Problem 2.5 *Given that the edges can be ordered in m^2 steps, with, for example, insertion sort, what is the running time of algorithm 1?*

Given:

Algorithm 1 Kruskal

```
1: Sort the edges: c(e_1) ≤ c(e_2) ≤ ... ≤ c(e_m)
2: T ⟵ ∅
3: for i : 1..m do
4:    if T ∪ {e_i} has no cycle then
5:       T ⟵ T ∪ {e_i}
6:    end if
7: end for
```

Where line 4 is accomplished by adding the array, D (which maps the index of each vertex to its corresponding component) and updating it with:

Algorithm 2 Merging components

```
k ⟵ D[r]
l ⟵ D[s]
for j : 1..n do
   if D[j] = l then
      D[j] ⟵ k
   end if
end for
```

Solution:
We know that lines 1-2 of algorithm 1 require at most $m^2 + 1$ steps. We must also create the array D, which requires n more steps (where n is the number of vertices).

The for loop on line 3 will go through exactly m iterations. "$T \cup \{e_i\}$ has no cycle" (where $e_i = (r, s)$) is equivalent to "$D[r] \neq D[s]$", so the check on line 4 only requires one step. For the purpose of establishing an upper bound it is safe to assume that every check returns "true", so we must go through the entirety of algorithm 2 in every iteration of the for loop.

Algorithm 2 requires 2 assignments, followed by a loop which runs n times and has at most 2 steps; algorithm 2 is $O(2n + 2) = O(n)$.

So the composite algorithm, where algorithm 2 is used to accomplish line 4 of algorithm 1 and insertion sort is used for line 1, is clearly $O(m^2 + n + 1 + m(2n + 2))$. Identically, if $p = \max(n, m)$, the algorithm is $O(p^2)$.

In other words, if the number of edges is greater than the number of vertices the bottleneck is the sorting algorithm. Moreover, under the assumption that the graph in question is connected, the number of vertices is at least $n - 1$; any graph with $n - 1$ edges is either already a spanning tree or is not connected, so it is safe to assume $m \geq n$. Using merge sort, heap sort or quick sort for the sorting process could improve the complexity to $O(m \log(m))$.