A. Writing All Required Algorithms

Kruskal's algorithm is a Greedy Algorithm used to find the Minimum Spanning Tree (MST) of a weighted graph. It relies on two main helper functions:

1. find_set Function:
This function identifies the root (or representative) of a subset using Path Compression to optimize future searches.

2. union_set Function:
This function merges two subsets into a single subset using Union by Rank to maintain a balanced tree structure.

3. Kruskal's Algorithm Steps:

Sort all the edges of the graph in ascending order based on their weights.

Initialize each vertex as its own parent (disjoint sets).

For each edge, check if the vertices of the edge belong to different sets using find_set.

If they do, include the edge in the MST and unite the sets using union_set.

Repeat until the MST contains V-1 edges, where V is the number of vertices.

B. Analysis of the Algorithms

Time Complexity:

1. Sorting the edges: Sorting the edges takes O(E log E), where E is the number of edges.

2. Union-Find Operations: Using Path Compression and Union by Rank, the time complexity for find_set and union_set is almost constant, amortized as O(log V).

3. Iterating through edges: The algorithm processes each edge once, taking O(E log V) for all union-find operations combined.

Thus, the overall time complexity of Kruskal's algorithm is:
O(E log E), as edge sorting dominates the complexity.

Space Complexity:

Parent Array: O(V), to store the representative of each subset.

Rank Array: O(V), to store the rank (height) of each tree.

MST Storage: O(V), to store the edges that form the Minimum Spanning Tree.

The overall space complexity is: O(V + E).

c:Implimentation of Algorithms

```
def find_set(parent, i):
    if parent[i] != i:
        parent[i] = find_set(parent, parent[i])
    return parent[i]

def union_set(parent, rank, x, y):
    rootX = find_set(parent, x)
    rootY = find_set(parent, y)

    if rootX != rootY:
        if rank[rootX] > rank[rootY]:
            parent[rootY] = rootX
        elif rank[rootX] < rank[rootY]:
            parent[rootX] = rootY
        else:
            parent[rootY] = rootX
            rank[rootX] += 1

def kruskal(vertices, edges):
    edges.sort(key=lambda edge: edge[2])

    parent = [i for i in range(vertices)]
    rank = [0] * vertices
    mst = []

    for edge in edges:
        u, v, weight = edge
        if find_set(parent, u) != find_set(parent, v):
            mst.append(edge)
            union_set(parent, rank, u, v)

    return mst

vertices = 4
edges = [
    (0, 1, 10),
    (0, 2, 6),
    (0, 3, 5),
    (1, 3, 15),
    (2, 3, 4)
]

mst = kruskal(vertices, edges)

print("Edges in the MST are:", mst)
```

Output:

```
Edges in the MST are:
[(2, 3, 4), (0, 3, 5), (0, 1, 10)]
```