**1.)**

|  |  |
| --- | --- |
| 1: | **function** Dijkstra(Graph, source): |
| 2: | **for each** vertex v in Graph: | // Initialization |
| 3: | dist[v] := infinity | // initial distance from source to vertex v is set to infinite |
| 4: | previous[v] := undefined | // Previous node in optimal path from source |
| 5: | dist[source] := 0 | // Distance from source to source |
| 6: | Q := the set of all nodes in Graph | // all nodes in the graph are unoptimized - thus are in Q |
| 7: | **while** Q **is not** empty: | // main loop |
| 8: | u := node in Q with smallest dist[ ] |  |
| 9: | remove u from Q |  |
| 10: | **for each** neighbor v of u: | // where v has not yet been removed from Q. |
| 11: | alt := dist[u] + dist\_between(u, v) |  |
| 12: | **if** alt < dist[v] | // Relax (u,v) |
| 13: | dist[v] := alt |  |
| 14: | previous[v] := u |  |
| 15: | **return** previous[ ] |  |

**2.)**

Initialization of the nodes and making the heap takes O(n) time. Now for every node V, we extra that node in O(logn) time and we relax on its edges in O(logn) for each edge.

O(n) + sum((logn) + mO(logn))

sumO(logn) + sum\*mO(logn)

overall : (n+m)(logn) 🡺 O(mlogn)

**3.)**

The runtime will be O(V^2) for adjacency matrix.

In the matrix representation, there is one row and one column for each vertex. Position i,j contains 1 if there is an edge from vertex i to vertex j and 0 if it doesn’t. The size of the whole matrix is |V|^2. Thus the complexity is |V|^2 because each position in the matrix is visited once to transverse through the vertex’s edges.