```
class Edge:
    def __init__(self, u, v, weight):
        self.u = u
        self.v = v
        self.weight = weight

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


class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.edges = []


    def add_edge(self, u, v, weight):
        self.edges.append(Edge(u, v, weight))


    def kruskal_mst(self):

         self.edges.sort(key=lambda edge: edge.weight)

        mst = []
        ds = DisjointSet(self.vertices)

        for edge in self.edges:
            u = edge.u
            v = edge.v


            if ds.find(u) != ds.find(v):
                mst.append(edge)
                ds.union(u, v)
        return mst


if __name__ == "__main__":

    g = Graph(6)
```

```
g.add_edge(0, 1, 4)
g.add_edge(0, 2, 4)
g.add_edge(1, 2, 2)
g.add_edge(1, 3, 5)
g.add_edge(2, 3, 5)
g.add_edge(2, 4, 6)
g.add_edge(3, 4, 8)
g.add_edge(3, 5, 7)
g.add_edge(4, 5, 9)


mst = g.kruskal_mst()

print("Edges in the Minimum Spanning Tree:")
for edge in mst:
     print(f"{edge.u} -- {edge.v} == {edge.weight}")


 Edges in the Minimum Spanning Tree:
 1 -- 2 == 2
 0 -- 1 == 4
 1 -- 3 == 5
 2 -- 4 == 6
 3 -- 5 == 7
```