Number Inversion

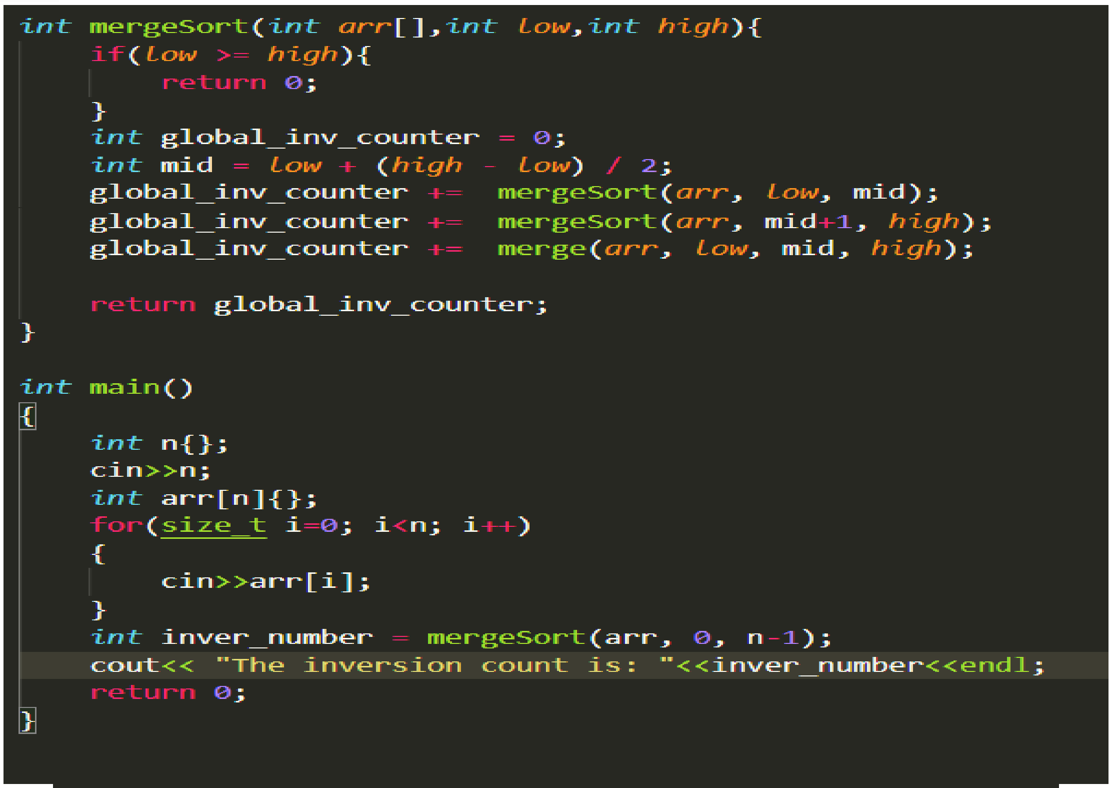
Problem Introduction:

An inversion of a sequence a0, a1, . . ., an−1 is a pair of indices 0 ≤ i < j < n such that ai > aj. The number of inversions of a sequence in some sense measures how close the sequence is to being sorted. For example, a sorted (in non-descending order) sequence contains no inversions at all, while in a sequence sorted in descending order any two elements constitute an inversion
(for a total of n (n − 1)/2 inversion.

Method used: (Divide and Conquer Approach using Merge Sort)

We use the classic merge sort algorithm in the same way we use it to sort an array, but while doing that for each element, we count all elements more than it to it's left as they represent all possible inversions for that element, then we add the count to the output, which all adds up during the recursion happens in the merge sort.

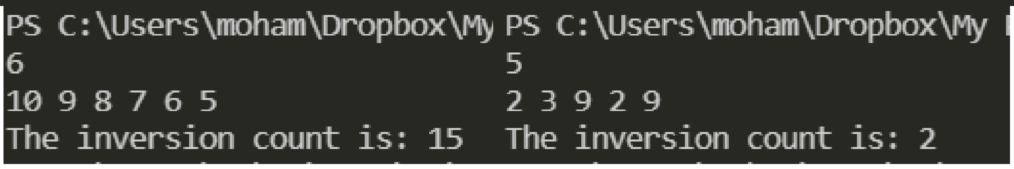
Of course, this method is more complex than the brute force approach but way faster and more optimized, as like sorting it takes time equal to nlog(n) instead of n^2.



```
int mergeSort(int arr[],int low,int high){
    if(low >= high){
        return 0;
    }
    int global_inv_counter = 0;
    int mid = low + (high - low) / 2;
    global_inv_counter +=  mergeSort(arr, low, mid);
    global_inv_counter +=  mergeSort(arr, mid+1, high);
    global_inv_counter +=  merge(arr, low, mid, high);

    return global_inv_counter;
}

int main()
{
    int n{};
    cin>>n;
    int arr[n]{};
    for(size_t i=0; i<n; i++)
    {
        cin>>arr[i];
    }
    int inver_number = mergeSort(arr, 0, n-1);
    cout<< "The inversion count is: "<<inver_number<<endl;
    return 0;
}
```



```
PS C:\Users\moham\Dropbox\My
6
10 9 8 7 6 5
The inversion count is: 15
```

```
PS C:\Users\moham\Dropbox\My
5
2 3 9 2 9
The inversion count is: 2
```