COUNT INVERSIONS

☆ In this problem, we are given an array of integers and we are supposed to return the number of pairs where left integer is greater in value but lesser in index

So (a[i], a[j]) such that i < j but a[i] > a[j]

We are to return the number of pairs not the set of all pairs

Brute force solution is to iterate for each element and find number of elements which are lesser than it. Time Complexity is $O(N^2)$ nearly.

Optimal Solution is to use the algorithm very similar to merge sort. What we do is we keep on splitting the array into half until we reach single element arrays. Then if the left element is greater than the right one, we just add one to our count and sort. When our left arrays element can form a pair with right array's element, so can all the elements to its right. Eg

[2 3 5] [1 4]

If here 2 can form a pair with 1 so can 3 and 5, so instead of adding 1 to the counter we add 3. We do this until array is fully sorted. Time Complexity for Merge Sort is O(N log N)

Pseudocode :

```
mergeSort(arr, low, high) {
    if (low >= high) return
    mid = (low + high) / 2
    mergeSort(arr, low, mid)
    mergeSort(arr, mid + 1, high)
    merge(arr, low, mid, high)
}

merge(arr, low, mid, high) {
    left = low
    right = mid + 1
    int ans[]
    while(left <= mid && right <= high) {
        if(arr[left] >= arr[right]) {
            ans.append(arr[right])
            right++, count += (mid - left + 1)
        } else {
            ans.append(arr[left])
            left++
        }
    }
    while(left <= mid) {
        ans.append(arr[left])
        left++
```

```
    }
    while (right < high) {
        ans.append(arr[right])
        right++
    }
    return ans
}
count = 0
countInversions(arr, N) {
    mergeSort(arr, 0, N-1)
    return count
}
```