Kruskal's MST algorithm:

Greedy approach:

For input $G = (V, E)$
Sort the edges by weight
Let $X = \emptyset$
Go thru the edges in order of weight
For edge $e = (y, z)$
If $X \cup e$ is acyclic then add $e$ into $X$

How do we test if adding edge $e = (y, z)$ to $X$ will create a cycle?

Check if in the graph $(V, X)$, are $y$ & $z$ in the same component?
if in the same component, then adding $e$ creates a cycle
if in diff. components then adding it is OK.
To check if \( y \) & \( z \) are in the same component, we'll use a new data structure called Union-Find.

First, why is Kruskal's algorithm correct?

Suppose by induction that a set \( X \) is part of some MST \( T \).

Consider the next edge \( e = (y, z) \) that we add to \( X \).

We want to show that \( X \cup e \) is part of some MST.

Let \( G' \) be the graph on edges \( X \), so \( G' = (V, X) \).

In \( G' \), let \( c(y) \) be the component containing vertex \( y \), and \( c(z) \) be the component for \( z \).
We're adding \( e \) to \( X \) so it must be the case that \( c(y) = c(z) \).

Claim: \( e \) is the min weight edge from \( c(y) \) to the rest of the graph.

Why? Suppose there's an edge \( e' = (a,b) \) where \( a \in c(y) \), \( b \notin c(y) \)

& \( w(e') < w(e) \).

Then Kruskal's alg considers \( e' \) before \( e \), and it would have \( c(a) = c(b) \) so it would add \( e' \).

Then \( z \) would be in \( c(y) \).

Let \( S = c(y) \).

Since \( e \) is the min weight edge of \( E \) crossing \( S \leftrightarrow \overline{S} \)

& since no edge of \( X \) crosses \( S \leftrightarrow \overline{S} \),

Then by the cut property,

\[ X \cup e \subset T \text{ for a MST } T. \]
Union-find data structure:

- collection of sets - each set corresponds to a component in the graph \((V, E)\)
- each set has a unique name - there is a specific "root" vertex in each component, and the set's name is the root vertex.

Operations:
- MakeSet \((x)\): create a new set just containing \(x\)
- Find \((x)\): What is the name of the set containing \(x\)?
- Union \((x, y)\): Merge the sets containing \(x\) & \(y\).

\(O(1)\) time per MakeSet
\(O(\log n)\) per Find, and per Union
To check if \( y \) & \( z \) are in the same or different components, just check if \( \text{find}(y) = \text{find}(z) \)?

When adding edge \( e=(y,z) \) to \( X \),

Then to merge components \( c(y) \) & \( c(z) \),

\( \text{do Union}(y,z) \).

\[
\text{Kruskal}(G,w):
\]

input: connected, undirected \( G=(V,E) \) with edge weights \( w(e) > 0 \) for \( e \in E \)

output: MST defined by \( X \in E \) for all \( z \in V \), \( \text{Makeset}(z) \)

\( X = \emptyset \)

Sort \( E \) by \( \uparrow \) weight

For edge \( e=(y,z) \): (go thru edges by \( \uparrow \) weight)

if \( \text{Find}(y) \neq \text{Find}(z) \)

then \( \text{add } e \text{ to } X \)

\( \text{Union}(y,z) \)

\( \text{Return}(X) \)
Running time: \( n = |V|, m = |E| \).

Sorting \( E \Rightarrow O(m \log n) \) time.

\( n \) makesets \( \Rightarrow O(n) \) time.

\( O(m) \) Finds & \( \cup \) Unions \( \Rightarrow O(m \log n) \) time.

Total time: \( O(m \log n) \) time.

Union-find data structure:

Each set is a directed tree:
- edges point up to the root.
- name of the set is the root.

Example: \( \{B, E\}, \{A, C, D, F, G, H\}, \{I, J\} \)
Each node $x$ has 2 values:

1) $\pi(x) =$ Parent of $x$
   
   if $x$ is the root, then $\pi(x) = x$

2) $\text{rank}(x) =$ height of subtree below $x$.

$\text{Makeset}(x)$:

$\pi(x) = x$

$\text{rank}(x) = 0$

$\text{Find}(x)$:

While $x \neq \pi(x)$ do:

$x = \pi(x)$

Return($x$).
To merge sets containing \( x \) and \( y \),
Point root of one to root of other
key: to minimize depth,
Point root with smaller depth to larger.
So root with smaller rank points to larger rank.

**Union\((x, y)\):**

\[ \text{If } \text{rank}(r_x) > \text{rank}(r_y) \]
\[ \text{then } \pi(y) = r_x \]

\[ \text{If } \text{rank}(r_y) > \text{rank}(r_x) \]
\[ \text{then } \pi(x) = r_y \]

\[ \text{If } \text{rank}(r_x) = \text{rank}(r_y) \]
\[ \text{then } \left\{ \begin{array}{l} \pi(r_x) = r_y \\ \text{rank}(r_x)++ \end{array} \right. \]
Key claim: max depth is $\leq \log n$

Hence, find & union take $O(\log n)$ time.

Claim 2: Root of rank $k$ has $\geq 2^k$ nodes in its subtree (including itself).

From claim 2: Let $l$ be # of nodes of rank $k$.
Then $l \times 2^k \leq n$

So $l \leq \frac{n}{2^k}$

Let $k = \log_2 n + 1$
Then $\frac{l}{2} < 1 \implies$ there are $0$ nodes of rank $\geq \log_2 n$.

Proof of claim 2: by induction on $k$.

Base case: $k = 0$: it includes itself, so $2^0 = 1$.

Assume true for rank $< k$.

Consider node of rank $k$.
It was formed by union of 2 nodes of rank $k-1$.
By induction, each had $\geq 2^{k-1}$ in their subtree.
So now there are $\geq 2 \times 2^{k-1} = 2^k$ in the subtree.