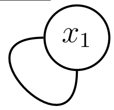
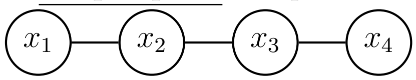
# Notes Week 8

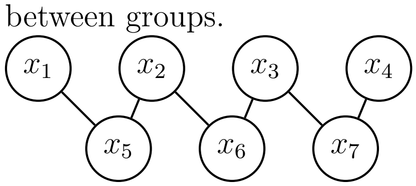
- A **loop** is an edge that connects a node to itself. Mostly relevant to directed graphs




- A **simple path** is a path where all vertices are distinct except possibly the first and last.




- A **cycle** is a path that starts and ends on the same vertex.
- **Acyclic** graphs contain no cycles.
- **DAG**: Directed Acyclic Graph
- **Strongly** connected graphs are ones where any vertex has a path to any other vertex; **Weakly** connected graphs have vertices without paths to other vertices.
- A **Bipartite Graph** is one we can separate vertices into two groups where edges only travel between groups.




- Formally, a **Bipartite Graph** is $BG = (A \cup B, E)$ such that every edge connects an element of A with an element B.
- A complete Bipartite graph is one where all nodes in group B are connected to all nodes in group B.
- A **Tree** is any connected, undirected, acyclic graph.

# Depth-First *vs* Breadth-First Searches

**Depth-First Search**

1. Push Start Vertex
2. Mark start vertex as visited
3. Loop until stack empty
   (a) Pop $U$
   (b) Mark $U$ as visited
   (c) For each of $U$ unvisited neighbors, push them

**Breadth-First Search**

1. Enqueue Start Vertex
2. Mark start vertex as visited
3. Loop until queue empty
   (a) Dequeue $U$
   (b) Mark $U$ as visited
   (c) For each of $U$ unvisited neighbors, Enqueue them

- Both BFS and DFS are $O(V + E)$
  - Vertices: $V = n$
  - Edges: $E = n(n-1)/2$ at most in a complete, undirected graph
  - Therefore: $O(n + n^2) = O(n^2)$ as an upper bound, but is dependet on
- Under what circumstances would BFS or DFS run faster?
- Well, neither? Closer on graph favors BFS.

# Topological Sort

- An ordering of vertices in a directed, acyclic graph such that, if there is a path, from vertex $A$ to vertex $B$, then vertex $B$ appears after vertex $A$ in the order.

$$x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$$

- Topological sorts are not guaranteed to be unique. There could be many correct orders.
- Examples: Family Tree, Course Prerequisites
- Assumptions:
  1. Indegree for each vertex is stored
  2. Edges stored in an adjacency list

**Algorithm 1** Topological Sort 1 $O(n^2)$

**while** Graph is not empty **do**
  Find any vertex with no incoming edges
  Display Vertex
  Remove it, and its edges, from the graph
**end while**

**Algorithm 2** Topological Sort 2

Maintain a queue of vertices with indegree 0
**while** Q **do**ueue is not empty
  Dequeue a vertex
  Display the Vertex
  Remove it and its edges from the graph
  Update remaining Vertices, enqueueing any whose indgree is 0
**end while**

- Each vertex enqueued and dequeued once $\rightarrow O(n^2)$
- Each edge gets removed once $\rightarrow n^2$ potential
- Therefore, Topological Sort 2 is $O(V + E)$
- TS2 depends on number of edges, whereas TS1 depends on nodes
- TS2 approaches TS1 only when the graph is complete, or close to it

# The Selection Problem

Find the $k^{th}$ largest number in a set of $n$ numbers.

- $i^{th}$ **order statistic**: the $i^{th}$ smallest term in a set of $n$ elements
- Recall week 1 notes; best algorithm was $O(nlog(n))$
- It's possible to make this $O(n)$
  - Pick a good pivot similar to QuickSort
  - Then like Binary Search, use the half that's relevant
  - The dividing runs in $(log(n))$ but the pivot search runs in $O(n)$