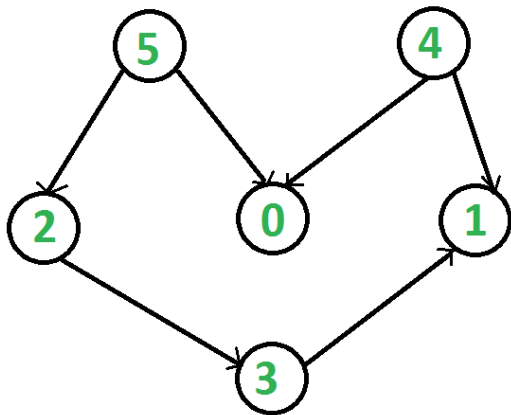


[Suggest a Topic](#)[Login](#)[Write an Article](#)

## Topological Sorting

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge  $uv$ , vertex  $u$  comes before  $v$  in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

For example, a topological sorting of the following graph is “5 4 2 3 1 0”. There can be more than one topological sorting for a graph. For example, another topological sorting of the following graph is “4 5 2 3 1 0”. The first vertex in topological sorting is always a vertex with in-degree as 0 (a vertex with no incoming edges).



### Topological Sorting vs Depth First Traversal (DFS):

In **DFS**, we print a vertex and then recursively call DFS for its adjacent vertices. In topological sorting, we need to print a vertex before its adjacent vertices. For example, in the given graph, the vertex '5' should be printed before vertex '0', but unlike **DFS**, the vertex '4' should also be printed before vertex '0'. So Topological sorting is different from DFS. For example, a DFS of the shown graph is “5 2 3 1 0 4”, but it is not a topological sorting

**Recommended: Please solve it on “[PRACTICE](#)” first, before moving on to the solution.**

### Algorithm to find Topological Sorting:

We recommend to first see implementation of DFS [here](#). We can modify **DFS** to find Topological Sorting of a graph. In **DFS**, we start from a vertex, we first print it and then recursively call DFS for its adjacent vertices. In topological sorting, we use a temporary stack. We don't print the vertex immediately, we first recursively call topological sorting for all its adjacent vertices, then push it to a stack. Finally, print contents of stack. Note that a vertex is pushed to stack only when all of its adjacent vertices (and their adjacent vertices and so on) are already in stack.

Following are the implementations of topological sorting. Please see the code for Depth **First Traversal for a disconnected Graph** and note the differences between the second code given there and the below code.

## C++

```
// A C++ program to print topological sorting of a DAG
#include<iostream>
#include <list>
#include <stack>
using namespace std;

// Class to represent a graph
class Graph
{
    int V;    // No. of vertices'

    // Pointer to an array containing adjacency lists
    list<int> *adj;

    // A function used by topologicalSort
    void topologicalSortUtil(int v, bool visited[], stack<int> &Stack);
public:
    Graph(int V);    // Constructor

    // function to add an edge to graph
    void addEdge(int v, int w);

    // prints a Topological Sort of the complete graph
    void topologicalSort();
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

// A recursive function used by topologicalSort
void Graph::topologicalSortUtil(int v, bool visited[],
                                stack<int> &Stack)
{
    // Mark the current node as visited.
    visited[v] = true;

    // Recur for all the vertices adjacent to this vertex
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            topologicalSortUtil(*i, visited, Stack);

    // Push current vertex to stack which stores result
    Stack.push(v);
}

// The function to do Topological Sort. It uses recursive
// topologicalSortUtil()
void Graph::topologicalSort()
{
    stack<int> Stack;

    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;
```



```

// 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] == false)
        topologicalSortUtil(i, visited, Stack);

// Print contents of stack
while (Stack.empty() == false)
{
    cout << Stack.top() << " ";
    Stack.pop();
}

// Driver program to test above functions
int main()
{
    // Create a graph given in the above diagram
    Graph g(6);
    g.addEdge(5, 2);
    g.addEdge(5, 0);
    g.addEdge(4, 0);
    g.addEdge(4, 1);
    g.addEdge(2, 3);
    g.addEdge(3, 1);

    cout << "Following is a Topological Sort of the given graph \n";
    g.topologicalSort();

    return 0;
}

```

Run on IDE

## Java

```

// A Java program to print topological sorting of a DAG
import java.io.*;
import java.util.*;

// This class represents a directed graph using adjacency
// list representation
class Graph
{
    private int V; // No. of vertices
    private LinkedList<Integer> adj[]; // Adjacency List

    //Constructor
    Graph(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; ++i)
            adj[i] = new LinkedList();
    }

    // Function to add an edge into the graph
    void addEdge(int v,int w) { adj[v].add(w); }

    // A recursive function used by topologicalSort
    void topologicalSortUtil(int v, boolean visited[],
                            Stack stack)
    {
        // Mark the current node as visited.
        visited[v] = true;
        Integer i;

        // Recur for all the vertices adjacent to this
        // vertex
        Iterator<Integer> it = adj[v].iterator();
        while (it.hasNext())

```

```

    {
        i = it.next();
        if (!visited[i])
            topologicalSortUtil(i, visited, stack);
    }

    // Push current vertex to stack which stores result
    stack.push(new Integer(v));
}

// The function to do Topological Sort. It uses
// recursive topologicalSortUtil()
void topologicalSort()
{
    Stack stack = new Stack();

    // Mark all the vertices as not visited
    boolean visited[] = new boolean[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    // 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] == false)
            topologicalSortUtil(i, visited, stack);

    // Print contents of stack
    while (stack.empty() == false)
        System.out.print(stack.pop() + " ");
}

// Driver method
public static void main(String args[])
{
    // Create a graph given in the above diagram
    Graph g = new Graph(6);
    g.addEdge(5, 2);
    g.addEdge(5, 0);
    g.addEdge(4, 0);
    g.addEdge(4, 1);
    g.addEdge(2, 3);
    g.addEdge(3, 1);

    System.out.println("Following is a Topological " +
                       "sort of the given graph");
    g.topologicalSort();
}
// This code is contributed by Aakash Hasija

```

Run on IDE

## Python

#Python program to print topological sorting of a DAG

```
from collections import defaultdict
```

#Class to represent a graph

```
class Graph:
    def __init__(self, vertices):
        self.graph = defaultdict(list) #dictionary containing adjacency List
        self.V = vertices #No. of vertices
```

# function to add an edge to graph

```
def addEdge(self, u, v):
    self.graph[u].append(v)
```

# A recursive function used by topologicalSort

```
def topologicalSortUtil(self, v, visited, stack):
```



```

# Mark the current node as visited.
visited[v] = True

# Recur for all the vertices adjacent to this vertex
for i in self.graph[v]:
    if visited[i] == False:
        self.topologicalSortUtil(i,visited,stack)

# Push current vertex to stack which stores result
stack.insert(0,v)

# The function to do Topological Sort. It uses recursive
# topologicalSortUtil()
def topologicalSort(self):
    # Mark all the vertices as not visited
    visited = [False]*self.V
    stack =[]

    # Call the recursive helper function to store Topological
    # Sort starting from all vertices one by one
    for i in range(self.V):
        if visited[i] == False:
            self.topologicalSortUtil(i,visited,stack)

    # Print contents of the stack
    print stack

g= Graph(6)
g.addEdge(5, 2);
g.addEdge(5, 0);
g.addEdge(4, 0);
g.addEdge(4, 1);
g.addEdge(2, 3);
g.addEdge(3, 1);

print "Following is a Topological Sort of the given graph"
g.topologicalSort()
#This code is contributed by Neelam Yadav

```

[Run on IDE](#)

Output:

```

Following is a Topological Sort of the given graph
5 4 2 3 1 0

```

**Time Complexity:** The above algorithm is simply DFS with an extra stack. So time complexity is same as DFS which is  $O(V+E)$ .

### 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 makefiles, data serialization, and resolving symbol dependencies in linkers [2].



## Topological Sorting | GeeksforGeeks



### Related Articles:

[Kahn's algorithm for Topological Sorting](#) : Another  $O(V + E)$  algorithm.

[All Topological Sorts of a Directed Acyclic Graph](#)

### References:

<http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/topoSort.htm>

[http://en.wikipedia.org/wiki/Topological\\_sorting](http://en.wikipedia.org/wiki/Topological_sorting)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [ConstantineShulyupin](#)

**Practice Tags :** Moonfrog Labs Accolite Microsoft OYO Rooms Amazon Flipkart DFS Graph

**Article Tags :** Graph Accolite Amazon DFS Flipkart Microsoft Moonfrog Labs OYO Rooms Topological Sorting





Please write to us at [contribute@geeksforgeeks.org](mailto:contribute@geeksforgeeks.org) to report any issue with the above content.

[Login to Improve this Article](#)

## Recommended Posts:

Depth First Search or DFS for a Graph

Kahn's algorithm for Topological Sorting

All Topological Sorts of a Directed Acyclic Graph

Detect Cycle in a Directed Graph

Longest Path in a Directed Acyclic Graph

Minimum cost path from source node to destination node via an intermediate node

Print all the cycles in an undirected graph

Dominant Set of a Graph

Print the DFS traversal step-wise (Backtracking also)

Find alphabetical order such that words can be considered sorted

(Login to Rate)

**3.1**

Average Difficulty : **3.1/5.0**  
Based on **209** vote(s)

Add to TODO List

Mark as DONE

Feedback

Add Notes

Improve Article

Basic

Easy

Medium

Hard

Expert

Writing code in comment? Please use [ide.geeksforgeeks.org](https://ide.geeksforgeeks.org), generate link and share the link here.

Load Comments

Share this post!



A computer science portal for geeks

710-B, Advant Navis Business Park,  
Sector-142, Noida, Uttar Pradesh - 201305  
[feedback@geeksforgeeks.org](mailto:feedback@geeksforgeeks.org)

#### COMPANY

[About Us](#)  
[Careers](#)  
[Privacy Policy](#)  
[Contact Us](#)

#### PRACTICE

[Company-wise](#)  
[Topic-wise](#)  
[Contests](#)  
[Subjective Questions](#)

#### LEARN

[Algorithms](#)  
[Data Structures](#)  
[Languages](#)  
[CS Subjects](#)  
[Video Tutorials](#)

#### CONTRIBUTE

[Write an Article](#)  
[Write Interview Experience](#)  
[Internships](#)  
[Videos](#)

@geeksforgeeks, Some rights reserved

























