1. Topological Ordering:

(a) Directed, acyclic graph:
   i. A *topological order* is a total ordering of the vertices such that for every edge, \((v, w)\), \(v < w\).

(b) Given an acyclic, directed, fixed graph with \(n\) vertices and \(m\) edges, a topological ordering can be found in \(O(n + m)\) time.

(c) One such algorithm is the repeated deletion of sources algorithm.
   i. Let \(L\) be the topsort list of nodes. Initiate a set \(S\) with all nodes that do not have incoming edges.
   ii. While \(S\) is not empty:
      A. Remove an element, \(v\), from \(S\) and place at the end of \(L\).
      B. Remove all edges \((v, w)\) from the graph. If any \(w\) has no more incoming edges, add \(w\) to \(S\).

(d) Correctness: Any vertex added to the list must have its ”parent” vertices added to the list before it since all incoming edges were removed.

2. Incremental Topological Ordering via DFS

(a) For dynamic graphs, the problem is maintaining a topological order of the directed graph as edges are added until a cycle is introduced.

(b) Let \(L\) be the list of vertices in topological order and \(ord(v)\) be the order statistic of \(v\) in \(L\).

(c) Ordered list maintenance data structure [1] and [2] allows:
   i. Check relative ordering of two vertices in \(O(1)\) time.
   ii. Insert a vertex after another vertex in \(O(1)\) time.
   iii. Delete a vertex from list in \(O(1)\) time.

(d) If we add a directed edge \((u, v)\) in the graph. We would run the following algorithm to update \(L\) (we assume that \(ord(v) < ord(u)\) otherwise we return the previous \(L\)):
   i. Start DFS from \(v\) and see if we can reach \(u\) from \(v\). If can reach \(u\), output cycle.
ii. While running DFS, mark all vertices, \( v_i \) that you encounter if \( \text{ord}(v_i) < \text{ord}(u) \) and \( \text{ord}(v_i) > \text{ord}(v) \).

iii. Terminate DFS branch if ever reach a vertex \( v_i \) where \( \text{ord}(v_i) > \text{ord}(u) \).

iv. Suppose \( C \) is a list of all vertices \( v_i \) where \( \text{ord}(v) < \text{ord}(v_i) < \text{ord}(u) \) in the order given in \( L \). Iterate in order all \( v_i \) in \( C \) and move a vertex to the end of \( C \) if that vertex is marked.

(e) Proof (shown in class): DFS will never reach vertices \( v_j \) where \( \text{ord}(v_j) < \text{ord}(v) \). (Why?) Can terminate DFS branch if reach vertex \( v_j \) where \( \text{ord}(v_j) > \text{ord}(u) \). (Why?)

(f) Runtime (worst case): \( O(m + n) \).

(g) Runtime (average case): \( O(n) \) amortized over \( m \) edge insertions. Each vertex may be marked at most \( O(n) \) times. Therefore, each edge may be traversed at most \( O(n) \) times by DFS. Over \( m \) edges insertions, the total number of traversed edges is \( O(nm) \) so \( O(n) \) amortized.

References
