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.

   The space complexity for adjacency matrix is $O(n^2)$ because we will need to have n rows and n columns in the adjacency matrix. The matrix size is depending on the number of nodes. We will need to know the relationship(with direction) between each node.

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.

   The rows and columns must be equal because they represent the relationship between each node. If there is no relationship between nodes, the value in the array cell will be 0. However, we could not have less rows or columns in the matrix.

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

   No. The graph should be directed, otherwise we could not have linear order. If the graph is undirected, there could be cycles in graph and we could not use for topological sort.

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

   No. If there is a cycle in the graph, that means we will be directed back to the start node. Therefore, there is no end of topological sort and there is no linear order.

5. For question number 4, how would you know if 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.

   We could use DFS to determine if the graph has cycle or not. If the graph has cycle, we will return to the start node by using DFS. Because DFS will keep going to the neighbor node, and if there is a cycle, we will return to the start.