Kruskal's Algorithm with Time Complexity Analysis

Algorithm: Kruskal's Minimum Spanning Tree (MST)

Input: A connected, weighted graph G with n vertices and m edges, represented by an edge list. Each edge (u, v) has a weight cost$[u, v]$.
Output: A minimum spanning tree represented as a set of edges t$[1 \ldots n-1, 1 \ldots 2]$, where t$[i, 1]$ and t$[i, 2]$ are the endpoints of the ith edge. The total cost of the MST is also returned. If no spanning tree exists, the algorithm outputs "No spanning tree."

Algorithm 1 Kruskal's Algorithm

```
1: Construct a heap of the edge costs using Heapify.
2: for i ← 1 to n do
3:     parent[i] ← −1                              ▷ Each vertex starts in its own set.
4: end for
5: i ← 0                                           ▷ Index for the MST edges.
6: mincost ← 0.0                                   ▷ Initialize total cost of the MST.
7: while i < n − 1 and (heap not empty) do
8:     Delete a minimum-cost edge (u, v) from the heap and reheapify.
9:     j ← Find(u)                                 ▷ Find the root of vertex u.
10:    k ← Find(v)                                 ▷ Find the root of vertex v.
11:    if j ≠ k then                               ▷ If u and v are in different sets.
12:        i ← i + 1
13:        t[i, 1] ← u, t[i, 2] ← v                ▷ Add edge (u, v) to the MST.
14:        mincost ← mincost + cost[u, v]
15:        Union(j, k)                             ▷ Merge the sets containing u and v.
16:    end if
17: end while
18: if i ≠ n − 1 then
19:    Write "No spanning tree."
20: else
21:    Return mincost
22: end if
```

Time Complexity Analysis

- **Step 1: Constructing the heap (line 1):**
  - Constructing a heap from m edges takes $O(m \log m)$ time.
- **Step 2: Initializing the parent array (lines 2–3):**
  - Initializing the parent array for n vertices takes $O(n)$ time.
- **Step 3: While loop (lines 7–16):**
  - The loop executes at most $n-1$ times since a minimum spanning tree contains $n-1$ edges.
  - Each iteration involves:
    * Deleting the minimum-cost edge and reheapifying (line 8): $O(\log m)$.
    * Finding the root of two vertices using `Find` (lines 9–10): Each `Find` operation takes $O(\log n)$ with path compression.

    - * Merging two sets using `Union` (line 14): The `Union` operation also takes $O(\log n)$.
  - Total cost per iteration: $O(\log m)+O(\log n)+O(\log n) = O(\log m+2\log n) = O(\log m)$ (since $m \geq n$ in a connected graph).
  - Total cost for all iterations: $O((n-1)\cdot\log m) = O(n\log m)$.

- **Step 4: Final check (lines 18–21):**
  - This step runs in $O(1)$ time.

Total Time Complexity:

$$\begin{aligned} O(m\log m) &\quad \text{(heap construction)} \\ +O(n) &\quad \text{(initialization)} \\ +O(n\log m) &\quad \text{(while loop)} \\ = O(m\log m + n\log m) &\quad \text{(since } m \geq n\text{)} \\ = O(m\log m). & \end{aligned}$$

Overall Time Complexity: $O(m\log m)$.