MERGE SORT

[3, 1, 2, 4, 1, 5, 2, 6, 4]   Divide & Merge

[3,1,2,4,1] [5,2,6,4]

[3,1,2] [4,1]

[4] [1]

[3,1] [2]

mid=1+1

[3] [1] → at this point low >= high

1 >= 1

if len(arr) == 1

Now merge them

[1,3]

Now

[1,3] [2] = [1,2,3]

thats how we get sorted array.

Pesudo Code

```
merge sort (arr, low, high)
{
    mid = (low + high)/2
    mergeSort( arr, low, mid)
    merge sort (arr, mid+1, high)
    merge (arr, low, mid, high)

}
```

→ Base case

```
if len(arr) == 1   return arr
```

Or in this way (optimized)

```
if (low >= high) return
```

· merge code

Code in files

3, 1, 2, 4, 1, 5, 6, 2, 4
0 1 2 3 4 5 6 7 8

1 1 2 3 4
3, 1, 2, 4, 1
0 1 2 3 4

2, 4, 5, 6 real arr sorted
5 6 2 4 arr
5 6 7 8 idx

Suppose these two are the 2nd last result.

left
1, 1, 2, 3, 4

right
2, 4, 5, 6

low mid mid+1 high

arr = [ ]
left = low
right = mid+1

while left <= mid and right <= high.

```
if (arr[left] <= arr[right])
    temp.add (arr[left])
    left ++
else
    temp.add (arr[right])
    right ++.
```

Count Inversions.

arr = [5, 3, 2, 4, 1]

i < j and (arr[i] > arr[j])
count + 1

thats the problem
on every element next
less element is count

Brute force:

count = 0

```
for (i = 0 -> n-1)
    for (j = i+1 -> n-1)
        if (arr [i] > arr[j] ) {
                count + 1
        }
```

this is the TC = $O(n^2)$
SC = $O(1)$

But we have to reduce the
time complexity.

[2, 3, 5, 6]   [2, 2, 4, 4, 8]

(3, 2)
(5, 2)
(5, 4)
(6, 2)
(6, 4)
⋮

So u stand here you go through every element in other array and count + 1
same for 3 then 5, then 6
So standing at 3 we can see that every element on the right sorted array is a comparison.

~~3, 2~~,

So if we try to merge them ~~first~~ ~~(2, 2)~~ and on every left > right count + 1

2 then 3 is > 2

res = [2]

3 then again

res = [2, 2, 2]
+3 +3

then 3 and 4
just
[2, 2, 2, 3]
then 5 +3+3+2+2
[2, 2, 2, 3, 4, 4]
[2, 2, 2, 3, 4, 4, 5]
then 6
[2, 2, 2, 3, 4, 4, 5, 6]
then 8
[2, 2, 2, 3, 4, 4, 5, 6, 8] iteration = 9

if we can just
devide the array in 2 sorted
array and specifically at that
point we can count.

if we see

like at point
3 when compared with
[2, 2] we did +3 +3
So
if i = 1 len = ~~3~~ 4
4 - 1 = 3 added
again
when
3 > 2
4 - 1 = 3 added
So the ~~good~~ question
is answered
on every sorting
inv += len(left) - i

walk through.

[5, 3, 2, 4, 1]

(5, 3, 2)　　　　[4, 1]

[5, 3]　　[2]　　　[4]　[1]

(3,5)

[5]　[3]

at this poin

[5]　[3]

5 > 3

count + 1

Now

[3, 5]　[2]　　len(left) = 2 − i = 2 − 0 = 2

3 > 2

+2 ⇒ count = 3

[2, 3, 5]

Now　　[4]　[1]

count = 4

[2, 3, 5]　　　[1, 4]

Now

[2, 3, 5] [1, 4]

2 > 1

count 4 + 3 = 7

Now

[1, 2] 3

Now

[1, 2, 3] 5 = 3 - 2 = 1

count = 8

So thats how it works.