**Kruskal’s Algorithm**

#### A - Step 1: Sorting Edges by Weight

* **Input**: List of edges with weights.
* **Process**: Sort the edges in ascending order based on their weights using any efficient sorting algorithm, such as **Merge Sort** or C++'s std::sort() function.
* **Output**: A sorted list of edges.

**Step 2: Union-Find (Disjoint Set Union - DSU) Operations**

* **Purpose**: Ensure that adding an edge does not form a cycle in the MST.
* **Operations**:
  1. **Find Operation**: Determines the representative (root) of the set to which a vertex belongs.
     + Uses **path compression** to make subsequent queries faster.
  2. **Union Operation**: Combines two sets (two vertices) into one.
     + Uses **union by rank** to keep the DSU structure balanced.

**Step 3: Kruskal’s MST Algorithm**

1. **Initialize**:
   * Sort all edges by weight.
   * Initialize a DSU structure for the graph.
   * Create an empty list to store the MST edges and initialize the total MST weight to 0.
2. **Process Edges**:
   * For each edge in the sorted list:
     + Check if its two endpoints belong to different sets using the **Find Operation**.
     + If they are in different sets, add the edge to the MST and perform a **Union Operation** to merge the sets.
3. **Termination**:
   * Stop once n−1n-1n−1 edges are added to the MST, where nnn is the number of vertices.
4. **Output**:
   * The list of edges in the MST and the total MST weight.

### ****(b) Detailed Analysis of Kruskal’s Algorithm****

#### ****Correctness****:

Kruskal’s algorithm works because it processes edges in increasing order of weight, ensuring that every edge added to the MST maintains the minimal weight property without forming cycles.

#### ****Time Complexity****:

1. **Sorting Edges**: Sorting the EEE edges requires O(Elog⁡E)O(E \log E)O(ElogE), where EEE is the number of edges.
2. **Union-Find Operations**:
   * For each edge, we perform a union-find operation.
   * Using path compression and union by rank, the time complexity of each operation is O(α(n))O(\alpha(n))O(α(n)), where α(n)\alpha(n)α(n) is the inverse Ackermann function (almost constant time in practice).
   * Over EEE edges, this is O(E⋅α(n))O(E \cdot \alpha(n))O(E⋅α(n)).

**Overall Complexity**: O(Elog⁡E+E⋅α(n))O(E \log E + E \cdot \alpha(n))O(ElogE+E⋅α(n)), which simplifies to O(Elog⁡E)O(E \log E)O(ElogE).

#### ****Space Complexity****:

* Storage for edges: O(E)O(E)O(E).
* DSU structure for union-find: O(n)O(n)O(n), where nnn is the number of vertices.

**Total Space Complexity**: O(E+n)O(E + n)O(E+n).

c- code in c++ in github repo 😊