Sum Of Maximum and Minimum Element of all Array elements in every K Window CodeStudio

This C++ program calculates the sum of maximum and minimum elements in subarrays of size 'k'

Approach 1: Function to calculate the sum of maximum and minimum elements in subarrays of size 'k' using brute force approach.

1. It calculates the number of subarrays of size 'k' (**size = n - k + 1**).
2. It iterates through each subarray of size 'k' and finds the maximum and minimum elements within each subarray.
3. The sum of maximum and minimum elements of each subarray is added to the **sum** variable.
4. Finally, it returns the **sum** as the result.
    - **Time Complexity: O(n * k) where 'n' is the size of the input array. In the worst case, the loop runs 'n - k + 1' times, and for each iteration, we find the maximum and minimum within a subarray of size 'k'.**
    - **Space Complexity: O(1) as no additional data structures are used apart from a few variables.**

Approach 2: Function to calculate the sum of maximum and minimum elements in subarrays of size 'k' using deque-based approach.

1. It initializes two deques (**maxDeque** and **minDeque**) to store the indices of maximum and minimum elements respectively.
2. It iterates through the input array and maintains these deques such that they always contain the indices within the current window of size 'k'. It removes elements from the front of the deques if they are out of the current window.
3. While iterating through the array, it also ensures that the deques are in decreasing order (for **maxDeque**) and increasing order (for **minDeque**) of elements.
4. When the current index is equal to or greater than 'k - 1', it calculates the sum of the maximum and minimum elements by using the front elements of the **maxDeque** and **minDeque**. These front elements represent the maximum and minimum elements within the current window.
5. The sum of maximum and minimum elements of each subarray is added to the **sum** variable.

6. Finally, it returns the **sum** as the result.

    - **Time Complexity: O(n) where 'n' is the size of the input array. The loop runs once through the entire array, and the operations performed on the deques are all constant time.**
    - **Space Complexity: O(k) as two deques are used, each with a maximum size of 'k'.**