TASK 1

(A) DFS

Here I did topological sort using DFS. For DFS, I declared a [visited] array with "True" value of len of the node number. Then I added a set called "cycle". This set kept track of cycle. ~~A the~~ At the begining of every recursive DFS step I added the node in cycle then later checked if the neighbors of the node is in the cycle. If the neighbors are there it determines a cycle and the loop breaks. And I kept the topological ordered elements in a stack which I popped later.

(B) BFS

Same algo as the before. I used indegree dictionary to keep track of the indegree of the graph. In this task luckily I implemented "raise ValueError" syntax to check the cycle in the graph. Here I used a queue to keep my bfs iteration.

#TASK 2

(I) Check indegree order of all the nodes of the graph.

(II) Then make a queue with those nodes who have '0' in degree order.

(III) So, I start a new stack "topo-order"

(IV) I start paping the ~~stack~~ queue elements from the first one and then look for the neighbor of them and I keep doing this till the ~~stack~~ queue gets empty.

(V) Finally I return the topo-order

#task 3

(I) I used kosaraju algo here

(II) In the first step I ran a dfs to make a stack according to the order of dfs visit

(III) Then I created a ~~traverse~~ transpose graph.

(IV) Then I ran second DFS algorithm to find the strongly connected components.