3.Hard\10.Count_inversions.cpp

```cpp
/*
QUESTION:
Given an array of integers. Find the Inversion Count in the array.

Inversion Count: For an array, inversion count indicates how far (or close) the array is from being sorted. If the array is already sorted then the inversion count is 0. If an array is sorted in the reverse order then the inversion count is the maximum.
Formally, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.

Example 1:
Input: N = 5, arr[] = {2, 4, 1, 3, 5}
Output: 3
Explanation: The sequence 2, 4, 1, 3, 5 has three inversions (2, 1), (4, 1), (4, 3).

Example 2:
Input: N = 5, arr[] = {2, 3, 4, 5, 6}
Output: 0
Explanation: As the sequence is already sorted, there is no inversion count.

APPROACH:
To find the inversion count in the array, we can utilize the merge sort algorithm. The idea is to divide the array into two halves, recursively count the inversions in each half, and then merge the two halves while counting the inversions across them.

CODE:
*/

long long int inv_cnt = 0;

long long int merge(long long start, long long mid, long long end, long long arr[]) {
    long long leftsize = mid - start + 1;
    long long rightsize = end - mid;
    long long left[leftsize], right[rightsize];

    for (long long i = 0; i < leftsize; i++) {
        left[i] = arr[start + i];
    }
    for (long long i = 0; i < rightsize; i++) {
        right[i] = arr[mid + 1 + i];
    }

    long long i = 0, j = 0, k = start;
    while (i < leftsize && j < rightsize) {
        if (left[i] > right[j]) {
            inv_cnt += leftsize - i;
            arr[k++] = right[j++];
        } else {
            arr[k++] = left[i++];
        }
    }
    while (i < leftsize) {
        arr[k++] = left[i++];
```

```cpp
    }
    while (j < rightsize) {
        arr[k++] = right[j++];
    }
}

void mergesort(long long start, long long end, long long arr[]) {
    if (start >= end)
        return;
    long long mid = start + (end - start) / 2;
    mergesort(start, mid, arr);
    mergesort(mid + 1, end, arr);
    merge(start, mid, end, arr);
}

long long int inversionCount(long long arr[], long long N) {
    mergesort(0, N - 1, arr);
    return inv_cnt;
}

/*
TIME COMPLEXITY: O(N log N), where N is the size of the array.
SPACE COMPLEXITY: O(N).
*/

```