Day 82/180 Merge Sort

1: Sort an array in non-increasing order using Merge Sort.

- This is the same as sorting in increasing order, you just have to change the comparison operator while comparing the numbers

```
#include <bits/stdc++.h>

using namespace std;

void merge(int arr[], int start, int mid, int end)
{
   vector<int>temp(end-start+1);
   int left = start, right = mid+1, index = 0;


   while(left<=mid&&right<=end)
   {
       if(arr[left]>=arr[right])
       {
           temp[index]=arr[left];
           index++, left++;
       }
       else
       {
           temp[index]=arr[right];
           index++, right++;
       }


   }
```

```
   // left array is not empty yet
   while(left<=mid)
   {
       temp[index]=arr[left];
       index++, left++;
   }
   // right array is not empty yet
   while(right<=end)
   {
        temp[index]=arr[right];
        index++, right++;
   }

   index=0;
  // put these value in array
  while(start<=end)
  {
   arr[start]=temp[index];
   start++, index++;
  }




}


void mergesort(int arr[], int start, int end)
{
   if(start==end)
   return;

   int mid = start+(end-start)/2;
   // left side
   mergesort(arr,start,mid);
```

```
    // right side
    mergesort(arr,mid+1,end);
    merge(arr,start,mid,end);
}




int main()
{
    int arr[] = {6,3,1,2,8,9,10,7,3,10};
    mergesort(arr,0,9);
    for(int i=0;i<10;i++)
    cout<<arr[i]<<" ";


}
```

2: Count Inversions

- So, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.
- As we know in merge sort while merging the array, we move elements that are to the right part to the left
- See in the **else part in the below code snap**, we're moving the arr[right] to the index position which means the value of arr[right] was less than all remaining elements.
- So, we have (mid-left+1) elements larger than arr[right] on the left.
- That is our count inversion: a[i] > a[j] and i < j.
- So, we just need to sum up and return the answer.

```
while(left<=mid&&right<=end)
```

```
   {
       if(arr[left]<=arr[right])
       {
           temp[index]=arr[left];
           index++, left++;

       }
       else
       {
           temp[index]=arr[right];
           ans+= (mid-left+1);
           index++, right++;
       }
}
```

Code :

```
void merge(long long arr[], int start, int mid, int end, long long int
&ans)
{
   vector<long long>temp(end-start+1);
   int left = start, right = mid+1, index = 0;


   while(left<=mid&&right<=end)
   {
       if(arr[left]<=arr[right])
       {
           temp[index]=arr[left];
           index++, left++;

       }
       else
       {
           temp[index]=arr[right];
           ans+= (mid-left+1);
           index++, right++;
```

```
        }


    }


    // left array is not empty yet
    while(left<=mid)
    {
        temp[index]=arr[left];
        index++, left++;
    }
    // right array is not empty yet
    while(right<=end)
    {
         temp[index]=arr[right];
         index++, right++;
    }

    index=0;
   // put these value in array
   while(start<=end)
   {
    arr[start]=temp[index];
    start++, index++;
   }




}


void mergesort(long long arr[], int start, int end, long long int &ans
)
{
    if(start==end)
    return;
```

```
    int mid = start+(end-start)/2;
    // left side
    mergesort(arr,start,mid,ans);
    // right side
    mergesort(arr,mid+1,end,ans);
    merge(arr,start,mid,end,ans);
}




    long long int inversionCount(long long arr[], long long N)
    {
        // Your Code Here
        long long int ans = 0;
        mergesort(arr,0,N-1,ans);
        return ans;
    }
```