**HeapSort**

#### A –

#### 1. Algorithm: Build Max-Heap

The first step in Heap Sort is to build a max-heap from the input array.

**Algorithm**:

1. Start from the last non-leaf node in the array, which is at index ⌊n/2⌋−1\lfloor n/2 \rfloor - 1⌊n/2⌋−1.
2. For each node, call the heapify function to ensure the subtree rooted at that node satisfies the max-heap property.
3. Continue iterating upward to the root.

**2. Algorithm: Heapify**

The heapify function ensures the subtree rooted at a given index iii satisfies the max-heap property.

**Algorithm**:

1. Input: Array AAA, index iii, and size nnn.
2. Let largest = iii.
3. Calculate the left and right children:
   * left = 2i+12i + 12i+1
   * right = 2i+22i + 22i+2
4. Compare the values of the node at iii, its left child, and its right child:
   * If the left child is greater than the root, update largest = left.
   * If the right child is greater than the largest, update largest = right.
5. If largest is not equal to iii, swap A[i]A[i]A[i] with A[largest]A[largest]A[largest], and recursively call heapify on the affected subtree.

**3. Algorithm: Heap Sort**

After the max-heap is built, repeatedly extract the maximum element (the root of the heap) and place it at the end of the array.

**Algorithm**:

1. Input: Array AAA of size nnn.
2. Build a max-heap using the buildMaxHeap function.
3. For i=n−1i = n-1i=n−1 down to 1:
   * Swap A[0]A[0]A[0] (the root of the heap) with A[i]A[i]A[i].
   * Reduce the heap size by 1.
   * Call heapify on the root.

--------------------------------------------------------------------------

B - Analyze the Algorithms

**1. Time Complexity**

* **Building Max-Heap**:
  + For a heap with nnn elements, the number of comparisons at each level decreases exponentially.
  + Total time complexity: O(n)O(n)O(n).
* **Heapify**:
  + In the worst case, heapify is called O(log⁡n)O(\log n)O(logn) times per element because the height of the heap is log⁡n\log nlogn.
  + Total time complexity for heapify: O(nlog⁡n)O(n \log n)O(nlogn).
* **Heap Sort**:
  + Building the max-heap: O(n)O(n)O(n).
  + Sorting by repeatedly removing the maximum element: O(nlog⁡n)O(n \log n)O(nlogn).
  + Total time complexity: O(nlog⁡n)O(n \log n)O(nlogn).

**2. Space Complexity**

Heap Sort is **in-place**, requiring no additional space apart from a constant number of auxiliary variables. Therefore:

* Space complexity: O(1)

C- the code is in c++ in github repo