, will cost ₹$100 \times r_j^t$ if it is bought in week t. The price growth rates for different amulets are distinct, and in fact $1 < r_1 < r_2 < \cdots < r_n$.

(a) Vinod wants to buy all the n amulets. In what order should he buy these amulets to minimize his total cost? [10]

(b) Use an *exchange argument* to show that your answer of part (a) is correct. [20]

3. The input to this problem is an array A containing n integers, where n is an odd number. Array A may have repeated elements. The task is to find an element x of A such that x is *not larger than* the median element of A. Recall that the median element of A is that—unique—element which appears at the middle position in a *sorted* version of A.

   (a) Describe—in words, or using pseudocode, or both—a *randomized* algorithm that runs in $\mathcal{O}(kn)$ time and solves the problem with a probability of success at least $1 - \frac{1}{2^k}$, for any choice of integer k. [10]

   (b) Show that your algorithm of part (a) indeed runs in $\mathcal{O}(kn)$ time and succeeds with probability at least $1 - \frac{1}{2^k}$. [20]

- This exam has 3 questions for a total of 100 marks.
- Warning: CMI's academic policy regarding cheating applies to this exam.

All arrays in this question paper have their indices starting at 0. If you wish to use 1-based indexing, you must clearly state this in each such answer. The arrays in these questions are objects whose sizes are fixed, in the sense that the size cannot be changed after the array is created. In particular, these are *not* Python's lists, whose sizes can be changed using, say, append(). Also: note that Python's list.append() does *not* run in *worst-case* constant time; be mindful of this when writing your pseudocode.

Unstated assumptions and lack of clarity in solutions can and will be used against you during evaluation. Please ask the invigilators if you have questions about the questions.

1. The input to this problem is an array A of length $n \geq 1$. Each element of array A is a pair of the form (StudentID, marks) where StudentID is an alphanumeric string, and marks is a non-negative integer. The pair (StudentID, marks) being present in A means that the student whose ID is StudentID, got marks marks in some exam. There may be more than one pair with the same StudentID; each such pair denotes the marks that this particular student got in a different exam. All pairs with the same StudentID occur *consecutively* in the array. The goal is to find the *maximum* marks that each student scored, among all the exams whose marks for this student are present in A. **[30]**

   Write the *complete pseudocode* for an algorithm MAXMARKS(n, A) that **prints out** the highest marks corresponding to each StudentID that is present in array A. For each distinct StudentID present in array A, the output should have *exactly one line that lists this StudentID and the* corresponding maximum marks. The algorithm must run in $\mathcal{O}(n)$ time and take at most *constant* extra space, in the worst case. Assume that two StudentIDs can be compared for equality in constant time, using the == operator. Assume also that you can use

   - A[i][0] to access the StudentID in location i of array A in constant time,
   - A[i][1] to access the marks in location i of array A in constant time, and
   - A Python-like function print() that prints out its arguments, and then a newline, in constant time. You may use Python-like—or any other sensible—string interpolation to include values of variables in the printed-out string.

   You will get the credit for this question *only if* your solution *correctly* solves *all* valid instances of the stated problem *within the required time and space bounds.*

   You do *not* have to explain why your pseudocode is correct. You do *not* have to provide an analysis of its running time and space.

2. The input to this problem consists of an array A of $n \geq 1$ integers, and an integer x. Array A is *sorted* in non-decreasing order. The goal is to find the *number of times* that integer x appears in A. Note that this number could be zero. **[30]**

   Write the *complete pseudocode* for an algorithm COUNTER(A, x) that returns the number of times that x appears in A. The algorithm must run in $\mathcal{O}(\log_2 n)$ time in the worst case. Assume that two numbers can be compared using the operators $\{<, >, ==\}$ in constant time. Assume also that you can use $A[i]$ to access the number in location i of array A, in constant time.

You will get the credit for this question *only if* your solution *correctly* solves *all* valid instances of the stated problem *within the required time bound*.

You do *not* have to explain why your pseudocode is correct. You do *not* have to provide an analysis of its running time.

3. Recall that a palindrome is a string that reads the same in either direction. A *non-trivial palindrome* is a palindrome with length (number of characters) at least two. Consider the following problem: **[40]**

Palindrome Sequence

- Input: An integer $n \geq 2$ and a string S of length n. String S is an array indexed from 0; its elements are thus $S[0], S[1], \ldots, S[(n-1)]$.
- Output: True if S can be obtained by concatenating one or more non-trivial palindromes, and False otherwise. Equivalently: True if S can be partitioned (that is: cut up, without dropping any element, and without rearranging the pieces) into one or more non-trivial palindromes, and False otherwise.

Some examples with $n = 10$: i j

- True instances: `abacabaddd`, `fjbubjfhuh`, `mttmzizzcz`, `fjfyspsyqq`, `abcdeedcba`
- False instances: `daabbbcccd`, `azatznnzth`, `ummnurlxmv`, `xqeppajynx`, `tyjglnvmaa`

Write the *complete* pseudocode for a *non-recursive* algorithm IsNTPSequence(n, S) that solves Palindrome Sequence using *dynamic programming*. The algorithm should have a worst-case running time of $O(n^c)$ for some small constant c.

You will get the credit for this question *only if* your pseudocode is *complete*, and implements a *non-recursive DP algorithm* which *correctly* solves *all* valid instances of the problem *within the required time bound*.

Make sure that you correctly initialize your DP table, that you check for sentinel values wherever required, and that you always compute and store the value in any cell of the table *before* you access the value in that cell for further computation.

You do *not* have to explain why your pseudocode is correct. You do *not* have to provide an analysis of its running time.