1-Heap Sort

a. Write all required algorithms needed to sort a sequence of numbers using Heap sort Algorithms.

- First rearrange array elements to form a Max Heap
- Then repeat the following steps until the heap have one element:

1. Swap the root element of the heap with the last element in the heap.
2. Remove the last element of the heap
3. rearrange array elements to form a Max Heap

- When there is only one element in the heap we get a sorted array

b. Analyze in detail your written algorithms in Part (a).

We treat the array as a complete binary tree, where each index represents a node. The relationships between parent and child nodes are defined as follows:

- The left child of a node at index i is located at 2i+1.
- The right child of a node at index i is located at 2i+2.

To ensure the array satisfies the max-heap property (where each parent node is greater than or equal to its children), we start by comparing each node with its children. If the max-heap property is violated (i.e., a child is larger than its parent), we swap the parent with the largest child. This process is repeated recursively, starting from the lowest level with children and

working up to the root. By doing this, we transform the array into a valid max-heap.

Once the max-heap is constructed, we proceed with the heap sort operation:

1. Swap the root (the largest element in the heap) with the last element of the array.
2. Remove the last element from the heap (it is now sorted).
3. Restore the max-heap property by repeating the heapify process on the remaining unsorted portion of the array.

This process continues until there is only one element left in the heap. At this point, the array is fully sorted.

Time Complexity:

Heapify: O(log n)

Building the heap : O(n)

Total :O(n log n)

2- Kruskal's algorithm

a. Write all required algorithms needed to find MST using Kruskal's Algorithm.

1. Sort all the edges in ascending order based on their weight
2. Pick the smallest edge and check if it forms a cycle if there is no cycle include this edge else discard it
3. Repeat Step 2 until we reaches |V|-1 edges in the spanning tree

b. Analyze in detail your written algorithms in Part (a).

We begin by sorting all the edges of the graph in ascending order based on their weights. Next, we use a **Union-Find** data structure to keep track of connected components in the graph.

Starting with an empty Minimum Spanning Tree (MST), we iterate through the sorted edges. For each edge, we use the **Find** operation to determine the sets of the two vertices connected by the edge.

- If the vertices belong to different sets, the edge does not form a cycle, so we add it to the MST and use the **Union** operation to merge the two sets.
- If the vertices are in the same set, the edge would form a cycle, so we discard it.

This process continues until the MST contains exactly |V|−1 edges. At this point, the MST is complete.

Time Complexity:

Sorting edges : O(n log n)

Union-Find : O(n)

Total : O(n log n)