Q4> Algorithm assignment -6.4
Technical problem involved here is topological sorting, which allows us to sort the nodes of the graph in an order, respecting the dependencies that exists between nodes in the graph All the nodes whose degree is zero comes before other processes in the sorting.
For example if u -> v exists and if only u is independent then u comes first, followed by v.

Topological sort may result in different sequence of nodes that satisfy the dependency requirement based on which independent node gets selected at a given point if there are more than one independent node.

The tax software system that needs to be built should make sure all the questions which are sub-processes should be used to question users first followed by the questions which are part of main process. Let P1, P2, P3…Pn represent the processes, with edges pointing from sub-process to its parent process. For example : total itemized deductions is a parent process here and mortgage is a sub process. So we should have an edge directing from total itemized deduction to mortgage.

After constructing graph in this order, the following steps can be used to solve:

Step 1 : Fetch all the processes which are independent i.e. nodes with indegree equal to zero and add them to a queue Q.

Step 2: Initialize an empty stack R that can hold topologically sorted processes

Step 3 : While Q has processes in it, remove the first process at the head of the queue, say p1 from Q and push the process p1 on to result stack R.

Step 4: initialize a counter to count the number of processes visited.

Step 5: For every sub-process p1 adjacent to P, remove the edge between p and p1 and check if P1 is indegree of p1 is zero.(i.e. if it is independent) then add p1 on to Q.

Step 6: If count is not equal to the number of vertices in graph then return empty array as cycle exists

Step 7: Return R

Processing the processes in the result stack by popping one after another will allow TaxNow software to present the questionnaire to the user in the order which respects dependencies.

Input: Let V be nodes in the graph, E be edges in the graph
Output : resStack containing processes in sorted order

```
FetchProcessesinSortedOrder(V, E)
{

    Initialize indegree for all vertices $v \in V$ as 0
```

```
// Calulating indegrees for all the vertices in the graph
for each((u,v) in E)
{
        indgree[u] ++;

}

Let Queue be a queue to keep track of all the independent
Processes


// Fetching all nodes with indegree 0 initially
for( v in v)
{
        If(indegree[v] == 0)
        {
                Queue.add(v)
        }

}
// count of visited processes
Int count =0;

Let resStack be a stack which holds all the processes in sorted order

// Processing all the process which are independent to get sorted order
while ( queue is not empty)
{
        u = queue.remove() // first process In the queue

        // add current process on to stack
          resStack.push( u)

        // For each adjacent edge of u
        for( (u, v)   in E)
        {
                // removing the edge (u,v)
                Indegree[v] - -

                If(indegree[v] == 0)
                {
                        Queue.add(v)

                }
        }
        Count ++;
}

// cycle exists if count is greater than the number of processes
```

```
            // cycle exists if count is not equal to the number of processes
            If(count != V.length)
                        Return empty array
           return resStack;

}
```