1. Run Dijkstra's shortest path algorithm in the following graph with vertex S as the source, and fill the table below with the results.

```
<table>
<thead>
<tr>
<th>Vertex</th>
<th>Distance</th>
<th>Predecessor</th>
<th>Processed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

![Graph Diagram]
2. Following is the pseudocode for Dijkstra's shortest path algorithm with binary heap. Do runtime analysis of this code. (Annotate the pseudocode with big-$O$ values next to relevant statements/loops in the code.)

```plaintext
1: function Dijkstra(Graph G, Vertex source) ▷ with MPQ
2:     initialize distances to $\infty$
3:     source.dist = 0
4:     mark all vertices unprocessed
5:     initialize MPQ as a min priority queue
6:     add source with priority 0
7:     while MPQ is not empty do
8:         u = MPQ.getMin()
9:             for each edge (u,v) leaving u do
10:                if u.dist + w(u,v) < v.dist then
11:                    if v.dist == $\infty$ then
12:                        MPQ.insert(v, u.dist + w(u,v))
13:                    else
14:                        MPQ.decreasePriority(v, u.dist + w(u,v))
15:                end if
16:         end for
17:     mark u as processed
18: end while
19: end function
```