1. [10 points]

What is the running time of DFS if we represent its input graph by an adjacency ***matrix*** and modify the algorithm to handle this form of input? Hint: The algorithm we looked at and analyzed in class was for an adjacency list. How would this change for an adjacency matrix?

DFS with an adjacency list had a run time of O(n + m) where n is the number of nodes and m is the number of edges. This is because DFS will visit each of the n nodes in the adjacency list, and each node has a list of all the nodes adjacent to it (connected to it by one of the m edges) which will also be visited. There are n nodes and m edges indicating adjacent nodes, so it is O(n + m) to visit all nodes and all nodes' neighbors.

With an adjacency matrix, however, you have a 2D array of size n^2 since we have n nodes and each position in the 2D array will represent the relationship between two nodes. A value of one at a position x, y in the matrix indicates that there is an edge between node x and node y (they are adjacent), and a zero means they are not connected by an edge. In order to represent every possible combination of two nodes that may or may not be adjacent to each other, we need two identical lists of all our nodes, one for the x-axis and one for the y-axis, that way every node has an adjacency value (1 or 0, true or false) with every other node (ex. the x-axis might list nodes 0, 1, 2, 3, 4, 5 in which case the y-axis would also be 0, 1, 2, 3, 4, 5). Since DFS has to visit each position in this 2D array, just as it had to visit each position in the adjacency list, we can say our run time with an adjacency matrix is proportional to the size of the matrix or O(n^2).

2. [10 points]

What is the running time of BFS if we represent its input graph by an adjacency ***list***, and assume that the graph is undirected and ***dense*** (every node in the graph is connected to all other nodes)? Hint: If there are *n* nodes and all nodes are connected, how many edges *m* are there in terms of big O?

Normally, BFS has a run time of O(n + m) with an adjacency list (same as DFS) since every position in the list must be visited. This run time is the number of nodes plus the total number of edges. If we have n nodes and each node is connected to every other node, then we know that each of these n nodes has n-1 nodes adjacent to it. So the total number of edges m = (n nodes * n-1 neighbors) = about n^2. Therefore, instead of O(n + m), our run time would be O(n + n^2) with an undirected, dense graph, since our number of nodes is n and our total number of edges is n^2. Ignoring lower order terms, we can say that the complexity is simply O(n^2).

3. [10 points]

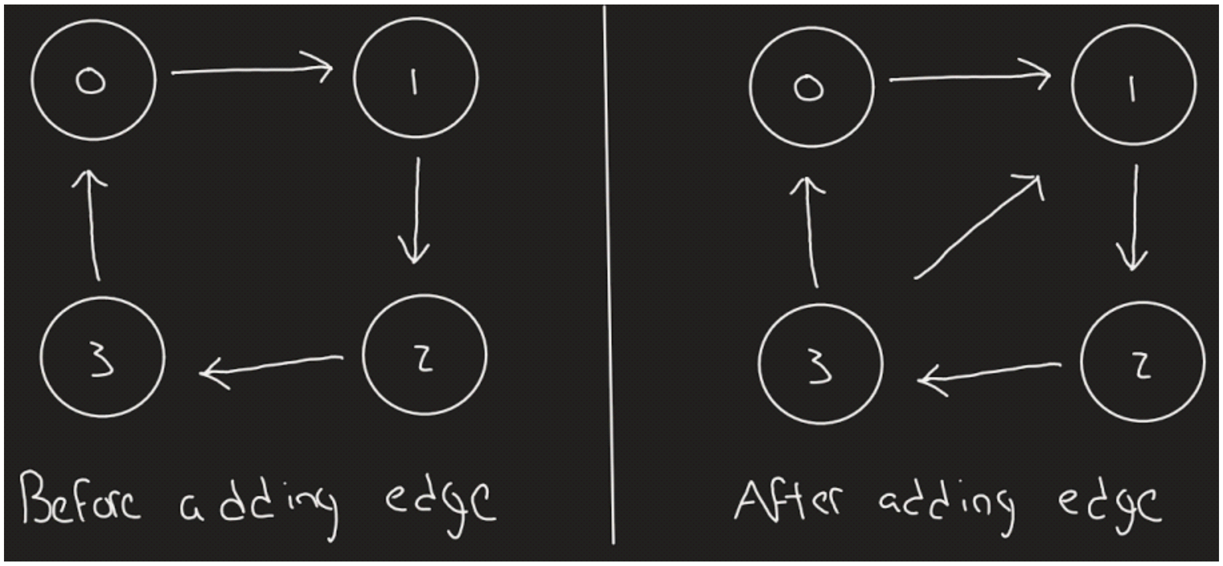
Is the topological sort in the Cormen textbook fig 22.7 unique? That is, are there other ways of sorting? Explain why or why not considering the DFS approach. Remember that the DFS approach is used to create the topological sort.

No, the topological sort is not unique. In this situation, DFS gives you a sequence representing the order in which you should put on your clothing items, but this is not the only possible sequence to do so. The sequence you get depends on how your algorithm decides which adjacent nodes to explore next (for example you could use a greedy strategy to pick the next node, or some type of heuristic). Instead of the order shown in the figure, you could put your watch on first or your undershorts before your socks. As long as you follow the edge directions (undershorts before pants, shirt before belt, etc.) there are multiple valid sequences that a DFS could give you.

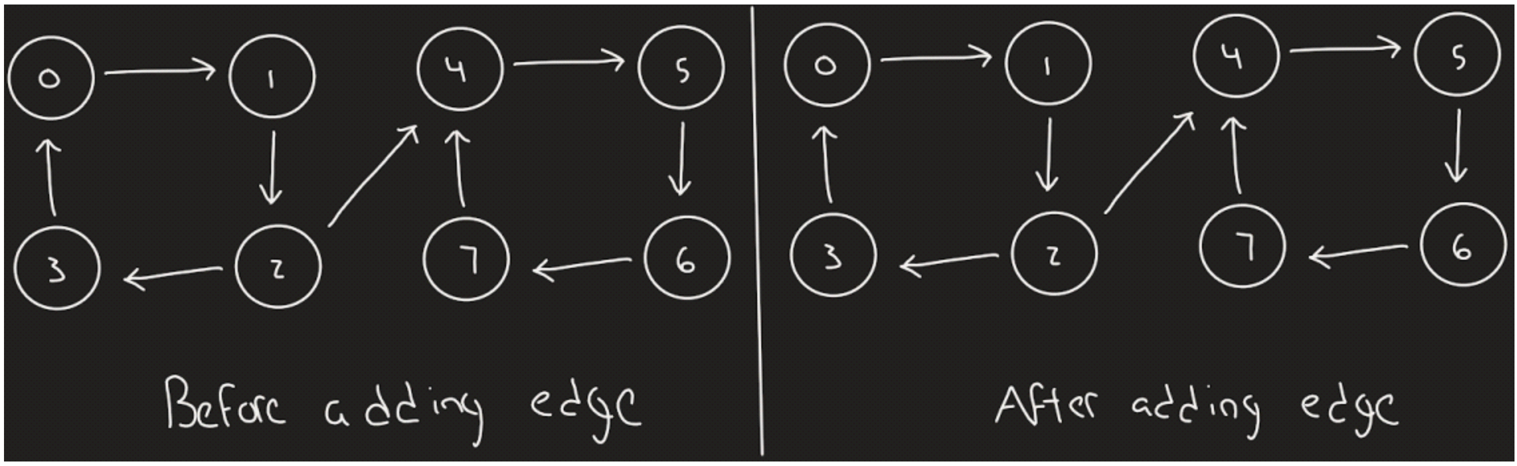
4. [10 points]

How can the number of strongly connected components of a graph change if a new edge is added? Demonstrate this by showing sketches of the different graph scenarios and adding a new edge.

If the edge you want to add is between two nodes that are in the same strongly connected component, then the total number of strongly connected components does not change, since by the definition of strongly connected every node is reachable from every other node. In this case, the edge you add does not have an effect because those two nodes were already reachable from each other, so the number of strongly connected components stays the same. This can be demonstrated by the example below, which has one strongly connected component before and after adding a new edge:



If instead you add an edge between two nodes that are in different strongly connected components, you will end up with one less strongly connected component than you started with because you have merged the two different strongly connected components into one. This is demonstrated below, where there are two strongly connected components before adding a new edge and just one after:


0
1
2
3
4
5
6
7
Before adding edge
0
1
2
3
4
5
6
7
After adding edge