For the following two problems:

1. Implement the solutions and upload it to GitHub

2. Prove the time complexity of the algorithms

3. Comment on way's you could improve your implementation (you don't need to do it just discuss it)

Problem 1

Given K sorted arrays of size N each, the task is to merge them all maintaining their sorted order.

Examples:

Input: K = 3, N = 4
array1 = [1,3,5,7]
array2 = [2,4,6,8]
array3 = [0,9,10,11]
Output: [0,1,2,3,4,5,6,7,8,9,10,11]
Merged array in a sorted order where every element is greater than the previous element.

Input: K = 3, N = 3
array1 = [1,3,7]
array2 = [2,4,8]
array3 = [9,10,11]
Output: [1,2,3,4,7,8,9,10,11]
Merged array in a sorted order where every element is greater than the previous element.

Solution

Time Complexity Analysis

- Inserting into heap: Each insertion into the heap takes O(logK), where K is the number of arrays.
- Removing from heap: Each removal takes O (logK).
- Since there are N elements in each of the K arrays, we will insert and remove a total of K×N elements from the heap.

Therefore, the total time complexity is O (K×N×log K)

Improvement Discussion

- **Divide and Conquer**: Another possible approach is to use a divide-and-conquer strategy to merge arrays in pairs, like merge sort. This approach can also lead to an efficient solution with O (K×N×log K) complexity.
- **Parallelization**: For large K, we could potentially split the arrays into subsets and merge them in parallel.

- **Memory Optimization**: Depending on memory constraints, in-place merging might be explored to reduce space usage but only if we could overwrite the arrays.

Problem 2

Given a sorted array array of size N, the task is to remove the duplicate elements from the array.

Examples:

Input: array = [2, 2, 2, 2, 2]
Output: array= [2]
Explanation: All the elements are 2, So only keep one instance of 2.

Input: array = [1, 2, 2, 3, 4, 4, 4, 5, 5]
Output: array [] = {1, 2, 3, 4, 5}

Solution:

Time Complexity

- The algorithm makes a single pass through the array, performing constant work at each step.
- Hence, the time complexity is O(N), where N is the size of the input array.
- The space complexity is O (1), as we are modifying the array in place without extra space for data structures.

Improvements Discussion

- **Handling very large arrays**: For very large datasets, we could explore techniques to improve cache locality, minimizing memory access time.
- **Parallelization**: For large arrays, a parallel approach could be used by dividing the array into smaller chunks, processing them independently, and then merging the results.
- **Optimal Space Usage**: The current solution already achieves optimal space complexity by modifying the array in place.