Data Structures/Algorithm Implementations

Count inversion pair in array range 1 to n

Problems:

Give an arrray[1..n], a pair call inversion if with i, j (1 <= i <j <= n) and (array[i] > array[j]) counting how many inversions pair?

Solution: Use divide and conquer (Hint: same with merge sort)

C++ source code:

```
/**
    * Brute Force
    * Complexity O(n^2)
    *
    **/
    /**
    *
    * Count Inversion by Merger Algorithms
    * Complexity O(nLog(n))
    *
    **/


#include <iostream>
#include <cstdio>
#include <cassert>
using namespace std;

const int __OO__ = 1e9 + 7;
const int SIZE = 1e6 + 5;

int n, a[SIZE];

int brute_force(int A[], int Len) {
    int inv = 0;
    for (int i = 1; i < Len; i ++)
        for (int j = i + 1; j <= Len; j ++)
            if (A[i] > A[j]) ++ inv;
    return inv;
}

int MERGE_INVERSIONS(int A[], int p, int q, int r) {
    int n1 = q - p + 1,
        n2 = r - q;
    int L[n1 + 1], R[n2 + 1];
    for (int i = 1; i <= n1; i ++)
        L[i] = A[p + i - 1];
    for (int j = 1; j <= n2; j ++)
        R[j] = A[q + j];
    L[n1 + 1] = __OO__, R[n2 + 1] = __OO__;
    int i = 1, j = 1;
    int mInv = 0; bool counted = false;
    for (int k = p; k <= r; k ++) {
        if (!counted && R[j] < L[i]) {
            mInv += n1 - i + 1;
            counted = true;
        }
        if (L[i] <= R[j]) {
            A[k] = L[i];
            i ++;
        } else {
            A[k] = R[j];
            j ++;
            counted = false;
        }
    }
    return mInv;
}
int COUNT_INVERSIONS(int A[], int p, int r) {
    int inv = 0;
    if (p < r) {
        int q = p + (r - p) / 2;
        inv += COUNT_INVERSIONS(A, p, q);
        inv += COUNT_INVERSIONS(A, q + 1, r);
        inv += MERGE_INVERSIONS(A, p, q, r);
    }
    return inv;
}
int main() {
    assert(freopen("INVERSIONS.INP", "r", stdin));
    cin >> n;
    for (int i = 1; i <= n; i ++)
        cin >> a[i];
    //cout << brute_force(a, n) << endl;
    cout << COUNT_INVERSIONS(a, 1, n) << endl;
    return 0;
}
```