Exercise 1. Show that if for every cut of a graph there is a unique light edge crossing the cut, then the graph has a unique minimum spanning tree. Show that the converse is not true by giving a counterexample.
Remark: Do not assume that all weight edges are distinct.

Solution. Assume for a contradiction that the graph has two distinct minimum spanning trees T and T'. Let (u, v) be an edge in T which is not in T'. Removing edge (u, v) cuts the tree T into two components (trees). Let T_u and T_v be the vertices in the component containing u and v, respectively. Consider the cut (T_u, T_v) and let (x, y) be the unique light edge crossing the cut. If $(x, y) \neq (u, v)$ then $w(x, y) < w(u, v)$ and the spanning tree $T - \{(u, v)\} \cup \{(x, y)\}$ has a better cost than T, a contradiction.

Hence, assume that $(x, y) = (u, v)$, i.e., the unique light edge (u, v) crossing the cut (T_u, T_v) doesn't belong to T'. Consider the path p from u to v in T'. Path starts in T_u and ends in T_v, hence there must be an edge e on it crossing the cut (T_u, T_v) (there might be several of them, but take one). As (u, v) is the unique light edge crossing this cut, $w(u, v) < w(e)$. If we add (u, v) to T' we get a cycle composed of (u, v) and p. By removing any edge from the cycle we get again a spanning tree. Hence $T' \cup \{(u, v)\} - \{e\}$ is a spanning tree and by above it has a better cost than T', again a contradiction.

Counterexample for the converse. Consider a graph with 3 vertices a, b, c and weights

$$w(a, b) = w(a, c) = 1 \qquad \text{and} \qquad w(b, c) = 2\,.$$

The graph has a unique minimal spanning tree (containing edges (a, b) and (a, c)), however cut $(\{a\}, \{b, c\})$ doesn't have a unique light edge crossing the cut. □

Exercise 2. Show that for each minimum spanning tree T of G, there is a way to sort the edges of G in Kruskal's algorithm so that the algorithm returns T.
Remark: Do not assume that all weight edges are distinct.

Solution. Sort the edges of G in Kruskal's algorithm so that for every edge in T, it appears earlier in the sorted list than any other edge not in T with the same weight.

Consider a following loop invariant:

- prior to each iteration $A \subseteq E(T)$.

Initialization and termination are straightforward. What about maintance? Assume that A is contained in $E(T)$ prior to a certain iteration. Let (u, v) be the edge added to A during this iteration. If $(u, v) \in E(T)$, we are done as $A \cup \{(u, v)\} \subseteq E(T)$. Hence, assume for contrary that $(u, v) \notin T$. Let e be an edge in $E(T) - A$ with the smallest weight. Obviously, $w(e) \geq w(u, v)$, otherwise the algorithm would add e to A in some step before as it crosses two components of current configuration. If $w(u, v) = w(e)$, e would appear before (u, v) in the list, since $(u, v) \notin E(T)$ and $e \in E(T)$. But then again, algorithm would add e to A before processing edge (u, v). Hence, we can assume that $w(u, v) < w(e)$.

Take a path p connecting u and v in T. Since, u and v are contained in different components at the current iteration, there is an edge f in p not yet contained in A. The edge must be somewhere in the remaining portion of the ordered list, hence $w(f) \geq w(e) > w(u, v)$. Now, removing f from and adding (u, v) to T gives a spanning tree with a better cost than the cost of T, a contradiction with optimality of T. □