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

1. heapify Function

This function ensures that a subtree rooted at node i in the array satisfies the Max-Heap property.

Parameters:

arr: The array to be heapified.

n: Size of the heap.

i: The current root node.

Steps:

1. Assume the root node i is the largest.

2. Check the left and right child nodes of i.

3. If any child is larger than the root, swap them.

4. Recursively heapify the affected subtree.

2. heap_sort Function

This function sorts an array using the Heap-Sort algorithm.

Steps:

1. Build a Max-Heap from the input array (bottom-up approach).

2. Repeatedly extract the largest element (root of the heap) and swap it with the last unsorted element.

3. Reduce the heap size and call heapify on the root node.

b. Analyze in detail your written algorithms

Time Complexity

1. heapify function:

In the worst case, heapify travels from the root to the lowest leaf, which takes O(log n) time.

2. Building the Max-Heap:

Heapify is called on all non-leaf nodes, and there are O(n) such nodes.

Total time complexity for building the heap: O(n).

3. Sorting:

Extracting the maximum element n times requires O(log n) per extraction.

Total time complexity for the sorting phase: O(n log n).

Overall Time Complexity:

Worst-case, Average-case, Best-case: O(n log n).

Space Complexity

The algorithm is in-place, so it requires O(1) additional space.

c: Implementation of Algorithms

```
# Function to heapify a subtree rooted at index i
def heapify(arr, n, i):
    largest = i  # Initialize largest as root
    left = 2 * i + 1  # Left child index
    right = 2 * i + 2  # Right child index

    # Check if left child exists and is greater than root
    if left < n and arr[left] > arr[largest]:
        largest = left

    # Check if right child exists and is greater than largest so far
    if right < n and arr[right] > arr[largest]:
        largest = right

    # Swap and continue heapifying if root is not largest
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

# Function to build a max-heap
def build_heap(arr):
    n = len(arr)
    # Start from the last non-leaf node and heapify each node
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

# Main function to perform heapsort
def heap_sort(arr):
    n = len(arr)
    build_heap(arr)

    # Extract elements one by one from the heap
    for i in range(n - 1, 0, -1):
        # Move current root to end
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

# Example usage
data = [12, 11, 13, 5, 6, 7]
print("Original array:", data)
heap_sort(data)
print("Sorted array:", data)
```

```
# Example Output:

Original array: [12, 11, 13, 5, 6, 7]

Sorted array: [5, 6, 7, 11, 12, 13]
```