Kruskal's Algorithm

This is a greedy algorithm that builds a tree by choosing the currently smallest edge weight, which avoids cycles.

Algorithm:

1. Sort edges according to weights. Label them as \( e_1, e_2, \ldots, e_m \). We have \( w(e_1) \leq w(e_2) \leq \ldots \leq w(e_m) \).

2. Initialize a spanning tree \( T = (V, S) \), by setting \( S = \emptyset \), and \( i = 0 \) (an index).

3. [Inductive step]: While \( |S| < n-1 \), let \( j \) be the smallest integer with \( j > i \), s.t. there are no cycles in \( S \cup \{ e_j \} \). Then set:
   \[
i = j, \quad S = S \cup \{ e_j \}\]
Kruskal's Algorithm

1. Start with the empty graph.
2. Sort all the edges by weight in non-decreasing order.
3. For each edge, add it to the growing spanning tree if it does not create a cycle.
4. Red edges comprise the minimum spanning tree.

Weight 5 closes a cycle.
**Kruskal's Simulations**

Always start with edge of smallest weight.

Only 6-1 edges!

A demonstration that every edge added to the tree closes a cycle.
Kruskal's Correctness

Why does the algorithm work? We proceed in two parts:
- the resulting graph is a spanning tree,
- this tree is of minimum weight.

Spanning Tree

Let $T$ be the resulting graph.

By construction, $T$ does not contain cycles. Also, we add exactly $(n-1)$ edges $\Rightarrow$ we showed that such a graph must be a tree on $n$ vertices.

Minimum Weight

We show by induction on the number of edges: Let $T_i = (V, E_i)$ be the graph
constructed after adding $i$ edges. Then there is some minimum spanning tree $T$ containing $T_i$.

In other words, the edges in $s_i$ always appear as the edges of the true spanning tree. Then at $i = n-1$ $T_i$ will be a minimum spanning tree.

**Base Case:** When $i = 0 \Rightarrow s_i = \emptyset$ and then it is clearly contained in a MST, $T$.

$i > 1$: Assume this property is correct for $i$, and we show it still holds for $i+1$.

Then $s_i$ is a set of edges contained in $T$. Now, suppose $e_{i+1}$ (the edge added to $s_i$)
is not contained in $T$, then $T \cup (e_{i+1})$
must close a cycle (why?). But now,
we can replace an edge $f$ of $T$ with $e_{i+1}$,
so we break the cycle, moreover, $w(e_{i+1}) \leq w(f)$
as we chose an edge of smallest weight (right after all edges in $S_i$).

$\Rightarrow w(T \cup (f) \cup (e_{i+1}))$ is smaller than $w(T)$. But this is a contradiction as $T$ is a MST.

$\Rightarrow S_i \cup (f) \cup (e_{i+1})$ must be in $T$. 
Kruskal: Complexity

Can be done in $O(\ell E \log |V| + |V| \log |V|)$

- Sorting edges according to weights
- Tracking connected components

$= O(\ell E \log |V|) = O(|V|^2 \log |V|)$

as $\ell E \leq \left(\frac{|V|}{2}\right)$.

This can be improved to:

$O(|V| \log (|V| + \ell E) + |E| + \ell E)$

in reverse Ackermann function,

$\ell |V| \leq 4$, for any practical sense.

So running time is

$O(|V| \log (|V|) + \ell E) = O(|V|^2)$
Prim's Algorithm

Algorithm:

1. Choose a root vertex \( v \).
2. Set \( W = \emptyset, S = \emptyset \) [Initialization].
3. While \( |W| < n \), let \( e \) be an edge of smallest weight, with one endpoint in \( W \) and the other not in \( W \).

Then update:

\[ W = W \cup \{ e \}, S = S \cup \{ v, y \} \]

[Inductive step].
Prim's simulations

Each time the currently chosen edge connects $T = (V, S)$ to the rest of the graph:

$\Rightarrow T$ is connected w/0 cycles.
Prim's Algorithm: correctness

First, the graph \( T \) computed by the algorithm is connected, as the original graph is connected, and each time we pick an edge connecting current connected component (containing current tree tree computed so far) with the rest of the graph, since \( G \) is connected, \( T = (W, S) \) is connected at any step.

\( T \) does not contain cycles, as \( e \) has only one endpoint in \( T \).
Prim: correctness

(continue). The minimality of $T$ follows from roughly similar arguments as in Kruskal's algorithm.

If there is another spanning tree $T'$ of minimum weight, it implies that we can replace one of its edges $f$ with an edge $e$ of smaller weight, $w(e) < w(f)$, and then obtain a contradiction to minimality.
Prim's complexity using "binary heaps", running time is:

\[ O\left( |V| \log |V| + |E| \log |V| \right) \]

Maintaining vertices sorting all edges outside the tree according to weights.

This can be improved to:

\[ O\left( |E| + |V| \log |V| \right) \]

using "Fibonacci heaps".