1. Suppose we wish to compute shortest paths in a complete directed graph (a directed graph in which there exists an edge in each direction between every two vertices), with positive edge weights (so that Dijkstra's algorithm may be used). But rather than using a complicated priority queue data structure, we use an unsorted list L of the vertices that have not yet been processed. That is, the simplified version of Dijkstra's algorithm performs the following steps:

```
initialize the D and P information used by the relaxation algorithms
for single source shortest paths

initialize L to be a list of all the vertices in the graph

while L is not empty:
    look at all of the vertices in L to find the vertex v with
    the minimum value of D[v]

    remove v from L

    for each edge v->w
        relax(v->w)
```

   You may assume that looking at all vertices in L takes time proportional to the number of vertices examined, and that removing v from L takes constant time.

   (a) What is the running time of this algorithm, using O-notation, as a function of the number n of vertices in the input graph?

   Solution: Dijkstra's algorithm runs in $O(n \cdot \text{extract-min} + m \cdot \text{update-priority})$. So on a complete directed graph, which has $m = O(n^2)$ edges, and using an unsorted list which takes $O(n)$ time per extract-min operation and constant time per update-priority operation, that comes out to $O(n^2)$.

   (b) Would this algorithm be a better or worse choice than the more usual form of Dijkstra's algorithm using a binary heap, for this type of graph? Explain your answer.

   Solution: Better since the running time using a binary heap is $O(n^2 \log n)$. (Both operations in a binary heap are $O(\log n)$.)

2. Suppose that a directed graph G has the property that every shortest path from the starting vertex s to every other vertex has at most four edges. What would this fact imply about the running time of the Bellman-Ford algorithm for finding shortest paths starting from s in G?

Solution: Bellman-Ford stops updating any D-values after four iterations. So with the optimization that checks whether any D-values were changed after each iteration and stops if there were no changes, the Bellman-Ford algorithm runs five iterations of the outer-loop. There are m relaxations in each loop, so the running time is $O(m)$.

3. Let G be a directed path with six vertices and five edges, each of which has length -1. Show the values of $h(v)$ for each vertex v, and of the new edge weights for each edge, that would be calculated by Johnson's algorithm on this graph.

Solution: If the graph is $1 \to 2 \to 3 \to 4 \to 5 \to 6$, then $h(i) = i - 1$ and all new edge weights are 0.

4. Suppose you are given a directed graph (containing cycles) in which all edges have weight 1. Name an algorithm that can solve the single-source shortest path problem in this graph in linear time, without using a priority queue data structure.

Solution: Do a BFS traversal and augment it with two dictionaries, one to keep track of the predecessor of each vertex, the other to keep track of the distance of each vertex.
The modifications are as follows:

- Initialize all predecessors and distances to None. Set the distance of the source to 0.
- When visiting a vertex, set its distance to be one more than that of its predecessor, and set the predecessors of its children to itself.