1. Given an adjacency-list representation of a directed graph, where each vertex maintains an array of its outgoing edges (but *not* its incoming edges), how long does it take, in the worst case, to compute the in-degree of a given vertex?

   As usual, we use n and m to denote the number of vertices and edges, respectively, of the given graph. Also, let k denote the maximum in-degree of a vertex. (Recall that the in-degree of a vertex is the number of edges that enter it.)

   (a) $\theta(m)$

2. Consider the following problem: given an undirected graph G with n vertices and m edges, and two vertices s and t, does there exist at least one s-t path?

   If G is given in its adjacency list representation, then the above problem can be solved in O(m + n) time, using BFS or DFS. (Make sure you see why this is true.)

   Suppose instead that G is given in its adjacency **matrix**. What running time is required, in the worst case, to solve the computational problem stated above? (Assume that G has no parallel edges.)

   (a) **Answer:** $\theta(n^2)$
   Explanation: One line: when you run BFS/DFS on a list, you touch each vertex once and each edge at most twice $\longrightarrow$ time $O(m+n)$.
   For the lower bound, observe that you might need to look at every entry of the adjacency matrix (e.g., if it has only one "1" and the rest are zeroes). One easy way to prove the upper bound is to first build an adjacency list representation (in $\theta(n^2)$ time, with a single scan over the given adjacency matrix) and then run BFS or DFS as in the video lectures.

   The adjacency list representation yields asymptotically faster BFS/DFS on sparse graphs, since even in the worst case
   $$\Theta(m+n) = \Theta(n+n) = \Theta(n) \ < \ \Theta(n^2),$$
   where the $\Theta(n^2)$ bound corresponds to scanning all rows in an $n \times n$ adjacency matrix.
   Quick summary:
   i. **Adjacency List:** BFS/DFS = $\theta(O^{m+n})$
   ii. **Adjacency Matrix:** BFS/DFS = $\theta(O^{n^2})$

3. Consider the following problem: given an undirected graph G with n vertices and m edges, and two vertices s and t, does there exist at least one s-t path?

   (a)

4. Consider our algorithm for computing a topological ordering that is based on depth-first search (i.e., NOT the "straightforward solution"). Suppose we run this algorithm on a graph G that is NOT directed acyclic. Obviously it won't compute a topological order (since none exist). Does it compute an ordering that minimizes the number of edges that go backward? For example, consider the four-node graph with the six directed edges (s, v), (s, w), (v, w), (v, t), (w, t), (t, s). Suppose the vertices are ordered s, v, w, t. Then there is one backwards arc, the (t, s) arc. No ordering of the vertices has zero backwards arcs, and some have more than one.

   (a) **Answer:** Sometimes yes, sometimes no **Explanation:**

5. **Programming Assignment 4**

   (a) **Part 1: Topological Sort**
   Hello
   Completed Solution