- This exam has 6 questions for a total of 150 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.

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.

Some graph definitions; read these if/when you need them.

All graphs in this exam are finite and undirected. Such a graph is:

- *Simple* if it has neither multiple edges nor self-loops;
- *Connected* if there is a path between any pair of its vertices;
- *Acyclic* if it has no cycles.

Note that these definitions are *not* cumulative; a connected graph need not be simple, etc.

Let G be a simple undirected graph. A path in G is a sequence of the form $v_0, e_1, v_1, e_2, \ldots, e_t, v_t$ where (i) each v_i is a vertex and each e_j is an edge, (ii) edge e_j is incident with vertices v_{j-1} and v_j for all j, and (iii) no vertex appears more than once. This is said to be a path from v_0 to v_t. In addition, the sequence consisting of a single vertex v is a path from v to v.

A *tree* is simple, connected, acyclic graph. There is exactly one path between any two distinct vertices in a tree. A *rooted tree* is a tree in which one vertex is designated as the *root vertex*.

1. An isolated vertex in a graph is a vertex with no edge incident on it. The notation $|S|$ denotes the cardinality of set S. We say that a graph is *special* if it is simple and has no isolated vertex.

   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. [5]

   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. [5]

3. Let T be a rooted tree with root vertex r. Vertex r has no parent. For any vertex $v \neq r$ in T, the *parent* of v is the first vertex which follows v in the unique path from v to r. Let x, y be two vertices in T, and let P_x, P_y, respectively, bet the paths from x, y to r, respectively. The *least common ancestor* of x and y, denoted $LCA(x, y)$, is the first vertex that is common to both of P_x, P_y. That is, let S be the sequence obtained by removing from P_x all those vertices that do not appear in P_y. Then $LCA(x, y)$ is the first vertex that appears in S.

   (a) Let T, r be as defined above. Prove that for any two vertices x, y in T, the vertex $LCA(x, y)$ exists in T and is unique. [5]

   (b) A computer program represents each vertex v of the rooted tree T as an object with two attributes: (i) a unique identifier $v.id$, and (ii) an attribute $v.parent$ which equals the id value of the parent vertex of v. The root vertex is the unique vertex in T with a $parent$ value of NIL. [15]

Write the pseudocode for a *recursive* algorithm LCA(x, y) that takes two variables x, y whose values are vertex objects from T, and returns a vertex object corresponding to the least common ancestor of these two vertices. Note that the id of the root vertex object is *not* given as part of the input. Use the notation x.id etc., to represent attributes.

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.

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

4. The Search problem is defined as follows:

Search

- Input: An integer $n \geq 1$, an array A of n integers, and an integer v. Array A is indexed from 0; its elements are thus $A[0], A[1], \ldots, A[(n-1)]$.
- Output: An index i such that $A[i] = v$, or the special value NIL if v does not appear in A.

(a) Write the pseudocode for an algorithm that solves the Search problem in $\mathcal{O}(n)$ time—in the worst case—by scanning through the array from left to right, looking for v. You will get the credit for this part only if your algorithm is correct, and runs within the required time bound. [10]

(b) Using a *loop invariant*, prove that your algorithm of part (a) is correct. Explain why your loop invariant fulfils the three necessary properties of a useful loop invariant. [10]

(c) 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]

5. 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 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. You do *not* have to use loop invariants in this proof; but you must correctly explain why each loop (if there are some) does what you expect it to do. [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]

6. 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) $T(n) = 3T(n/4) + 5$ [15]

   (b) $T(n) = T(n/2) + 2\sqrt{n}$ [15]