Note: There are three questions in this homework.

1. Given a directed acyclic graph $G = (V, E)$, write an $O(V^2)$ algorithm to determine whether for all pairs of nodes $u, v \in V$, there is a path from $u$ to $v$ or a path from $v$ to $u$. Describe your algorithm in (high-level) pseudo-code and explain it in plain English. (Hint: topological sort.)

2. Let $G(V, E)$ be an undirected graph. Modify the following algorithm so that it answers whether $G$ contains a cycle of odd length. Your modification must not increase the algorithm’s time complexity. Do NOT rewrite the whole algorithm; just make the necessary changes.

   procedure $Search(G = (V, E))$
   // Assume $V = \{1, 2, \ldots, n\}$ //
   // global variables: odd-cycle, visited[1..n] //
   $visited[1..n] \leftarrow 0$

   odd-cycle $\leftarrow$ false

   for $i \leftarrow 1$ to $n$

       if $visited[i] = 0$ then call $dfs(i)$

   return odd-cycle

   procedure $dfs(v)$

   $visited[v] \leftarrow 1;$

   for each node $w$ such that $(v, w) \in E$ do

       if $visited[w] = 0$ then call $dfs(w);$
3. Let $G = (V, E)$ be a directed graph. Modify the if statement in the $dfs$ procedure (and modify nothing else) to determine if $(v, w)$ is a tree, forward, back, or cross edge. Your modification must not increase the time complexity of the algorithm. Do NOT rewrite the whole algorithm; just make the necessary changes.

```
procedure Search($G = (V, E)$)
    // Assume $V = \{1, 2, \ldots, n\}$ //
    // time, vn[1..n], and fn[v] are global variables //
    time ← 0;
    vn[1..n] ← 0;
    fn[1..n] ← 0;
    for $i ← 1$ to $n$
        if $vn[i] = 0$ then call $dfs(i)$

procedure dfs($v$)
    $vn[v] ← time ← time + 1$;
    for each node $w$ such that $(v, w) \in E$ do
        if $vn[w] = 0$ then call $dfs(w)$;
    $fn[v] ← time ← time + 1$
```