Kruskal's Algorithm for Minimal Spanning Tree

1. Form the spanning tree in incremental manner.
2. Initially start with a forest consisting only of vertices (and no edges).
3. At each stage, add an edge with minimum weight (to the forest constructed so far), which does not create a cycle.
4. Keep repeating step 3 until one gets a spanning tree.

For step 3:
(a) need to find edge of minimal weight among the remaining edges.

To do this efficiently, one can sort the edges in the beginning, and then go through them one by one.

(b) Need to be able to detect a cycle when an edge is added to the forest.

For (b):
(b1) Note that there may be several trees in the forest. Cycle is created only if the new added edge is within the same tree.
(b2) Keep the vertices in trees of the forest as sets. Each set corresponds to vertices in a tree of the forest.

(b2.1) Initially we have n sets, each consisting of one vertex of the tree.
(b2.2) Need to be able to detect whether two vertices belong to same set or different sets.
(b2.3) When an edge is introduced between vertices of two different sets, they need to be combined into one set.

Kruskal's Spanning Tree Algorithm

Input a weighted graph $G = (V, E)$.

Assume that the graph is given as adjacency list, so that one can get the edges in $O(m)$ time rather than $O(n^2)$ time, where m is the number of edges and n is the number of vertices.

Vertices are numbered 1 to n.

Output a spanning tree of the graph.

1. Sort the edges by weight. Suppose $e_1, e_2, \ldots,$ is the sorted order of the edges.
2. Initially, each vertex v is in a set of its own called v.
   $set(v)$ denotes the name of the set to which v belongs.
3. Initially, the spanning forest F consists just of the vertices.
4. For $i = 1$ to m do {
   4.1 Suppose $e_i = (v, w)$.
   4.2 If $set(v) \neq set(w)$, then {
      4.3 Let $F = F \cup \{(v, w)\}$.
      4.4 Combine sets $set(v)$ and $set(w)$.
      }
   }

Correctness:
Claim: At the beginning/end of every iteration of the For loop, suppose F is the forest that has been construted so far. Then there is a minimal spanning tree T which contains F.
Proof: At the beginning of the first iteration of the loop, this clearly holds as the forest has no edges.
Suppose by induction that the claim holds after some iteration of the for loop. We will claim that it holds after we add the next edge in the algorithm:
So suppose (v, w) is added to the forest.
If (v, w) is an edge in T then we are done.
Otherwise, consider $T \cup (v, w)$.
This graph has a cycle.
This cycle must include (v, w) and another edge which is not in F.
This another edge (say (v', w')) has weight at least as large as (v, w).
Then, consider the tree $T' = (T \cup (v, w)) - (v', w')$.
Then, T' is a minimal spanning tree of the original graph.

Complexity:

1. Sorting of edges takes time $O(m \log m)$.
2. Steps 2 and 3 are linear time.
3. Step 4.3 takes unit time.
4. Step 4.1, and 4.4: ??

If one only needs operations of
(a) "union", and
(b) "find the name of the set to which v belongs to" (which allows one to check if two elements belongs to the same set)
Then,
this can be done using $m \log m$ operations, where one starts with m sets containing 1 element each.
Thus, total complexity is $O(m \log m)$.