Algorithms: Assignment 2
Shubham Saxena
ss4017@rit.edu

Instructions to execute programs:
All inputs are taken in standard input.
Input must be in the following format:
All algorithms take weighted – directed or weighted undirected graphs.
All the algorithms can be made directed (or undirected) by just modifying one line, the one that assigns the weight of {vertex-1 to vertex-2} to {vertex-2 to vertex-1} as well.

1. Kruskal's Algorithm for Minimum Spanning Tree:
It is a greedy algorithm to find the minimum spanning tree. This algorithm needs weighted directed or undirected graph. Complexity not affected by the directed or undirected nature of the graph. If un-weighted input is to be given, all weights can be given as 1.
Time Complexity of the algorithm is O(|E| log |V|)
Reading input: O(E)
Sorting O(E log E)
For all E till count (V-1) : Ω(E).

O(log V) for number of sets.
O(1)- adding new edge
O(1)-
O(1)

For sorting the edges we use a sort of O(|E| log(E)) complexity. So final complexity is O(|E| log |V|).
Where E is the number of edges and V is the number of vertices.
Time taken

2. Dijkstra's Single Source Shortest Paths:
It needs weighted directed or undirected graph.
It is a greedy algorithm.

Complexity: O(|V^2|)
For all vertices θ (v)
For all vertices (connected, ie degree of vertex) θ (v)
Find min O(1)
Update output O(1)

Formula: output[vertex a]= min (output[vertex a], (output[b] + inputcost [a][b]))
Base case output [vertex 0]=0;
This is because all the pairs of vertices are visited once (two for loops).
If I were to use a priority queue then the complexity would be log |V|.

3. Floyd Warshall's All Pairs Shortest Paths:
This algorithm uses dynamic programming approach.
Needs a weighted directed or undirected graph.

Complexity: O(n^3)
Reading O(n^2).
base case calculation : O(n^2)
filling dynamic array O(n^3)
Dynamic arrays: S[i][j][k] = min cost from vertex I to vertex j uncluding k number of vertices.
Formula: S[i][j][k]= min (S[i][j][k-1], S[i][k][k-1] + S[k][j][k-1]))
Result: S[i][j][n]= min path from I to j including all the vertices.

4. Transitive Closure:
This is a dynamic programming algorithm.
Finds if there is a path from a vertex to any other vertex in the graph, directly or indirectly.
Complexity: O(n^3)

Dynamic arrays: S[i][j][k] = min cost from vertex I to vertex j including k number of vertices.

Time table: (all values in Milliseconds)

Time taken	Kruskals (directed)	Kruskal (undirected)	Djisktra's	Floyd Warshal	Transitivity Closure
NodeCon <10 v= 40	2.2	3.0	14.8	15.5	6.4
NodeCon < 10 v= 60	3.1	3.9	16.0	16.3	8.7
NodeCon <10 v= 100	6.0	7.8	42.9	41.2	11
NodeCon >n/2 v= 40	2.2	2.8	15.3	11.5	4.8
NodeCon >n/2 v= 60	7.0	3.1	75.1	65.4	34.2
NodeCon >n/2 v= 100	9.0	6.2	70.2	78.1	44.0
NodeCon random v= 40	1.0	4.0	15.1	15.0	5.2

NodeCon random v= 60	2.5	4.0	15.2	16.5	8.4
NodeCon random v= 100	5.0	8.1	42	43.2	12.4

Conclusion:

All the above algorithms were understood, implemented and tested successfully. Greedy and Dynamic programming concepts were used.