Answer to the question. 1(A)

Task 1(A)

The provided code implements topological sort using DFS to arrange nodes of a directed graph in a linear order such that for every directed edge from node A to B, node A precedes node B. It uses DFS to explore the graph, marking nodes as visited to avoid cycles. If a cycle is detected (a node is revisited within the same DFS path), it returns 'Impossible' since a topological sort cannot exist for cyclic graphs. Otherwise, after visiting

Answer to the question 1B

Task 1(B):

The code approached here is Kahn's algorithm, a BFS approach to topological sorting for directed graphs. It first calculates the in-degree for all nodes. It then enqueues all nodes with zero in-degree, as these have no dependencies and can be processed first. In each step, a node with zero in-degree is dequeued and added to topological order. Its removal simulates 'processing' that node and the in-degree of all its neighbours is decreased by one to reflect the removal of edges. If a neighbours in-degree drops to zero, its added to queue, as it's now ready to be processed. This process repeats until all nodes are processed. If the total number of processed nodes equals the number of nodes in the graph, a valid topological order is returned; otherwise, it indicates the presence of a cycle, making topological sorting impossible.

Task 2

The code snippet implements a lexicographical topological sort which arranges the nodes of a DAG in a linear order that is not only topologically valid but also in the smallest possible lexicographical order among all valid topological sorts. This is achieved using a priority queue (min-heap) to always select the smallest-numbered node that has no remaining dependencies (in-degree of zero). As nodes are processed and dependencies (edges) are effectively 'removed' (by decreasing the in-degree of dependent nodes), new nodes without dependencies are added to the priority queue, ensuring that at each step, the lexicographically smallest next node is processed. If all nodes are processed, a valid lexicographical topological order is returned; if not, it indicates a cycle in the graph, making a topological sort 'Impossible'.

Answer to the question. 3

Kosaraju's algorithm as implemented in the given code identifies strongly connected component (scc)s in a directed graph. first a DFS is done from each unvisited node, tracking the finish times of each node in a stack. This orders the nodes by decreasing order of their finish times, which helps in processing the nodes in correct order during second DFS. Then a transposed graph is created which helps in connecting the components when traversing. Using nodes ordered by stack from before, DFS is performed on the reversed graph. Started DFS from top of stack, we marked visited nodes and collected them as part of same SCC, until all nodes are processed. Each DFS from an univisited node in this step identifies a new SCC.