1. We have focused primarily on *time complexity* in this course, but when choosing data structures, space complexity is often as important of a constraint. Given an adjacency matrix, what is the 'space complexity' in Big-O. That is, given *n* nodes, how much space (i.e. memory) would I need to represent all of the relationships given. Explain your response.

O(N^2) since it requires a 2D structure to present the mutual relationship in row(N) and column(N) which is N^2 possible relationships in total where N is the number of nodes

2. Will it ever make sense for ROWS != COLUMNS in an adjacency matrix? That is, if we want to be able to model relationships between every node in a graph, must rows always equal the number of columns in an adjacency matrix? Explain why or why not.

yes, due to its definition, adjacency matrix represents the connectivity of nodes with a graph therefore ROWS = COLUMNS must exist to show the uni-directional relationship.

3. Can you run topological sort on a graph that is undirected?

can’t since node will be mutually adjacent or not and it would be hard to determine which node comes first when u, v is adjacent since it’s undirected.

4. Can you run topological sort on a directed graph that has cycle?

I can use Tarjan algorithm to detect if cycle exist first, if so, then make the strongly connected component as one node go create a condensed graph then perform topological sort again. need extra pointer to store when to enter the cycle.

5. For question number 4, how would you know you have a cycle in a graph? What algorithm or strategy could you use to detect the cycles? **Hint** we have already learned about this traversal.

use DFS to check if there is back edge(repeat vertex on recursion stack of vertices with current vertex).