Topological Sorting

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. Topological Sorting for a graph is not possible if the graph is not a DAG.

```
class Solution:

    #Function to return list containing vertices in Topological order.
    def topoSort(self, V, adj):
        # Code here
        visited = [False]*V
        res = []
        for i in range(V):
            if visited[i]==False:
                self.dfs(visited,V,adj,res,i)
        return res[::-1]


    def dfs(self,visited,V,adj,res,src):
        visited[src] = True
        for nbr in adj[src]:
            if visited[nbr]==False:
                self.dfs(visited,V,adj,res,nbr)
        res.append(src)
```

Complexity Analysis:

- **Time Complexity:** O(V+E).
  The above algorithm is simply DFS with an extra stack. So time complexity is the same as DFS which is.
- **Auxiliary space:** O(V).
  The extra space is needed for the stack.

Note: Here, we can also use vector instead of the stack. If the vector is used then print the elements in reverse order to get the topological sorting.

Applications:
Topological Sorting is mainly used for scheduling jobs from the given dependencies among jobs. In computer science, applications of this type arise in instruction scheduling, ordering of formula cell evaluation when recomputing formula values in spreadsheets, logic synthesis, determining the order of

compilation tasks to perform in make files, data serialization, and resolving symbol dependencies in linkers