a. Required Algorithms for Finding MST Using Kruskal’s Algorithm

Overview of Kruskal's Algorithm:

Kruskal's algorithm finds the Minimum Spanning Tree (MST) of a graph by:

1. Sorting all edges by weight.

2. Using a union-find (disjoint set) data structure to avoid cycles while adding the smallest edge to the MST.

Algorithm 1: Kruskal’s Algorithm Kruskal(G):

Input:

A graph G = (V, E), where V is the set of vertices and E is the set of edges with weights.

Output:

The edges of the Minimum Spanning Tree (MST).

```
1. Initialize MST = {}.
2. Sort all edges in E by weight (ascending).
3. Initialize a disjoint set for all vertices in V.
4. For each edge (u, v) in the sorted edge list:
   a. If u and v are not in the same set:
      i. Add (u, v) to MST.
      ii. Union the sets of u and v.
5. Return MST.
```

Algorithm 2: Disjoint Set Operations

- Find: Determines the representative of the set containing an element.

- Union: Combines two sets.

```
MakeSet(v):
Input: A vertex v.
Output: Initializes a new set containing v.
1. Parent[v] = v.
2. Rank[v] = 0.
```

```
Find(v):
Input: A vertex v.
Output: The representative of the set containing v.
1. If Parent[v] $\neq$ v:
    a. Parent[v] = Find(Parent[v]) (path compression).
2. Return Parent[v].
```

```
Union(u, v):
Input: Two vertices u and v.
Output: Merges the sets containing u and v.
1. RootU = Find(u).
2. RootV = Find(v).
3. If RootU $\neq$ RootV:
    a. If Rank [RootU] > Rank [RootV], Parent [RootV] = RootU.
    b. Else if Rank [RootU] < Rank [RootV], Parent [RootU] = RootV.
    c. Else, Parent [RootV] = RootU and increment Rank [RootU].
```

---

b. Analysis of Kruskal's Algorithm

Time Complexity:

1. Sorting the Edges:

   - Sorting all edges E takes O(E log E).

2. Union-Find Operations:

   - For E edges, the union and find operations are performed.

   - Using path compression and union by rank, each operation runs in nearly constant time O(alpha(V)), where alpha is the inverse Ackermann function.

   - Total time: O(E alpha(V)).

Overall time complexity: O(E log E + E alpha(V)) approx O(E log E).

Space Complexity:

- Storing the edges and disjoint set requires O(V + E) space.