1. (**(*)) [11 points] Prove the BFS shortest path algorithm I described in class is correct. That is, prove that if the shortest distance from $s$ to $v$ is $n$, then $L[v] = n$ (and if $s$ and $v$ are not connected, then $L[v] = \infty$.)

(Hints: Your proof will likely involve showing that closer nodes are explored first. Treat connected and disconnected cases separately.)

2. (*(*)) [6 points] What is the runtime of the BFS shortest path algorithm I described in class if the graph is given as an adjacency matrix instead of an adjacency list? (Let $n$ be the total number of vertices in the graph, $m$ the total number of edges, $n_s$ the number of vertices connected to $s$, the starting nodes, and $m_s$ the number of edges connected to $s$, the starting node.)

3. [6 points each] (*) Consider the following graphs and how Dijkstra’s algorithm behaves on these instances. For each graph, for each loop of Dijkstra’s algorithm, write the vertices in $X$ at the beginning of the loop, the value of $A[v]$ for the vertex $v$ most recently added to $X$, and calculate Dijkstra’s criterion for each edge from $X$ to $V - X$. Assume that $s$ is the starting node in both graphs. For example, for the graph on the left, after initialization we have $X = \{s\}$, $A[s] = 0$, and the edges from $X$ to $V - X$ are $(s, u)$ with criterion value 1, and $(s, v)$ with criterion value 4. Does the algorithm find the shortest path for each vertex in each graph?

4. Given a path from $s$ to $t$ in a graph $G = (E, V)$, we define the bottleneck of the path to be the largest weight of any edge on the path.

   (a) [9 points] (**(*)) Describe (using pseudocode) a modification of Dijkstra’s algorithm that solves this problem. (Hint - the algorithm stays the same, just the criterion changes.)

   (b) [11 points] (***) Prove your algorithm finds the path with smallest bottleneck from $s$ to every other vertex in $G$.

5. [6 points each] (*)(*) For the following statements, either explain why it is true (proof not required), or provide a counter example.
(a) Consider a graph $G$ that is directed, has negative edge weights, but no negative cycles (a negative cycle is a cycle where the sum of edge-weights in the cycle have negative value.) Then there will always be a vertex where the incorrect distance is calculated.

(b) Consider a graph $G$ that is directed, and that has a negative cycle that is reachable from $s$. Then there will always be a vertex where the incorrect distance is calculated.