NAMES: Michael Pass, Suchita Patel, Dakota Krogmeier

Worksheet 22: Work with the other students in your Zoom breakout room to complete this worksheet. You should only submit one paper for the group.

1. Develop an efficient algorithm to add an edge to an adjacency list representation of a directed graph (3 points):

   KEY IDEA:
   To have two checks to determine whether a vertex already exists in a graph and whether it was connecting to itself. The first check is to see if the integers of x and y are equal to each other and if they are, it removes the loop. The last check is to check if the edge is already in the graph. After the checks, connect the vertices of x and y by adding the edge.

   Algorithm AddEdge(*adj[][], (x,y)*):

   Input: A directed graph, G = (V, E), represented as an adjacency list, and an edge, (x, y)

   Output: The directed graph, G = (V, E ∪ { (x, y) }), represented as an adjacency list

   PROCESS:

```
for v in adj[x]:
        if v = y:
                remove loop
if adj[x] contains y:
        print("edge already exists")
else:
        adj[x].add(y)
return adj
```

2. Develop an efficient algorithm to remove an edge to an adjacency list representation of a directed graph (3 points):

KEY IDEA:
Because we do not know the index of the edge we want to remove, or whether this edge exists, we iterate through adjacency list adj[u][] and once v is detected, remove v from adjacency list.

Algorithm RemoveEdge(adj[][], (x,y)):

Input: A directed graph, G = (V, E), represented as an adjacency list, and an edge, (x, y)

Output: The directed graph, G = (V, E \ { (x, y) }), represented as an adjacency list

PROCESS:

```
for vertex in adj[x][]:
        if vertex = y:
                adj[x].remove(y)

return adj
```