Task 1: Heap-Sort Algorithm

a. Algorithm Overview

Heap-Sort is a comparison-based sorting algorithm that utilizes a binary heap structure to efficiently sort elements.

Steps:

1. Build a Max-Heap from the input data.
2. Repeatedly swap the root node (maximum element) with the last node of the heap.
3. Reduce the heap size and re-heapify the root node.
4. Repeat until the entire array is sorted.

b. Algorithm Analysis

- **Time Complexity:**
  - **Build Heap:** O(n)
  - **Heapify (per node):** O(log n)
  - **Total Time Complexity:** O(n log n)
- **Space Complexity:** O(1) (In-place sorting, no extra memory required).
- **Best, Average, Worst Case Complexity:** O(n log n)
- **Stability:** Heap-Sort is *not stable* as it does not guarantee the relative order of duplicate elements.

Advantages:

- Efficient for large datasets.
- In-place sorting (no additional space required).

Disadvantages:

- Not stable.
- Slower than QuickSort for smaller datasets.

Task 2: Kruskal's Algorithm

a. Algorithm Overview

Kruskal's Algorithm is a graph algorithm used to find the Minimum Spanning Tree (MST) of a weighted, connected, and undirected graph.

Steps:

1. Sort all edges in ascending order by weight.
2. Initialize a Union-Find structure to detect cycles.
3. Iterate over the edges:
   - Add the edge to the MST if it does not form a cycle.
4. Stop when enough edges are added to form the MST.

b. Algorithm Analysis

- **Time Complexity:** O(E log E), where *E* is the number of edges.
- **Space Complexity:** O(V), where *V* is the number of vertices.
- **Best, Average, Worst Case Complexity:** O(E log E).
- **Stability:** Not applicable.

Advantages:

- Simple and efficient for sparse graphs.
- Guarantees an MST.

Disadvantages:

- Edge sorting can be computationally expensive for dense graphs.