Part A : Write all required algorithms needed to sort a sequence of numbers

using Heapsort Algorithms

HEAPSORT(arr):

1. n ← length of arr
2. BUILD_MAX_HEAP(arr, n)
3. for i ← n - 1 downto 1 do:
   a. Swap(arr[0], arr[i]) // Move the maximum element to the end
   b. n ← n - 1 // Reduce the size of the heap
   c. MAX_HEAPed(arr, 0, n)

BUILD_MAX_HEAP(arr, n):

1. for i ← floor(n / 2) - 1 downto 0 do:
   a. MAX_HEAPIFY(arr, i, n)

MAX_HEAPed(arr, i, n):

1. left ← 2 * i + 1 // Left child index
2. right ← 2 * i + 2 // Right child index
3. largest ← i
4. if left < n and arr[left] > arr[largest]:
   a. largest ← left
5. if right < n and arr[right] > arr[largest]:
   a. largest ← right
6. if largest ≠ i:
   a. Swap(arr[i], arr[largest])
   b. MAX_HEAPed(arr, largest, n)

Part (b): Analysis of Heapsort

Time Complexity Analysis:

Building the Max Heap:

The BUILD_MAX_HEAP function calls MAX_HEAPed for each non-leaf node.

The height of the heap is ($\log n$), and each MAX_HEAPed operation takes $O(\log n)$.

The total cost of building the heap is O(n).

Extracting Maximum Elements:

The loop in HEAPSORT runs n−1 times.

Each iteration performs a constant-time swap and a call to MAX_HEAPIFY, which takes O(log n).

The total cost of this phase is O(n log n).

Overall Time Complexity: O(n)+O(n log n)=O(n log n).