Experiment 7

Kruskal's MST Algorithm

Program:

```
#include <stdio.h>
#include <stdlib.h>

// Comparator function to use in sorting
int comparator(const void* p1, const void* p2)
{
	const int(*x)[3] = p1;
	const int(*y)[3] = p2;

	return (*x)[2] - (*y)[2];
}

// Initialization of parent[] and rank[] arrays
void makeSet(int parent[], int rank[], int n)
{
	for (int i = 0; i < n; i++) {
		parent[i] = i;
		rank[i] = 0;
	}
}

// Function to find the parent of a node
int findParent(int parent[], int component)
{
	if (parent[component] == component)
		return component;

	return parent[component]
		= findParent(parent, parent[component]);
}

// Function to unite two sets
void unionSet(int u, int v, int parent[], int rank[], int n)
{
	// Finding the parents
	u = findParent(parent, u);
	v = findParent(parent, v);
```

```
	if (rank[u] < rank[v]) {
		parent[u] = v;
	}
	else if (rank[u] > rank[v]) {
		parent[v] = u;
	}
	else {
		parent[v] = u;

		// Since the rank increases if
		// the ranks of two sets are same
		rank[u]++;
	}
}

// Function to find the MST
void kruskalAlgo(int n, int edge[n][3])
{
	// First we sort the edge array in ascending order
	// so that we can access minimum distances/cost
	qsort(edge, n, sizeof(edge[0]), comparator);

	int parent[n];
	int rank[n];

	// Function to initialize parent[] and rank[]
	makeSet(parent, rank, n);

	// To store the minimun cost
	int minCost = 0;

	printf(
		"Following are the edges in the constructed MST\n");
	for (int i = 0; i < n; i++) {
		int v1 = findParent(parent, edge[i][0]);
		int v2 = findParent(parent, edge[i][1]);
		int wt = edge[i][2];

		// If the parents are different that
		// means they are in different sets so
		// union them
		if (v1 != v2) {
			unionSet(v1, v2, parent, rank, n);
			minCost += wt;
```

```
                        printf("%d -- %d == %d\n", edge[i][0],
                                edge[i][1], wt);
                }
        }

        printf("Minimum Cost Spanning Tree: %d\n", minCost);
}

// Driver code
int main()
{
        int edge[5][3] = { { 0, 1, 10 },
                                        { 0, 2, 6 },
                                        { 0, 3, 5 },
                                        { 1, 3, 15 },
                                        { 2, 3, 4 } };

        kruskalAlgo(5, edge);

        return 0;
}
```

Output: