Heap Sort using Max Heap LeetCode

Given an array of integers nums, sort the array in ascending order and return it.

Example: [34, 43, 54, 21, 44, 35, 71, 55, 29, 93]

Output: [21, 29, 34, 35, 43, 44, 54, 55, 71, 93]

Approach 1: Function to perform Heap Sort with the help of Iterative Max Heapify Function

- **Function Purpose:** To perform Heap Sort using an iterative **maxHeapify** function.
- **Explanation:**
    - The **maxHeapify** function is used to build a max heap from the input array.
    - The code first constructs a max-heap from the input array by iteratively calling **maxHeapify**.
    - Once the max heap is constructed, it repeatedly extracts the maximum element (root of the max-heap) and moves it to the end of the array.
    - After each extraction, it re-heapifies the remaining elements to maintain the max-heap property.
- **Time Complexity: O(N log N) for the worst-case scenario.**
- **Space Complexity: O(1) for the in-place sorting.**

Approach 2: Function to perform Heap Sort using Priority Queue (Max Heap)

- **Function Purpose:** To perform Heap Sort using a priority queue (max heap).
- **Explanation:**
    - This approach uses a priority queue (implemented as a max heap) to sort the elements.
    - It first populates the priority queue with the elements from the input array, effectively building a max-heap.
    - Then, it extracts elements from the max-heap and places them back into the array in ascending order, which effectively sorts the array.
- **Time Complexity: O(N log N) for building the max-heap and extracting elements.**
- **Space Complexity: O(N) for the additional space required by the priority queue.**

Conclusion:

- Approach 1 (Iterative Max Heapify) is the better choice for implementing Heap Sort when memory efficiency is a concern. It has a space complexity of O(1), making it more memory-efficient.