So far we've seen DFS/BFS on graphs, and before that we saw sorting via total orderings. With a total order, we can place elements into a graph with an edge from $u$ to $v$ if $u \leq v$. This produces a nice "linked list" with freedom to move about equal elements.

Ex) \hspace{2cm} 1 \ 2 \ 3 \ 3 \ 4 \ 5

\[ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \]

Note we only drew the most immediate edges here.

What if we want to sort a partial ordering? For example $(a,b) \leq (c,d)$ if $a \leq c$ and $b \leq d$ is a partial ordering. We can compare such things as $(1,2) \leq (3,7)$, but we can't compare $(1,2)$ and $(2,1)$. We can still use the same setup as before though.

Ex) \hspace{2cm}

\[ (0,0) \rightarrow (1,0) \rightarrow (1,1) \rightarrow (2,0) \rightarrow (2,1) \rightarrow (2,2) \]

As before we only drew immediate edges, but $\leq$ is transitive, so the rest are implied.
We can have more practical partial orderings as well. For example, a prerequisite graph for courses produces a partial ordering. You must take Calc I before Calc II or Discrete Math, but Calc II and Discrete Math have no bearing on each other and are thus incomparable. In this case, we would want to be able to produce an ordering of classes that satisfies all prerequisites. The way we do this is via a topological sort. Given an acyclic digraph (DAG), we topologically sort it as follows:

$$TS(G)$$
Let $$L$$ be a list
While $$\exists u \in V$$ unvisited
$$DFS(G, u, L)$$
Return $$L$$

**Runtime** $$O(V+E)$$

$$DFS(G, u, L)$$
Mark $$u$$ visited
For each unvisited neighbor $$v$$ of $$u$$
$$DFS(G, v, L)$$
Put $$u$$ at the front of $$L$$
Return

**Ex:**

- **Pick H first in $$TS$$**
- **Pick I, D, H in $$TS$$**
- **Pick A, H in $$TS$$**

**HIKJADCEBF**
**HACBDEFGIKJ HIKJADCEBF**

Depending on where you start, you get very different but equally valid topological sorts. Notice that in a topological sort, if there is a path $$u \to v$$, $$u$$ appears before $$v$$. The converse is not true. In the example above, I appears before C, but I \(\nRightarrow\) C.
There is an alternative algorithm (Kahn’s algorithm) to topological sort. Since a DAG has no cycles, there must be a vertex with indegree 0. If we remove it, the remaining graph is still a DAG. Make note of the vertex and then repeat until every vertex is gone.

KahnTS(G)
Let L be a list
While V ≠ ∅
Remove a vertex u ∈ V with indegree 0
Put u at the end of L
Return L

Notice that we insert at the end of the list because we’re picking vertices with nothing before them. With DFS, we picked vertices with nothing after them.

To make the above algorithm more concrete, here’s how we do it.

KahnTSv2(G)
Let L be a list
Let Q be a queue
For each u ∈ V
For each neighbor v of u
v.indegree++

For each u ∈ V
If u.indegree = 0
Enqueue u into Q

While Q is not empty
Dequeue u from Q
Put u at the end of L
For each neighbor v of u
v.indegree--
If v.indegree = 0
Enqueue v into Q

Return L

Runtime: O(V + E) (AL)
We visit every vertex and remove every edge by simply ignoring its adjacency list, but determining indegree is hard. We have to precalculate it and keep track of deltas.

Runtime: O(V + E) (AL)
With an AM, you’d have to check every entry, so you end up with O(V²). This is worse because on sparse graphs E ≪ V².