```
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])  # Path compression
        return self.parent[u]

    def union(self, u, v):
        u_root = self.find(u)
        v_root = self.find(v)
        if u_root == v_root:
            return False  # Cycle detected
        # Union by rank
        if self.rank[u_root] < self.rank[v_root]:
            self.parent[u_root] = v_root
        elif self.rank[u_root] > self.rank[v_root]:
            self.parent[v_root] = u_root
        else:
            self.parent[v_root] = u_root
            self.rank[u_root] += 1
        return True

def kruskal_mst(vertices, edges):
    # Sort all edges based on weight
    edges.sort()
    ds = DisjointSet(vertices)
    mst = []
    total_cost = 0

    for weight, u, v in edges:
        if ds.union(u, v):
            mst.append((u, v, weight))
            total_cost += weight
    return mst, total_cost


# Sample graph input
if __name__ == "__main__":
    vertices = 5
    edges = [
        (1, 0, 1),
        (3, 0, 2),
        (2, 1, 2),
        (4, 1, 3),
        (5, 2, 3),
        (7, 3, 4),
        (6, 2, 4)
    ]
    mst, cost = kruskal_mst(vertices, edges)
    print("Edges in Minimum Spanning Tree (MST):")
    for u, v, weight in mst:
        print(f"{u} - {v} : {weight}")


    print(f"\nTotal cost of MST: {cost}")
#OUTPUT
```

```
Edges in Minimum Spanning Tree (MST):
0 - 1 : 1
1 - 2 : 2
1 - 3 : 4
2 - 4 : 6

Total cost of MST: 13
```