94. Minimum Spanning Tree

PROGRAM:-

```
class DisjointSet:
    def _init_(self, n):
        self.parent = list(range(n))
        self.rank = [0] * n

    def find(self, u):
        if self.parent[u] != u:
            self.parent[u] = self.find(self.parent[u])
        return self.parent[u]

    def union(self, u, v):
        root_u = self.find(u)
        root_v = self.find(v)

        if root_u != root_v:
            if self.rank[root_u] > self.rank[root_v]:
                self.parent[root_v] = root_u
            elif self.rank[root_u] < self.rank[root_v]:
                self.parent[root_u] = root_v
            else:
                self.parent[root_v] = root_u
                self.rank[root_u] += 1

def kruskal(n, edges):
    # Sort edges by weight
    edges.sort(key=lambda x: x[2])
    disjoint_set = DisjointSet(n)
    mst = []
    mst_cost = 0

    for u, v, weight in edges:
        if disjoint_set.find(u) != disjoint_set.find(v):
            disjoint_set.union(u, v)
            mst.append((u, v, weight))
            mst_cost += weight

    return mst, mst_cost

# Example usage:
n = 4
edges = [
    (0, 1, 10),
    (0, 2, 6),
    (0, 3, 5),
    (1, 3, 15),
    (2, 3, 4)
]
```

```
mst, mst_cost = kruskal(n, edges)
print("Edges in MST:", mst)
print("Total cost of MST:", mst_cost)
```

OUTPUT:-

```
Edges in MST: [(2, 3, 4), (0, 3, 5), (0, 1, 10)]
Total cost of MST: 19

=== Code Execution Successful ===
```

TIME COMPLEXITY:- O(E log E+E log V)