(i) The following programing assignment measures the ability to analyze and implement Heap-Sort algorithm. You are required to work *individually* in this work.

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

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

c. Implement your written algorithms in Part (a).

Part (a): Algorithms for Heap Sort

Heap Sort algorithm can be divided into two main steps:

1. **Building a Max-Heap**: Rearrange the array to satisfy the max-heap property, where the root is the largest element.
2. **Sorting the Heap**: Extract the maximum element from the heap, move it to the end, and reheapify the remaining heap.

Algorithms Needed:

1. **Heapify (Max-Heapify)**: Ensures the max-heap property for a subtree rooted at an index.
2. **Build-Heap**: Converts the entire array into a max-heap.
3. **Heap-Sort**: Sorts the array using the heap structure.

Part (b): Detailed Analysis of Algorithms

1. Heapify

- **Input**: Array A, index i , and heap size n.
- **Output**: The subtree rooted at i satisfies the max-heap property.
- **Time Complexity**: O(log n) for each call, as it adjusts a single path down the tree.

- **Algorithm**:
    1. Identify the largest element among the root, left child, and right child.
    2. Swap the root with the largest element if necessary.
    3. Recursively apply Heapify to the affected subtree.

2. Build-Heap

- **Input**: Array A of size n.
- **Output**: The array A becomes a max-heap.
- **Time Complexity**: O(n)
- **Algorithm**:
    1. Start from the last non-leaf node (n/2) and work upward.
    2. Apply Heapify to each node.

3. Heap-Sort

- **Input**: Array A of size n.
- **Output**: Sorted array A in ascending order.
- **Time Complexity**: O(n log n) combining the O(n) Build-Heap and O(n log n) sorting steps.
- **Algorithm**:
    1. Build a max-heap from the array.
    2. Extract the maximum element (root) by swapping it with the last element.
    3. Reduce the heap size and apply Heapify to the root.
    4. Repeat until the heap size is 1.

Part (c): Implementation

https://github.com/MENNA-1112004/assignment1-algo.git

(ii) The following programing assignment measures the ability to analyze and implement Kruskal's algorithm to find MST of a network. You are required to work with your colleagues in a teamwork (Maximum *Two–Three members*).

a. Write all required algorithms needed to find MST using Kruskal's Algorithm.
b. Analyze in detail your written algorithms in Part (a).
c. Implement your written algorithms in Part (a).

Part (a): Write all required algorithms needed to find MST using Kruskal's Algorithm

1. **Steps of Kruskal's Algorithm:**
   - Input: A connected, weighted graph G(V,E), where V is the set of vertices and E is the set of edges.
   - Output: A Minimum Spanning Tree (MST) containing |V| − 1 edges.
   - **Algorithm:**
     1. Sort all edges in E by their weights in non-decreasing order.
     2. Initialize an empty list MST[] to store the edges of the MST.
     3. Use a Disjoint Set Union (DSU) (also called Union-Find) to manage subsets of vertices and check for cycles:
        - Initialize each vertex as its own set (parent pointer).
        - Perform union and find operations to group vertices.
     4. Iterate through the sorted edges:
        - For each edge (u , v) with weight w, check if u and v are in the same subset using the find operation.
        - If not, include (u , v) in MST[] and union their subsets.

5. Stop when MST[] contains |V|−1 edges.

2. **Key Sub-algorithms:**
   - Sorting Edges: Use a sorting algorithm (e.g., quicksort, mergesort).
   - Union-Find Operations:
     - find(x): Find the root/representative of the set containing x.
     - union(x, y): Merge the sets containing x and y.

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

1. **Time Complexity Analysis:**

a. Sorting the edges: Sorting the edges takes O(E log E) time, where E is the number of edges.

b. Union-Find Operations: Each find and union operation has a time complexity of O(log V), where V is the number of vertices. This is because the find and union operations are highly optimized using path compression and union by rank, which ensures that the trees remain shallow.

c. Total for all edges: there are E edges, and each edge involves performing find and union operations, the total time for all these operations is O(E log V).

d. Overall Complexity: Combining the sorting and union-find operations, the total time complexity is O(E log E + E log V).

E log E will generally dominate E log V (because the number of edges E is usually greater than or equal to the number of vertices V), the final time complexity simplifies to O(E log E).

2. **Limitations:**

- **The algorithm assumes the graph is connected; otherwise, it finds a minimum spanning forest.**

Part (c): Implement your written algorithms in Part (a)

https://github.com/MENNA-1112004/assignment-2-algo.git