1. Suppose you had a map of settlements on the planet X.

We want to build roads that allow us to travel between any pair of cities. Because resources are scarce, we want the total length of all roads built to be minimal.

a) Which pair of cities would you connect first? Why these cities?

b) What cities would you connect next?

c) Outline in English the algorithm you are following in (a) and (b).

d) What would be some characteristics of the resulting "graph" after all the cities are connected?

e) What would be the run-time of your algorithm?

f) Does your algorithm come up with the overall best (globally optimal) result?
2. Prim’s algorithm for determining the minimum-spanning tree of a graph is an example of a greedy algorithm. Unlike divide-and-conquer and dynamic programming algorithms, greedy algorithms DO NOT divide a problem into smaller subproblems. Instead a greedy algorithm builds a solution by making a sequence of choices that look best (“locally” optimal) at the moment without regard for past or future choices (no backtracking to fix bad choices).

a) What greedy criteria does Prim’s algorithm use to select the next vertex and edge to the partial minimum spanning tree?

b) What data structure could be used to efficiently determine that selection?

c) What would the run-time be for Prim's algorithm assuming an adjacency matrix graph implementation?

d) What would the run-time be for Prim's algorithm assuming an adjacency matrix graph implementation?
3. A strongly connected component, \( C \), of a graph \( G = (V, E) \) is the largest subset of vertices \( C \subseteq V \) such that for every pair of vertices \( v, w \in C \) we have a path from \( v \) to \( w \) and a path from \( w \) to \( v \).

a) Find the strongly connected components in the below graph:

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{graph.png}
\end{figure}

b) The transpose of a graph \( G = (V, E) \), denoted \( G^T \), has the same vertices as \( G \) and the same edges, except all the edges are reversed. Draw \( G^T \) above on the right.

c) The algorithm to compute the strongly connected components for a graph is:
1. Call dfs for the graph \( G \) to compute the finish times (order that we leave) for each vertex.
2. Compute \( G^T \).
3. Call dfs for the graph \( G^T \), but in the main loop of dfs, explore each vertex in decreasing order of finish time.
4. Each tree in the forest computed in step 3 is a strongly connected component.

Trace the algorithm