Question 1 (Depth First Search Tree, 30 points). What is the Depth First Search Tree obtained when exploring the graph below starting at A? When running the algorithm, if given a choice about which of several vertices to visit next, always visit the alphabetically first one.

The answer is as shown below: DFS starts at A. From there E is alphabetically first of its neighbors so it goes there. Next M, then J, then K, then H, then G, then F, then B, then D, then O, then I, then C. At this point we are stuck and need to back up all the way to F to find a new edge to N. Again, we are stuck and need to back up all the way to M to add the final edge to L.
Question 2 (Intermediate Vertices on Shortest Paths, 35 points). Given an unweighted, undirected graph $G$ and two of its vertices, $v$ and $w$, give an algorithm that returns the set of all vertices $u$ of $G$ so that there exists a shortest path from $v$ to $w$ passing through $u$. For full credit, your algorithm must run in linear time.

We note that the shortest path from $v$ to $w$ through $u$ has length $d(v, u) + d(u, w)$. This is a shortest path from $v$ to $w$ if and only if $d(v, u) + d(u, w) = d(v, w)$. To solve this we first run BFS on $G$ starting at $v$ and record all values of $d(v, u)$ for all $u$ (including $u = w$). We then run BFS on $G$ starting at $w$ and record the values of $d(w, u)$ for all $u$. Finally, for each vertex $u \in V$ we add $u$ to our output list if $d(v, u) + d(u, w) = d(v, w)$ (noting that we can look up all of these numbers in our lists).

The runtime of this algorithm is $O(|V| + |E|)$ for each BFS, and an additional $O(|V|)$ for the final checks for each $u$. The total runtime is therefore $O(|V| + |E|)$. 
**Question 3** (Connected Points, 35 points). *Given a directed graph $G$ and a set $S$ of its vertices, give an algorithm to determine whether or not there is any vertex of $S$ reachable from any other vertex of $S$. For full credit, your algorithm should run in time $O(|V| + |E|)$. [Hint: Make sure that your algorithm doesn’t count paths from a vertex in $S$ back to itself. You may want to begin by computing the metagraph of $G$.]*

First compute the metagraph $G' = (V', E')$ of $G$. If any two elements of $S$ lie in the same strongly connected component, one can be reached from the other, so you can return ‘YES’. Otherwise, let $S' \subset V'$ be the set of connected components containing elements of $S$. We need to determine if any element of $S'$ is reachable from any other element of $S'$. However, since $G'$ is a DAG, it suffices to check whether there is any vertex $v$ so that $v$ is reachable from some vertex in $S'$ but has an edge to another vertex of $S'$. To do this we create a new graph $H$ by adding a vertex $v_0$ to $G'$ with outgoing edges to the elements of $S'$. Running $\text{explore}(v)$, we find the collection of vertices reachable from $v_0$ (and thus from $S'$). We check to see if any reachable vertex $w \neq v_0$ has an edge to an element of $S'$. If so, we return ‘YES’. Otherwise, we return ‘NO’.