Task 01 (A)

The dfs takes the current course, g, visited nodes as v and a stack as s. Then the find_course_order takes the total number of courses N and list of prerequisites. It initializes an empty graph. The "v" is used to track visited courses. The loop will iterate from 1 to N. If the current node hasn't ~~e~~ been visited it calls the dfs function. After all this the stack contains the courses in valid order. After reversing it s[::-1] it gives correct order.

Task 01 (B)

The find_course_order takes the total number of courses N. and a list of prerequisites. It initializes an empty graph. The loop ~~is~~ iterates through each prerequisite (a,b) in the input. It adds course b as a prerequisite for course 'a' in the graph. and it increments the indegree of ~~course~~ course b since it now it has one more prerequisite. After this function initializes an empty queue. Another loop iterates through each course from 1 to N. If a course has 0 indegree it means it has no ~~prep~~ prerequisites. While queue is not empty the code pops a course from front of the queue. Then added to course order. Then the code iterates through each prerequisite 'p' of the completed course. It decrements the indegree of p since one of it's prerequisite has been satisfied. If indegree of p == 0 means its added to queue. If not empty list is returned.

Task 02

The function takes node, graph, visited and result. We initialize visited[node] = 1 then we iterate in the graph. if visited[i] = 1 it will return impossible otherwise we initialize visited[i] to 0. and now we will append the result the values will be under find_lexicographically_smallest_sequence takes a list and an ~~itege~~ integer. We run 2 loops one over prerequisite and other one over (1 to N+1). If visited 0 we return it to dfs.

Task-03

Create a stack and store Nodes. initialize visited array of size N to keep track. run a loop from 0 to N. If the node is not marked True in visited arr, call the recursive function. Now mark the current Node as True. Run a loop.

On all the nodes which has a directed edge to the current node. ~~It~~ If the node is False call the function. push the current Node in the stack.