**Question 1** (Topological Sort, 30 points). *Give a topological order of the DAG below:*

Running DFS on the graph we obtain the following pre- and post-orders: Putting the vertices in reverse post-order, we get the topological ordering: I,G,F,K,B,H,J,A,E,L,C,D.

Note that other orderings are possible.
Question 2 (Minimizing Bad Edges, 35 points). Let $G$ be an undirected graph, some of whose edges are designated as bad. Give an algorithm that given $G$ and two vertices $v$ and $w$ returns the smallest number of bad edges in any path from $v$ to $w$. For full credit, your algorithm should run in linear time.

We can solve this problem in the following way:

- Let $G_g$ be the graph consisting of the good edges of $G$. Run DFS on $G_g$, computing its connected components.
- Construct a new graph, $C$ whose vertices are the connected components of $G_g$, where there is an edge between two vertices in $C$ if there is a (necessarily bad) edge between two vertices of the corresponding components in $G$.
- Run BFS on $C$ to find the distance between $v$'s component and $w$'s component.

It is easy to see that every step here is linear time. Constructing the graphs requires enumerating the edges of $G$, creating edges in our new graphs where necessary. DFS and BFS are both linear time.

To show correctness, we note that any path from $v$ to $w$ can be thought of as a path on $C$ where the number of bad edges crossed is exactly the length of the path in $C$. This is done by moving to a different vertex in $C$ only when our original path crosses a bad edge. In fact, any path in $C$ can be obtained this way, because two vertices in the same component of $G_g$ can be reached from one another without crossing a bad edge. Therefore, the smallest possible number of bad edges is the same as the length of the shortest path in $C$.

Alternative Solution: We can slightly modify BFS in order to make this work. In particular, when exploring a good edge $(v, w)$ if $w$ is unexplored, we put $w$ at the front of the queue (rather than the back) and set $d(w)$ to $d(v)$ rather than to $d(v) + 1$. Upon doing this it is not hard to see that we are still processing vertices in increasing order of distance (measured solely in terms of bad edges) from $v$ and thus that the algorithm works and in the same runtime as BFS.
Question 3 (Water System Design, 35 points). Helen is trying to update the water system in her city. She currently has a directed graph $G$ representing the existing system of (one-way) pipes. She would like to install a series of reservoirs at vertices of this graph so that every other vertex is reachable from some reservoir. Give an algorithm to compute the minimal number of reservoirs that she will need to build to accomplish this. For full credit, your algorithm should run in linear time.

The algorithm for this is actually quite simple. Just compute the metagraph of $G$ (linear time), and then return the number of source components.

Showing that this works is somewhat more difficult. On the one hand, note that Helen must put at least one reservoir in each source component. This is because the vertices in a source component cannot be reached from any vertices outside of that component. Therefore, the total number of reservoirs must be at least the number of source components. On the other hand, if Helen places exactly one reservoir in each source component that this will always be sufficient. This is because every component in the metagraph can be reached from a source component (in fact in any DAG any vertex is reachable from a source by tracing a path backwards until you get stuck). This path in terms of components can then be turned into a path between vertices without much difficulty.