Prim & Kruskal

The time complexity of Prims Algorithm:

We need to maintain the distance of each vertex from the small tree $T$ in each iteration. We enter the vertices into a priority-queue (a heap) with this distance from $T$ as a key. Initially we have the tree as a single vertex $v$. The key of $u \neq v$ is $dist(u, v)$. We insert into a heap all the vertices ($V$ times $O(\log V)$ is $O(V \cdot \log V)$ in total).

Now, each time a vertex $u$ enters the small tree $T$, we must do a (possible) decrease-key to all its neighbors (as their distance to $T$ may have decreased and be replaced by $dist(w, u)$, if $dist(w, u)$ is smaller than their current distance). This adds to at most $O(E)$ decrease-keys (one per each edge, or alternatively, $\sum_{v \in V} deg(v) = O(E)$). We saw in class that each Decrease-Key requires $O(\log V)$ time: so $O(E \cdot \log V)$ in total.

Finally, $V$ times of delete-minimum (to choose the vertex joining the tree) add up to $O(V \cdot \log V)$. As the graph is connected, $V = O(E)$, so the running time is:

$$O(E \cdot \log V).$$

Note that using Fibonacci-Heaps, one can do each decrease-key in $O(1)$ so the time is $O(E + V \cdot \log V)$ (check this!)

Proof of Correctness for the Kruskal algorithm:

Let $\{T_i\}$ be the collection of trees (forest) we are “growing”. We prove by induction on the edges added, that there is always an optimum tree $T^*$ containing all the edges of $\bigcup_i T_i$. This means that at the end, after $n - 1$ edges are added, we get an optimum tree (why?).

Initially, we have no edges, and each vertex is an isolated vertex. So any $T^*$ contains the empty set.

Now, suppose after $i$ edges are added, there is an appropriate optimum tree, and consider the next edge $e$ added. This is (according to the algorithm) the smallest edge $e$ not closing a cycle. In other words, the smallest edge $e$ passing from one tree, say $T_1$ to another tree, say $T_2$ (edges internal to some $T_i$ close a cycle).

If $e \in T^*$, we are done (why?). Suppose now that $e \notin T^*$. Add $e$ to $T^*$. A unique cycle $C$ is closed. The edge $e$ belongs to the cycle $C$. Now, go over the cycle starting with walking on $e$ from $T_1$ to $T_2$. Continuing over this cycle, we must return to $T_1$ (why?). The edge $e'$ that first returns to $T_1$, is an edge connecting two sub-trees $\{T_i\}$ one of which is $T_1$ (why?). So, by the property of the algorithm:

$$w(e) \leq w(e').$$

So, the tree $T^* \setminus \{e'\} \cup \{e\}$ is still a legal tree and is minimum because its weight is identical
to the weight of $T^*$. Further, this tree contains $e$, as required.