Algorithms Project

Task 1:

a. Write all required algorithms to sort a sequence of numbers using Heap-Sort:

1. algorithm 1: Heapify
2. algorithm 2: build max-heap
3. algorithm 3: heap-sort

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

1. Heapify Time Complexity

The heapify function runs in O(log n) because it traverses down a binary heap structure.

2. Build Max-Heap Time Complexity

The build_max_heap runs in O(n). This is because it calls heapify for all non-leaf nodes starting from the last non-leaf node toward the root.

3. Heap-Sort Time Complexity

The main loop in the Heap-Sort runs for n-1 iterations, and for each iteration, the heapify function (running in O(log n)) is called. Therefore:

- Time complexity: O(n log n)

4. Space Complexity

Heap-Sort runs in-place, meaning no extra memory is allocated. Therefore, the space complexity is O(1).

Algorithms Project

Task 2:

1. **Write all required algorithms to find MST using Kruskal’s Algorithm**
    a. Algorithm 1: Find
    b. Algorithm 2: Union
    c. Algorithm 3: Kruskal's Algorithm
2. **Analyze in Detail the Written Algorithms**

    a. Find Operation

    - Time Complexity: $O(\alpha(n))$
      The find function uses path compression, making it nearly constant time with the inverse Ackermann function, $\alpha(n)$, which grows extremely slowly.

    b. Union Operation

    - Time Complexity: $O(\alpha(n))$
      Union-by-rank ensures optimal merging of sets, and the rank is incremented only in the case of equal heights.

    c. Kruskal’s Algorithm

    - Time Complexity:
        - Sorting edges: $O(E \log E)$, where E is the number of edges.
        - Iterating through edges and performing find/union: $O(E\ \alpha(V))$, where V is the number of vertices.

    Combined, the complexity is $O(E \log E)$, since $\alpha(V)$ is negligible compared to $\log E$.

    - Space Complexity: $O(V)$
      Arrays parent and rank store information for V vertices.