Count Inversions

BRUTE FORCE

Count all pairs where i<j and arr[i]>arr[j] using 2 loops

```
#include <bits/stdc++.h>
long long getInversions(long long *arr, int n){
    // Write your code here.
    long long cnt=0;
    for(int i=0;i<n-1;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            if(arr[i]>arr[j])
                cnt++;
        }
    }
    return cnt;
}
```

- Time Complexity : O(N^2)
- Space Complexity : O(1)

Optimal Approach :

Using Merge Sort maintain a variable count and wherever arr[left] > arr[right] then increment counter by mid-left+1 for all recursive calls add counter in mergeSort and merge.

```
#include <bits/stdc++.h>
long long merge(long long *arr,int low,int mid,int high)
{
    long long cnt=0;
    int left=low,right=mid+1;
    vector<long long> temp;
    while(left<=mid && right<=high)
    {
        if(arr[left]<arr[right])
        {
            temp.push_back(arr[left]);
```

```
            left++;
        }
        else
        {
            temp.push_back(arr[right]);
            right++;
            cnt += (mid-left+1);
        }
    }
    while(left<=mid)
    {
        temp.push_back(arr[left]);
        left++;
    }
    while(right<=high)
    {
        temp.push_back(arr[right]);
        right++;
    }
    for(int i=low;i<=high;i++)
    {
        arr[i]=temp[i-low];
    }
    return cnt;
}
long long mergeSort(long long *arr,int low,int high)
{
    long long cnt=0;
    if(low>=high)
        return cnt;
    int mid=(low+high)/2;
    cnt+=mergeSort(arr,low,mid);
    cnt+=mergeSort(arr,mid+1,high);
    cnt+=merge(arr,low,mid,high);
    return cnt;
}
long long getInversions(long long *arr, int n){
    // Write your code here.
    return mergeSort(arr,0,n-1);
}
```

- Time Complexity : O(NlogN)
- Space Complexity : O(N)