Heap Sort using Min 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 Min-Heapify function

- **Function Purpose:** To perform Heap Sort using an iterative **minHeapify** function.
- **Explanation:**
    - The **minHeapify** function is used to build a min heap from the input array.
    - The code first constructs a min-heap from the input array by iteratively calling **minHeapify**.
    - Once the min heap is constructed, it repeatedly extracts the minimum element (root of the min-heap) and moves it to the end of the array.
    - After each extraction, it re-heapifies the remaining elements to maintain the min-heap property.
    - To obtain the sorted result in ascending order, the array is reversed.
- **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 a min-heap implemented with a priority queue

- **Function Purpose:** To perform Heap Sort using a priority queue (min heap).
- **Explanation:**
    - This approach uses a priority queue (implemented as a min heap) to sort the elements.
    - It first populates the priority queue with the elements from the input array, effectively building a min-heap.
    - Then, it extracts elements from the min-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 min-heap and extracting elements.**
- **Space Complexity: O(N) for the additional space required by the priority queue.**

Conclusion:

- Approach 1 (Iterative Min-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.