```
Function Kosaraju_SCC(graph):
    stack = empty stack          //Used to store nodes by their finishing order after the DFS is complete on each node.
 visited = empty set    //  A set to keep track of visited nodes, so each node is only processed once.
    SCC_list = empty list   //

    // Step 1: DFS to fill stack with nodes by finishing time
    For each node v in graph:
        If v is not in visited:
            DFS_Fill_Order(v, graph, visited, stack)

    // Step 2: Transpose the graph
    transposed_graph = Transpose(graph)

    // Step 3: Find SCCs in transposed graph
    visited = empty set
    While stack is not empty:
        v = stack.pop()
```

If v is not in visited:

scc = empty list // Create an empty list `SCC_list` to store all SCCs found.

DFS_Collect_SCC(v, transposed_graph, visited, scc)

SCC_list.append(scc)

Return SCC_list

// Helper function to perform DFS and fill stack by finishing order

Function DFS_Fill_Order(v, graph, visited, stack):

visited.add(v) //Add v to the `visited` set.

For each neighbor in graph[v]: //For each neighbor of vvv in the adjacency list `graph[v]`

If neighbor is not in visited: //If the neighbor is unvisited, recursively call DFS_Fill_Order on it.

DFS_Fill_Order(neighbor, graph, visited, stack)

stack.push(v) //Once all paths from v are fully explored, push v onto `stack`

```
// Helper function to transpose the graph
Function Transpose(graph):
    transposed = empty graph
    For each node v in graph:
        For each neighbor in graph[v]:
            Add edge from neighbor to v in transposed
    Return transposed

// Helper function to collect nodes in an SCC
Function DFS_Collect_SCC(v, graph, visited, scc):
    visited.add(v)  //
    scc.append(v) //
    For each neighbor in graph[v]:
        If neighbor is not in visited:
            DFS_Collect_SCC(neighbor, graph, visited, scc)
```