1.

a.

Claim: The MST of a connected weighted undirected gragh G must include the minimum weight edge in every cycle in G

**Claim:**  
The MST of a connected weighted undirected graph G must include the minimum-weight edge in every cycle in G.

**Proof (by contradiction, using Kruskal's algorithm):**

Suppose, for contradiction, there is a cycle CCC in G, and let e be the minimum-weight edge in CCC. Assume there is a minimum spanning tree (MST) TTT that does not contain e.

Kruskal’s algorithm always selects the smallest available edge that does not form a cycle. When the algorithm processes e, if it does not create a cycle, it must be included in the MST. If e is not included, it must be because all other edges in the cycle were chosen first, but those edges are heavier, which contradicts Kruskal’s greedy strategy.

Therefore, the minimum-weight edge in any cycle must be included in the MST.

(b)

The statement is **false**.  
Having two edges with the same weight does **not necessarily** mean that there are multiple minimum spanning trees (MSTs).

**Explanation:**  
When Kruskal’s algorithm encounters two edges with the same weight (say, e1 and e2), there are two cases:

* **Case 1 (Mutually Exclusive):**  
  If the choice between e1 and e2 is mutually exclusive (that is, including e1 would make e2 form a cycle, and vice versa), then either edge could be chosen, resulting in different MSTs. In this case, multiple MSTs may exist.
* **Case 2 (Not Mutually Exclusive):**  
  If both edges must be included, or both cannot be included, or the structure of the graph forces a unique selection, then the MST is still unique even though two equal-weight edges exist.

Therefore, the existence of equal-weight edges alone does **not** guarantee multiple MSTs. Multiple MSTs only occur if these edges can be interchanged in different spanning trees while keeping the total weight minimum and the tree connected.

**Counterexample:**  
Consider a graph with four vertices A, B, C, D and edges:

AB: 1

AC: 1

AD: 2

BC: 3

BD: 3

CD: 3

Both AB and AC have the same weight, but the MST is unique: AB, AC, and AD.

**Conclusion:**  
The statement is disproved.

We use Prim’s algorithm with a min-heap (priority queue) to construct a Minimum Spanning Tree (MST). During construction, we monitor whether multiple equal-weight edges are available to connect new nodes. If such alternatives exist at any step, the MST is not unique.

1. **Initialization**:

Select an arbitrary starting node.

Create a boolean array visited[] to track which nodes have been added to the MST.

Initialize a min-heap to store edges in the format (weight, from, to).

1. **Construct the MST while checking uniqueness**:

In each step, pop from the heap all edges that share the **minimum weight**.

From those edges, select the ones that connect to **unvisited nodes**—these are the **valid candidate edges**.

**If two or more valid edges exist**:

These edges represent different ways to grow the MST without increasing its weight.

Therefore, **multiple MSTs are possible ⇒ the MST is not unique**.

If **only one valid edge** exists, add it to the MST and push all of its outgoing edges (that connect to unvisited nodes) into the heap.

If **no valid edge** exists (e.g., all edges connect to already visited nodes), skip and continue.

1. **Termination**:

If the MST includes all vertices and **no step encountered multiple valid minimal edges**, the MST is **unique**.

d)

The algorithm is correct based on the greedy nature of Prim’s algorithm:

* Prim always chooses the edge with the smallest weight that connects a new vertex to the MST.
* If at any point there are **multiple equal-weight edges** that can legally connect new nodes, any of them can be selected—yielding **distinct but valid MSTs with the same total weight**.
* The algorithm explicitly checks for this condition by examining all equal-weight edges at each step.
* If **no such alternative choices** exist throughout the construction process, the MST is **structurally unique**.

Thus, the algorithm correctly determines whether or not multiple MSTs are possible.

e)

The overall time complexity of the algorithm is O(mlogn), where m=∣E∣ is the number of edges and n=∣V∣is the number of vertices.

Step-by-step breakdown:

1. Initialization:

Initializing visited[], the result array, and the heap structure takes O(n) time.

1. Heap Insertions (Push operations):

Each edge is inserted into the heap at most once, when its starting vertex is first added to the MST.

There are O(m) insertions in total, each taking O(logn) time.

⇒ Total: O(mlogn)

1. Heap Extractions (Pop operations):

Each edge is popped at most once from the heap.

Each pop costs O(logn), and up to mmm pops occur.

⇒ Total: O(mlogn)

1. Uniqueness checks:

At each step, we may compare multiple equal-weight edges to check whether more than one can extend the MST.

These comparisons are done during heap processing and require at most O(m) total time.

Final complexity:

O(mlogn)​

This accounts for all heap operations and edge scans, and satisfies the efficiency requirement.

The algorithm assumes that the input graph is connected. If the graph is disconnected, Prim’s algorithm will produce a minimum spanning forest instead of a single spanning tree.

2.