Kruskal’s algorithm.
let \( E \leftarrow e_1, \ldots, e_m \) be edges in increasing order of cost.
let \( T \leftarrow \emptyset \).

for \( i \leftarrow 1 \) to \( n \) do
  suppose \( e_i = (u, v) \).
  if there is no path from \( u \) to \( v \) in \((V, T)\),
    add \( e_i \) to \( T \).
end for

return \( T \).

Claim: if \( G = (V, E) \) is connected,
so is \((V, T)\), where \( T \) is set returned by Kruskal.

Proof: suppose \((V, T)\) is not connected.
so there are two vertices \( u \) and \( v \)
so that there is no path between them in \((V, T)\).
let \( S \subseteq V \) be all the vertices to which there is a
path from \( u \) in \((V,T)\).

Clearly, \( u \in S \) and \( v \in V-S \).

In \((V,T)\), there is no edge from \( S \) to \( V-S \). (otherwise, \( S \) would be larger.)

Since algo only adds edges to \( T \), there is no edge in \((V,T)\) from \( S \) to \( V-S \) during at any point of algo.

Since there is a path from \( u \) to \( v \) in \( G \), there is an edge \( f=(u',v') \) from \( u' \in S \) to \( v' \in V-S \) in \( G=(V,E) \).

Evidently, \( f \) was considered by algo but was not added to \( T \). This is a contradiction because there is no path from \( u' \) to \( v' \) in \((V,T)\) at the time \( f \) was considered.
Claim 2. Suppose $e_i = (u', v')$ is added by algorithm to $T$. Then $e_i \in \text{OPT}$.

Remark: OPT is some optimal solution.

Proof. Consider $(V, T)$ just before $e_i$ is considered. Let $S$ be all vertices to which there is a path from $u'$ in $(V, T)$. Clearly, $u' \in S$ and $v' \in V - S$.

There is no edge from $S$ to $V - S$ in $(V, T)$. So among edges from $S$ to $V - S$, $e_i$ is first one algo considers, so it is the cheapest edge.
Suppose $e_i \notin \text{OPT}$. There is a path $\pi$ from $u'$ to $v'$ in $\text{OPT}$. Since $u' \in S$ and $v' \in V-S$, there must be some edge $f$ in $\pi$ that goes from $S$ to $V-S$. We have $c_{e_i} < c_f$.

Observe that $\text{OPT} - f + e_i$ is also a spanning subgraph, i.e., $\exists (V, \text{OPT} - f + e_i)$ is connected.

But

$$\text{cost}(\text{OPT} - f + e_i) = \text{cost}(\text{OPT}) - c_f + c_{e_i} < \text{cost}(\text{OPT})$$

a contradiction to optimality of $\text{OPT}$. 
Prim's Algorithm.

\[ S \leftarrow \{ s \}, \text{ where } s \in V \text{ is an arbitrary vertex.} \]

\[ T \leftarrow \emptyset. \]

While \((S \neq V)\) do

Pick cheapest edge from \(S\) to \(V-S\). (Note that such edges exist.)

Suppose this is \(e = (u, v)\) where \(u \in S\), \(v \in V-S\).

Add \(v\) to \(S\) and \(e\) to \(T\).

Endwhile

Return \(T\)

Claim 1 \((V, T)\) is connected.

Proof: This follows from the fact that the while loop maintains the following invariant:
Any pair of vertices in $S$ is connected by a path in $(V,T)$. At the end, $S = V$. So at the end, any pair of vertices in $S \cup V$ is connected by a path in $(V,T)$. So $(V,T)$ is connected.

The invariant itself is readily seen to hold.

Claim 2: Suppose $e \in T$, then $e \notin OPT$.

Remark: Recall that $OPT$ is some optimal solution.
Proof: Suppose $e = (u,v)$ was added in some iteration of while loop because it was the cheapest edge
Connecting $S$ to $V-S$.

Suppose $e \in \text{OPT}$. There is a simple path $P$ from $u$ to $v$ in $\text{OPT}$, and this path has an edge $f$ from $S$ to $V-S$.

As in claim 2 of Kruskal,

$\text{OPT} - f + e$ is a spanning subgraph whose cost is less than $\text{OPT}$, a contradiction.
Graph Representation

The input graph $G = (V, E)$ is represented by an adjacency list representation.

$V = \{1, 2, \ldots, n\}$

There is an array $\text{Adj}[1..n]$:

$\text{Adj}[u]$ is a list containing all the nodes $v$ such that $(u,v) \in E$. For $(u,v) \in E$, the list element in $\text{Adj}[u]$ corresponding to $v$ also stores $c(u,v)$.

Implementing Prim's algo:

For each $v \in V - S$,

let $S(v)$ be the vertex $u$ that minimizes $c(u,v)$ over all $(u,v) \in E$ such that $u \in S$. 
For every \( v \in V - S \),
we store the pair \((v, s(v), c(v, s(v)))\)
in a priority queue with key \(c(v, s(v))\).

After adding \( s \) to \( S \) initialize,
we insert a priority queue element corresponding to every element incident to \( s \).

For vertices \( u \) not incident to \( s \),
we add dummy triple \((v, -\infty, \infty)\).

In each iteration of while loop,
extract-min will give us the edge \((u,v)\) we want.
Once we add $v$ to $S$, we go through every $w \in V - S$ such that $(v, w) \in E$ and update the priority queue entry corresponding to $w$ if necessary by a Change-Key operation.

**Running time:**

$O(n)$ for initialization. While loop takes $O(1) + \deg(v)$ a single iteration net counting the one extract-min operation and multiple Change-Key operations.

Note that there are $n-1$ iterations of while loop.

Total # of Change-Key operations overall $\leq m$. 
So running time is
\[ \leq O(n) + n \times \text{Time for extract-min} \\
+ m \times \text{Time for change-key} \\
= O(n + n \log n + m \log n) \\
= O(m \log n) \]