Kruskal's MST

$O(m \log m)$: Sort all edges from least to greatest weight.

$T = \varnothing$

Consider edges (in order):

- If next edge does not create a cycle in $T$, add it to $T$.

Data structure called Union-Find takes nearly constant time for each iteration of loop to consider the edges.
Prim's $O(m + n \log n)$
with Fibonacci heap

Kruskal $O(m \log m)$
Depth First Search

At high-level just like breadth first search except

1) Stack vs Queue

2) won't have source & instead we'll start from any undiscovered vertex & keep going until everything is discovered
top-level dfs

dfs()
color all vertices white & set parent's to null

time = 1

for each vertex u in graph
  if (u.color == white)
    dfs visit (u)

u will be root of a tree in dfs forest
dfs Visit (Vertex u)

u. color = gray
u. discovery Time = time +

For all vertices V reachable from U

do

this at most once per vertex

- if (V. color == white)
  - V. parent = U
  - dfs Visit (V)

- if (V. color == gray)
  - what does this mean? found a cycle

u. color = black
u. finishing Time = time +

Time Complexity: O(n + m)
Precedence Graph

Vertex corresponds to some task in a process

$U \rightarrow V$ means task $u$ must be done before task $v$

Question: Is there an order to do all tasks that obeys all precedence constraints?
Topological Sort
Reverse order of finishing times is a valid topological order UNLESS there is a cycle

1. Prove that if the graph has a cycle some back edge (gray to gray) is found

2. When no cycle the reverse order of finishing times is correct
If exploring $u \rightarrow v$ and $v$ is grey then there's a cycle.
Suppose there's a cycle

Let v be the first vertex discovered on this cycle.

v can't be removed from stack until u is discovered.
Prove when no cycle, reverse order of finishing times is a valid order.

Suppose not. There's some edge.

Working with u on top of stack where u finishes before v.
Strongly Connected Components

4 strongly connected components
Garbage Collection

Garbage is unreachable memory

user space | heap (do a "new" to get it)

ptr

[Diagram of a garbage collection process with objects and a reference pointer]