1. This question considers the case when a graph has more than one edge with the same weight value, in the context of Kruskal’s algorithm. In the case where edge weights are non-unique, there may be more than one (in fact, there could be many) MSTs. This question is asking us to “trick” Kruskal’s algorithm to return the MST we want. Note that Kruskal’s algorithm is deterministic - once we have finished sorting the edges in terms of weight. So our only chance to “trick” Kruskal is in doing the sorting.

Suppose we want the algorithm to return $T$, by obeying the rules of Kruskal’s algorithm. We use the following trick. Take *any* sorting algorithm and sort the edges of $G$ in terms of edge weight. Now examine this sorted list together with our special MST $T$. For every weight $w$ that labels *some* edge of $G$, the edges of $G$ with weight $w$ will all be grouped together in the sorted list. Identify which weight-$w$ edges actually belong to $T$ (not just $G$) and move these to the front of the $w$-group of edges in the sorted list (notice that after doing this the list is still sorted - but clearly not a stable sort!).

Do this for every weight $w$ which labels *some* edge of $G$.

After doing this procedure, get Kruskal to start processing the edges in this order, adding each edge *iff* its endpoints belong to different components of the current forest.

**Claim:** the tree that Kruskal returns is $T$.

**proof:** We prove this by induction. For each edge, we need to prove two things:

1. If $e$ is in $T$, it gets added by Kruskal.
2. If $e$ is not in $T$ it does not get added by Kruskal.

Suppose that after processing $k$ edges we have a forest $F$ of little sub-MSTs such that every edge of $F$ also lies in $T$ (and such that any edge of $T \setminus F$ appears after position $k$ in our special sorted list). This is the Induction Hypothesis.

Induction step: We must prove the same thing holds, after we process edge $e_{k+1}$.

**Case (1).** Suppose $e_{k+1} \in T$. We must show Kruskal will add $e_{k+1}$ to $F$.

Consider our current sub-forest $F$ after adding $e_1, \ldots, e_k$. The rule of Kruskal is that $e_{k+1}$ will be added to $F$ *iff* $u$ and $v$ belong to different subtrees of $F$. However, recall that $F \subseteq T$. If $u$ and $v$ lay in the *same* subtree of $F$, that would induce a cycle in $F \cup \{e_{k+1}\}$, and (since $F \cup \{e_{k+1}\} \subseteq T$) therefore also induce a cycle in $T$. Contradiction! We know $T$ contains no cycles. Hence there is no problem, and $e_{k+1}$ will be added.

**Case (2).** Suppose $e_{k+1} \notin T$, we now show $e_{k+1}$ will not be added to $F$.

Proof by contradiction. First observe that by our sorting of the edges (and by $e_{k+1} \notin T$), it must be the case that for every $(x, y) \in T \setminus F$, $W(x, y) > W(e_{k+1})$. Now suppose (setting up for our contradiction) that $e_{k+1} = (u, v)$ such that $u$ and $v$ are in different components of $F$ (and hence $e_{k+1}$ will be added to $F$). Consider the
path \( p_{uv}(T) \) between \( u \) and \( v \) in \( T \). Because \( u \) and \( v \) do not appear in the same connected component in \( F \), \( p_{uv}(T) \) must contain some edge \((x,y)\) of \( T \setminus F \). However, if that is the case \( W(x,y) > W(u,v) \). Then we can obtain an alternative spanning tree \( T' = (\mathcal{T} \setminus \{(x,y)\}) \cup \{(u,v)\} \) such that \( W(T') < W(T) \). Hence \( T \) was not a MST at all. Contradiction! So \( e_{k+1} \) would never have been added.

2. Here is the proof.
3. **Qn:** How fast Kruskal’s algorithm can be if the weights are in the range $1, 2, \ldots, |V|$ (same question, if the weights are in the range $1, \ldots, C$ for some constant $C$).

**Solution:** This question is really just asking about the “division of labour” between the initial sorting of the edges of $G$ (done on line 4 of the implementation, slide 19, lecture 10) and the work done by the loop on lines 5-8 of that implementation. We will write $n = |V|, m = |E|$.

We will be using some Data Structure for Disjoint Sets to implement Kruskal. Hence the total running time of the implementation works out at

$$T_{\text{sort}}(m) + n(T_{\text{MAKE-SET}}(n) + T_{\text{UNION}}(n)) + mT_{\text{FIND-SET}}(n).$$

Apart from the time to do sorting, this is the time to do $2n + m$ operations, $n$ of which are $\text{MAKE-SET}$. In Disjoint Sets terminology, we have $m’$ (for $m’ = 2n + m$) operations, $n$ of which are $\text{MAKE-SET}$. We know from our work on Disjoint Sets, that this can be done (using a Linked List implementation with the Weighted-union heuristic) in time $O(m’ + n \lg n)$, which is $O(m + n \lg n)$.

So any implementation of Kruskal takes time $T_{\text{sort}}(m) + O(m + n \lg n)$.

In the general case, we only know that $T_{\text{sort}}(m) = O(m \lg m)$. The term $m \lg m$ asymptotically dominates $O(m + n \lg n)$, so in the general case, the time to sort actually drives the running time of Kruskal.

Consider the case where the weights all come from $\{1, \ldots, n\}$ (hence these weights can be represented using $\lg n + 1$ bits). In this case, we look back to lecture 8 on Counting sort. Clearly $n = O(m)$ (we’ll assume $G$ was connected, so $m \geq n - 1$), so we can use Counting sort to sort all the edge weights in $\Theta(m)$ time. Hence in this case, the running time of Kruskal is

$$\Theta(m) + O(m + n \lg n) = O(m + n \lg n).$$

The second part of this question asks about the case when all weights lie in $\{1, \ldots, C\}$ for some constant integer $C$? Well, even in this case it still takes $\Omega(m)$ time to do the sorting. Why? Well we have to look at every edge-weight in order to do the sort, hence the best possible algorithm has $\Omega(m)$ running time. Hence the running time for Kruskal is exactly the same as for the $\{1, \ldots, |V|\}$ case.