**Code:**

#include <bits/stdc++.h>

using namespace std;

struct Edge {

int src, dest, weight;

};

bool compareEdges(const Edge &a, const Edge &b) {

return a.weight < b.weight;

}

void kruskalMST(vector<Edge> edges, int V) {

sort(edges.begin(), edges.end(), compareEdges);

vector<int> parent(V);

vector<int> rank(V, 0);

for (int i = 0; i < V; i++) {

parent[i] = i;

}

vector<Edge> result;

for (Edge edge : edges) {

int x = edge.src;

int y = edge.dest;

while (parent[x] != x)

x = parent[x];

while (parent[y] != y)

y = parent[y];

if (x != y) {

result.push\_back(edge);

if (rank[x] < rank[y])

parent[x] = y;

else if (rank[x] > rank[y])

parent[y] = x;

else {

parent[y] = x;

rank[x]++;

}

}

}

cout << "Kruskal's Algorithm - Edges in Minimum Spanning Tree:" << endl;

for (Edge edge : result) {

cout << edge.src << " - " << edge.dest << " : " << edge.weight << endl;

}

}

void primMST(vector<vector<int>> &graph, int V) {

vector<int> parent(V, -1);

vector<int> key(V, INT\_MAX);

key[0] = 0;

vector<bool> inMST(V, false);

for (int count = 0; count < V - 1; count++) {

int u = -1;

for (int v = 0; v < V; v++) {

if (!inMST[v] && (u == -1 || key[v] < key[u]))

u = v;

}

inMST[u] = true;

for (int v = 0; v < V; v++) {

if (graph[u][v] && !inMST[v] && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

}

cout << "\nPrim's Algorithm - Edges in Minimum Spanning Tree:" << endl;

for (int i = 1; i < V; i++) {

cout << parent[i] << " - " << i << " : " << graph[i][parent[i]] << endl;

}

}

int main() {

auto start = chrono::steady\_clock::now();

int V = 4;

vector<Edge> edges = {

{0, 1, 10},

{0, 2, 6},

{0, 3, 5},

{1, 3, 15},

{2, 3, 4}

};

kruskalMST(edges, V);

vector<vector<int>> graph = {

{0, 10, 6, 5},

{10, 0, 0, 15},

{6, 0, 0, 4},

{5, 15, 4, 0}

};

primMST(graph, V);

auto end = chrono::steady\_clock::now();

auto diff = end - start;

cout <<"Execution time: "<< chrono::duration <double, milli> (diff).count() << " ms\n";

return 0;

}

**Output:**

