Name:-Dikesh Ganboi
Roll NO:- A36

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
#include <chrono>

using namespace std;
using namespace std::chrono;

// Structure to represent an edge in the graph
struct Edge {
    int dest; // Destination vertex
    int weight; // Weight of the edge
};

// Structure to represent a graph
class Graph {
    int V; // Number of vertices
    vector<vector<Edge>> adj; // Adjacency list to store edges

public:
    Graph(int V) : V(V), adj(V) {}

    // Function to add an edge to the graph
    void addEdge(int src, int dest, int weight) {
        adj[src].push_back({dest, weight});
        adj[dest].push_back({src, weight}); // For undirected graph
    }

    // Function to find the MST using Prim's Algorithm
    void primMST() {
        vector<int> key(V, INT_MAX); // Key values used to pick the minimum
weight edge
        vector<bool> inMST(V, false); // To represent vertices included in MST
        vector<int> parent(V, -1); // To store parent array which stores MST

        // Priority queue to store vertices based on key values
        priority_queue<pair<int, int>, vector<pair<int, int>>,
greater<pair<int, int>>> pq;

        // Start with vertex 0
        int src = 0;
        key[src] = 0;
        pq.push({0, src});

        while (!pq.empty()) {
            int u = pq.top().second;
            pq.pop();

            // Mark vertex as included in MST
            inMST[u] = true;

            // Update key value and push adjacent vertices not included in MST
            for (const auto& edge : adj[u]) {
                int v = edge.dest;
                int weight = edge.weight;
                if (!inMST[v] && weight < key[v]) {
                    key[v] = weight;
                    parent[v] = u;
                    pq.push({key[v], v});
                }
            }
        }

        // Print MST
        cout << "Edges in Minimum Spanning Tree:" << endl;
        for (int i = 1; i < V; ++i)
            cout << parent[i] << " - " << i << endl;
    }
};

int main() {
    // Create a graph
    int V = 5; // Number of vertices
    Graph g(V);

    // Add edges to the graph
    g.addEdge(0, 1, 2);
    g.addEdge(0, 3, 6);
    g.addEdge(1, 2, 3);
    g.addEdge(1, 3, 8);
    g.addEdge(1, 4, 5);
    g.addEdge(2, 4, 7);
    g.addEdge(3, 4, 9);

    // Measure time taken by Prim's Algorithm
    auto start = high_resolution_clock::now();
    g.primMST();
    auto stop = high_resolution_clock::now();
```

```cpp
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "Time taken by Prim's Algorithm: " << duration.count() << " microseconds" << endl;

    return 0;
}
```

OUTPUT:-

```
PS C:\Users\HP\Desktop\DAA EXperiment> cd "c:\Users\HP\Desktop\DAA EXperiment\" ; if ($?) { g++ primsalgo.cpp -o primsalgo } ; if ($?) { .\primsalgo }

Edges in Minimum Spanning Tree:
0 - 1
1 - 2
0 - 3
1 - 4
Time taken by Prim's Algorithm: 0 microseconds
```