1. (a) Assuming that $G$ is represented by an adjacency matrix $A[1..n, 1..n]$, give a $\Theta(n^2)$-time algorithm to compute the adjacency list representation of $G$. (Represent the addition of an element $v$ to a list $l$ using pseudocode by $l \leftarrow l \cup \{v\}$.)

   (b) Assuming that $G$ is represented by an adjacency list $\text{Adj}[1..n]$, give a $\Theta(n^2)$-time algorithm to compute the adjacency matrix of $G$.

2. In Dijkstra's algorithm, at each iteration the nodes $y$ in $Q$ that are adjacent to the node $x$ being processed, may get new, potential shortest path distances. In other words, the array $D$ is possibly updated for those vertices.

   Draw a directed, weighted graph $G = (V, E)$ on four vertices such that at each iteration *all* of the nodes in $Q$ get new, potential shortest path distances. In other words, the array $D$ is updated for every vertex in $Q$. To keep it simple, $G$ should have no cycles.

3. Let $G = (V, E)$ be a directed, weighted graph with weight function $w : E \to \{0, 1, 2, \ldots, s\}$ for some nonnegative integer $s$.

   (a) Modify Dijkstra's algorithm to compute the shortest paths from a given source vertex $s$ in time $O(m + sn)$.

   (b) Modify your algorithm from Part (a) to run in time $O((m + n) \log s)$. Hint: How many distinct shortest-path potential distances can there be in $Q$ at any given point in time?

4. The *optimization version* of the *Longest Path Problem* is: find the longest, simple path and its weight in a directed, weighted graph $G = (V, E)$. (It might not traverse every vertex.) The *decision version* is: Given a weighted, directed graph $G = (V, E)$ and a bound $B$, does $G$ have a simple path of weight $B$ or more?

   (a) Show that the decision version is in **NP**. Make sure to state what the certificate is, and to show that the verification is in polynomial time.

   (b) Show that if you can solve the optimization problem in polynomial time, then you can solve the decision version in polynomial time.

   (c) Show that if you can solve the decision version in polynomial time, then you can solve the optimization problem in polynomial time.

5. (**Challenge problem – will not be graded.**) Generalize Problem (2) to an arbitrary number of vertices $n$.