Divide And Conquer
Count Inversions

Count the inversions in the given array.
Two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j

Sample Test Case:
A: [3, 5, 6, 9, 1, 2, 7, 8]
Inversions: 10
Explanation: (3,1), (3,2), (5,1), (5,2), (6,1), (6,2), (9,1), (9,2), (9,7), (9,8)

Brute Force:
```
int inv = 0;
for(int i = 0; i < n; i++){
    for(int j = i + 1; j < n; j++){
       if(a[i] > a[j])
           inv++;
    }
}
cout<<inv;
```
Time Complexity: O(N^2)

Using Divide and Conquer:
Idea: Divide the array into two parts, get the inversions for the left part and get the inversions for the right part recursively and merge the two arrays, and get their inversions.
```
#include "bits/stdc++.h"
using namespace std;
long long merge(int arr[],int l,int mid, int r){
    long long inv = 0;
    int n1 = mid-l+1;
    int n2 = r - mid;
    int a[n1];
    int b[n2];
    for(int i = 0;i<n1;i++){
        a[i] = arr[l+i];
    }
    for(int i = 0;i<n2;i++){
        b[i] = arr[mid+i+1];
    }
    int i = 0,j = 0, k = l;
    while(i<n1 and j<n2){
        if(a[i] <= b[j]){
            arr[k] = a[i];
            k++;i++;
        }
        else{
            arr[k] = b[j];
            inv += n1 - i;
            k++;j++;
        }
    }
    while(i<n1){
        arr[k] = a[i];
        k++;i++;
    }
    while(j<n2){
        arr[k] = b[j];
        k++;j++;
    }
    return inv;
}
long long mergeSort(int arr[],int l,int r){
    long long inv = 0;
    if(l < r){
        int mid = (l+r)/2;
        inv += mergeSort(arr,l,mid);
        inv += mergeSort(arr,mid+1,r);
        inv += merge(arr,l,mid,r);
    }
    return inv;
}
int32_t main() {
    int n;cin>>n;
    int arr[n];
    for(int i = 0;i<n;i++){
        cin>>arr[i];
    }
    cout<<mergeSort(arr,0,n-1);
    return 0;
}
```

Time Complexity: O(N log N)