Merge K Sorted Arrays CodeStudio

You are given K arrays which are individually sorted in ascending order. You need to merge all the given arrays such that the output array should be sorted in ascending order.

Example: [1, 5, 9], [45, 90], [2, 6, 78, 100, 234]

Output: [1, 2, 5, 6, 9, 45, 78, 90, 100, 234]

Approach 1: Function to merge k sorted arrays using a brute force approach

- **Functionality:**
    - Merges k sorted arrays using a brute force approach.
- **Explanation:**
    - Iterates through each array and element to collect all values.
    - Sorts the collected values to obtain the final merged array.
- **Time Complexity:**
    - **mergeKSortedArraysBruteForce**:
        - **Time complexity for iterating through each array and element: O(N), where N is the total number of elements in all arrays.**
        - **Sorting the collected values: O(N log N).**
        - **Overall Time Complexity: O(N log N).**
- **Space Complexity:**
    - **O(N), where N is the total number of elements in all arrays.**

Approach 2: Function to merge k sorted arrays using a min-heap

- **Functionality:**
    - Merges k sorted arrays using a min-heap.
- **Explanation:**
    - Creates a min-heap using **Element** objects representing values, row index, and column index.
    - Initializes the min-heap with the first element from each array.
    - Continues merging elements until the min-heap is empty.
- **Time Complexity:**
    - **mergeKSortedArraysUsingHeap**:

        - **Time complexity for initializing the min-heap: O(K log K), where K is the number of arrays.**
        - **Each insertion and extraction from the heap: O(N log K), where N is the total number of elements in all arrays.**
        - **Overall Time Complexity: O(N log K).**
- **Space Complexity:**
    - **O(K), where K is the number of arrays.**

Approach 3: Merge k sorted arrays using a divide and conquer approach

- **Functionality:**
    - Merges two sorted arrays, recursively merges halves of k arrays, and merges k sorted arrays using divide and conquer.
- **Explanation:**
    - Utilizes **mergeTwoSortedArray** to merge two sorted arrays.
    - Recursively merges left and right halves using **mergeKSortedArrayHelper**.
    - Merges the two sorted halves using **mergeTwoSortedArray**.
- **Time Complexity:**
    - **mergeTwoSortedArray**: O(M + N), where M and N are the sizes of the two arrays being merged.
    - **mergeKSortedArrayHelper**:
        - **Time complexity for each level of recursion: O(N).**
        - **Number of levels in the recursion: O(log K).**
        - **Overall Time Complexity: O(N log K), where N is the total number of elements in all arrays, and K is the number of arrays.**
    - **mergeKSortedArraysDivideAndConquer: O(N log K).**
- **Space Complexity:**
    - **O(N), where N is the total number of elements in all arrays.**

Conclusion

- **Brute Force:** Simple, but less efficient due to sorting.
- **Min Heap:** Efficient for large datasets, particularly when K is significantly smaller than the total number of elements.

- **Divide and Conquer:** Balances efficiency and simplicity, suitable for various scenarios.