Practice problems (don’t turn in):

DPV Problem 3.3 (Topological ordering example)
DPV Problem 3.4 (SCC algorithm example)
DPV Problem 3.5 (Reverse of graph)
DPV Problem 3.8 (Pouring water)

Instructions: In the subsequent algorithm design problems: use the algorithms from class, such as DFS, Explore, BFS, Dijkstra’s, connected components, SCC, etc., as a black-box subroutine for your algorithm. So say what you are giving as input, then what algorithm you are running, and what’s the output you’re taking from it. We don’t want you to go into the details of these algorithms and tinker with them, just use it as a black-box as I do below with Dijkstra’s algorithm. If you attempt to modify one of these algorithms you will not receive full credit, even if it is correct.

Make sure to explain your algorithm in words, no pseudocode. Explain briefly why it works, and explain the running time. Here’s an example:

1. Take undirected graph $G = (V, E)$ as input.
2. Find the vertex with largest degree, call it $v^*$. 
3. Compute the complement of the graph $G$, call it $\overline{G} = (V, \overline{E})$.
   This is the graph where $(v, w) \in \overline{E} \iff (v, w) \not\in E$.
4. Run Dijkstra’s algorithm on $\overline{G}$ with $s = v^*$. Let $\text{dist}[v]$ denote the output.
5. For each vertex $z$, let $f(v) = (\text{dist}[v])^2$.
6. Output the array $f()$.

This works because …

The running time is $O(n^2)$ because steps 2, 3, and 5 take $O(n)$ time, step 4 takes …
Problem 1  [DPV] Problem 3.15 (Computopia)

Note, linear time means $O(n + m)$ where $n = |V|$ and $m = |E|$.

Part (a):

Part (b):
**Problem 2  Global Destination**

In this problem: use the algorithms from class, such as DFS, Explore, BFS, Dijkstra’s, connected components, etc., as a black-box subroutine for your algorithm; see the instructions on the front page. Make sure to explain your algorithm in words. (Note, this is problem 3.22 in [DPV].)

Let $G = (V, E)$ be a directed graph given in its adjacency list representation. A vertex $v$ is called a **global destination** if every other vertex has a path to $v$.

**Part (a).** Give an algorithm that takes as input a directed graph $G = (V, E)$ and a specific vertex $s$, and determines if $s$ is a global destination. Your algorithm should have linear running time, i.e., $O(n + m) = O(|V| + |E|)$.

---

**Part (b). More Global Destination:**

Given an input graph $G = (V, E)$ determine if $G$ has a global destination or not. The running time of your algorithm should still be $O(|V| + |E|)$. In this problem you are no longer given $s$, and you need to determine whether or not $G$ contains a global destination.