Task 1.1

first I run a modified dfs that keeps a path tracker that helps it detect a cycle. This is so that ~~we~~ if there's a cycle, topological sort cannot happen. Then I run another dfs that pushes every element into a stack once it is fully visited. Now print the stack

Task 1.2

First I create an indegree list to list the number of incoming ~~no~~ edges. Add in ~~quee~~ queue those nodes who's indegrees are 0. Run bfs from the queue. Upon ~~t~~ hitting each ~~a~~ v of u, decrease indegree of v, append if indegree 0.

Task 2:

Here everything is the ~~t~~ same as task 1.2. Except ~~insted~~ instead of queue, we use priority queue. This uses the minheap system of heap data structure the always pop the smallest element ~~th~~ in the queue.

Task 3:

We use the kosaraju's algorithm. First we run dfs that stores elements in a ~~st~~ stack once they are fully explored. Now we transpose the graph. Now we pop an element from the stack each time and run the most basic dfs on the transposed graph, given ~~a node~~ the popped node is not already visited.