Q8. Implement Bellman Ford Algorithm. Analyze its time complexity.

TIME COMPLEXITY: O(V * E)

SPACE COMPLEXITY: O(V)

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

```
#include <iostream>
#include <queue>
#include <vector>
#include <string>
#include <map>
#include <cmath>

using namespace std;

class Edge
{
public:
    int source, destination, weight;
    Edge(int source, int destination, int weight) : source(source),
     destination(destination), weight(weight) {}
};
class Graph
{
public:
    int V, E;
    vector<Edge> edges;
    vector<int> distance;
    Graph(int V, int E) : V(V), E(E), distance(vector<int>(V, INT_MAX))
    {
        edges.reserve(E);
    }
    void addEdge(int source, int destination, int weight)
    {
        edges.push_back(Edge(source, destination, weight));
    }
    void print()
    {
        for(int i = 0; i < this -> E; ++i)
            cout << edges[i].source << "\t" << edges[i].destination << "\t"
             << edges[i].weight << endl;
    }
    void printGraph()
    {
        cout << "VERTEX\tDISTANCE" << endl;
        for(int i = 0; i < this -> V; ++i)
            cout << "  " << i << "\t\t  " << distance[i] << endl;
    }
    void bellmanFord(int source)
    {
        distance[source] = 0;
        for (int i = 1; i <= V - 1; i++)
```

```
            for (int j = 0; j < E; j++)
            {
                int u = edges[j].source;
                int v = edges[j].destination;
                int weight = edges[j].weight;
                if (distance[u] != INT_MAX)
                    distance[v] = min(distance[u] + weight, distance[v]);
            }
        for (int i = 0; i < E; i++)
        {
            int u = edges[i].source;
            int v = edges[i].destination;
            int weight = edges[i].weight;
            if (distance[u] != INT_MAX && distance[u] + weight <
             distance[v])
            {
                cout << "Graph contains negative weight cycle" << endl;
                return;
            }
        }
        printGraph();
    }
};
int main()
{
    Graph graph(5, 8);
    graph.addEdge(0, 1, -1); graph.addEdge(0, 2, 4); graph.addEdge(1, 3,
     2); graph.addEdge(1, 2, 3); graph.addEdge(1, 4, 2); graph.addEdge(3,
     2, 5); graph.addEdge(3, 1, 1); graph.addEdge(4, 3, -3);
    cout << "INPUT GRAPH: " << endl;
    graph.print();
    graph.bellmanFord(0);
    return 0;
}
```

OUTPUT:

```
INPUT GRAPH:
    0    1    -1
    0    2    4
    1    3    2
    1    2    3
    1    4    2
    3    2    5
    3    1    1
    4    3    -3

VERTEX    DISTANCE
  0          0
  1          -1
  2          2
  3          -2
  4          1
Program ended with exit code: 0
```