Topological Sort :-

Topological sort

Difficulty: **Medium** Accuracy: **56.52%** Submissions: **273K+** Points: **4** Average Time: **15m**

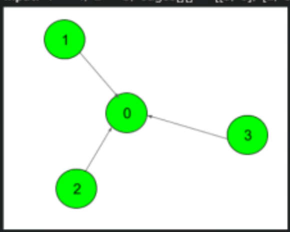
Given a **Directed Acyclic Graph (DAG)** of **V** (0 to V-1) vertices and **E** edges represented as a 2D list of **edges[][]**, where each entry **edges[i] = [u, v]** denotes an **directed edge u -> v**. Return **topological sort** for the given graph.

> Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge **u -> v**, vertex **u** comes before **v** in the ordering.

Note: As there are multiple Topological orders possible, you may return any of them. If your returned Topological sort is correct then the output will be **true** else **false**.

Examples:

Input: V = 4, E = 3, edges[][] = [[3, 0], [1, 0], [2, 0]]



Output: true
Explanation: The output true denotes that the order is valid. Few valid Topological orders for the given graph are:
[3, 2, 1, 0]
[1, 2, 3, 0]
[2, 3, 1, 0]

Kahn's Algorithm :-

```
class Solution{
  public:
   vector<int> topoSort (int V, vector<vector<int>> & edges) {
       vector <int> ans;
       vector <vector<int>> graph(V);
       vector <int> indegree(V);
       for(vector <int> & edge : edges){
            graph[edge[0]].push_back(edge[1]);
            graph[edge[1]].push_back(edge[0]);
            indegree [edge [1]] ++; }
     vector<bool> vis(V);
     queue<int> q;
     for (int i=0; i<V; i++) {
       if(!indegree [i]) q.push(i); }
      while (!q.empty()){
           int x = q.front();
            q.pop();
            if(!vis[x]) continue;
```

```
vis[x]=1;
ans.push_back(x);
for(int i=0; i<graph[x].size(); i++){
    indegree[graph[x][i]]--;
    if(!indegree[graph[x][i]]) q.push(graph[x][i]);
}}
return ans;}};
```

T.C. $O(V+E)$.
S.C. $O(V^2)$

Topological Sorting in Directed Acyclic Graphs (DAGs):-

DAGs are a special type of graph in which each edge is directed such that no cycle exists in the graph, let's understand why Topological sorting only exists for DAGs:-

- Why Topological sort is not possible for graphs with undirected edges?

  This is due to the fact that undirected edge between two vertices u & v means, there is an edge from u to v as well as from v to u. Because of this both the nodes u & v depend upon each other & none of them can appear before the other in the topological ordering without creating a contradiction.

- Why Topological Sort is not possible for graphs having cycles?

  Imagine a graph with 3 vertices & edges = {1 to 2, 2 to 3, 3 to 1} forming a cycle. Now if we try to topologically sort this graph starting from any vertex, it will always create a contradiction to our definition. All the vertices in a cycle are indirectly dependent on each other hence topological sorting fails.

Topological Sort:- (Dependency)

Topological sorting for Directed Acyclic Graph (DAG) is a linear

Topological [illegible]

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u-v, vertex u comes before v in the ordering.

Note:- Topological sorting for a graph is not always possible if the graph is not a DAG.

Topological Sorting May Not be Unique:-

Topological sorting is a dependency problem in which completion of one task depends upon the completion of several other tasks whose order can vary.

Algorithm for Topological Sorting using DFS:-

- Create a graph with n vertices & m-directed edges.
- Initialize a stack and a visited array of size n.
- For each unvisited vertex in the graph, do the following-
  - Call the DFS function with the vertex as parameter
  - In the DFS function, mark the vertex as visited & recursively call the DFS function for all unvisited neighbors of the vertex.
  - Once all the neighbors have been visited, push the vertex onto the stack.
- After all vertices have been visited, pop elements from the stack & append them to the output list until the stack is empty.
- The resulting list is the topologically sorted order of the graph.

C++ (g++ 5.4) Start Timer

```cpp
// } Driver Code Ends

class Solution {
  public:

    // Function to perform DFS and topological sorting
    void topologicalSortUtil(int v, vector<vector<int>> &adj, vector<bool> &visited, stack<int> &st){

        // Mark the current node as visited
        visited[v]=true;

        // Recur for all adjacent vertices
        for(int i:adj[v]){
            if(!visited[i])
                    topologicalSortUtil(i, adj, visited, st);
        }

        // Push current vertex to stack which stores the problem
        st.push(v);
    }


    vector<vector<int>> constructadj(int V, vector<vector<int>> &edges){
        vector<vector<int>> adj(V);
        for(auto it:edges){
            adj[it[0]].push_back(it[1]);
        }
        return adj;
    }


    // Function to perform Topological Sort
    vector<int> topoSort(int V, vector<vector<int>>& edges) {
        // code here
        // Stack to store the result
        stack<int> st;

        vector<bool> visited(V, false);
        vector<vector<int>> adj=constructAdj(V, edges);
        // Call the recursive helper function to store
        // Topological sort starting from all vertices
        // one by one
        for(int i=0;i<V;i++){
            if(!visited[i])
                topologicalSort(i, adj, visited, st);
        }
        vector<int> ans;

        // Append contents of stack
        while(!st.empty()){
            ans.push_back(st.top());
            st.pop();
        }
        return ans;
    }
};
```

T.C. O(V+E). This algorithm is simply DFS with extra stack. So, the time complexity is the same as DFS

A.S. O(V) due to the creation of the stack