Topological Sort

You can only do a topological sort on a **Directed Acyclic Graph (DAG)**.
A topological ordering is one in which all the edges go in the **same direction**. Either all edges point right to left or all edges point left to right.

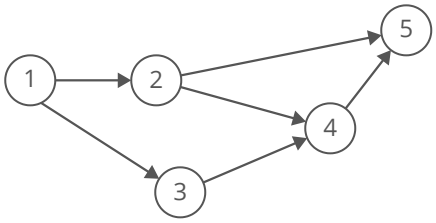
- The resulting order thus follows each node appearing before each of the nodes it points to.
- **NOTE:** Topological orderings are **NOT** unique.
  - Meaning you can have multiple valid topological orderings per graph. Source

Why Can't a Cyclical Graph Have a Topological Ordering?

There cannot be an order if there's a **cyclical dependency** since there's nowhere to start. You'll have trouble ordering the nodes in a single direction if there's a cycle.
Every node in a cycle depends on another, so any graph with a directed cycle is therefore forbidden.

Fun Fact: By definition every tree has a topological ordering, because they do not contain cycles. Source

Example from interviewcake.com



The **topological order** of the DAG above is `[1,2,3,4,5].`

- Notice how node 1 appears before both of the nodes it points to, the same rule apply for the others.
- Each node appears before the nodes `i`.

Recall earlier when I mentioned that you could have more than one topological orderings? Well for the graph above, `[1, 3, 2, 4, 5]` is another topological ordering.
Source

Pseudocode

```
// Require: G is a DAG and stored as an adjacency list
function topsort(G):
        visited = [false,..., false]
        ordering = [0,..., 0]
        for each v ∈ V do:
                if visited[v] is false do:
                        dfs(v, visited, ordering, G)
        return ordering

// Executes Depth First Search DFS,     inserts a node directly inside the next
valid position of the array and returns the next position, pos-1, as we are going
in reverse order.
function dfs(v, visited, ordering, G):
        visited[v] = true
        E = G.getEdges(v) //edges tell us the neighbors
        for each e ∈ E do:
                if visited[e] is false do:
                        dfs(e, visited, ordering, G)
        ordering.append(v)
```

Algorithm Description

1. Pick an **unvisited** node
2. Do a **Depth First Search (DFS)**, beginning with the selected node and explore only the unvisited nodes.
3. On the recursive callback of the DFS, add the current node to the topological ordering in **reverse order**

Time Complexity

The worst case time complexity of a topological sort is **O(V+E)**