Q3 Implement Dijkstra's Algorithm. Analyze its time complexity.

TIME COMPLEXITY:
1. O(E + V * logV) using Fibonacci Heaps.
2. O(V^2 + E * logV) using Matrix Representation and Priority Queue.
3. O((V + E) * log V) using Adjacency list and Priority Queue. (this one is used below)

SPACE COMPLEXITY: O(V)

where V is the number of vertices of the graph and E is the number of the edges.

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

using namespace std;

class Vertex
{
public:
    int index;
    int distance;
    bool visited;
};
class Comp
{
public:

    bool operator()(const Vertex& v1, const Vertex& v2)
    {
        return v1.distance > v2.distance;
    }

};
void dijkstras(int ** edges, int v, Vertex * input)
{
    int count = 0;
    int j = 0;
    int sum = 0;
    while (count != v - 1)
    {
        priority_queue<Vertex, vector<Vertex>, Comp> queue;
        for (int i = 0; i < v; i++)
        {
            if (edges[i][j])
            {
                if (input[i].visited)
                {
                    continue;
                }
                if (sum + edges[i][j] < input[i].distance)
```

```
                {
                    input[i].distance = sum + edges[i][j];
                }
            }
        }
        for (int i = 0; i < v; i++)
        {
            if (!input[i].visited)
            {
                queue.push(input[i]);
            }
            else continue;
        }
        j = queue.top().index;
        input[j].visited = true;
        sum = queue.top().distance;
        count++;
    }
    cout << "SOURCE VERTEX: " << 0 << endl;
    cout << "DISTANCES: \nVERTEX\tDISTANCE" << endl;
    for (int i = 0; i < v; i++)
    {
        cout << "  " << input[i].index << "\t\t  " << input[i].distance <<
         endl;
    }
}

int main()
{
    cout << "ENTER THE NUMBER OF VERTICES AND EDGES: ";
    int v, e;
    cin >> v >> e;
    int ** edges = new int*[v];
    for (int i = 0; i < v; i++)
    {
        edges[i] = new int[v];
        for (int j = 0; j < v; j++)
            edges[i][j] = 0;
    }
    cout << "INPUT GRAPH: " << endl;
    for (int i = 0; i < e; i++)
    {
        int s, d, w;
        cin >> s >> d >> w;
        edges[s][d] = w;
        edges[d][s] = w;
    }
    Vertex * input = new Vertex[v];
    for (int i = 0; i < v; i++)
    {
        input[i].index = i;
        if (i)
        {
            input[i].distance = INT_MAX;
            input[i].visited = false;
```

```
        }
        else
        {
            input[i].visited = true;
            input[i].distance = 0;
        }

    }
    dijkstras(edges, v, input);
    return 0;
}
```

OUTPUT:

```
ENTER THE NUMBER OF VERTICES AND EDGES: 7 10
INPUT GRAPH:
0 1 6
0 2 5
0 3 5
2 3 2
1 2 2
4 6 3
5 6 3
3 5 1
2 4 11
2 4 1
SOURCE VERTEX: 0
DISTANCES:
VERTEX    DISTANCE
  0          0
  1          6
  2          5
  3          5
  4          6
  5          6
  6          9
Program ended with exit code: 0
```