DAY - 34

(1) Count Inversions in a list/array.

Approach :- Using Enhance Merge sort

Algorithm :-

-> The idea is similar to merge sort, divide the array into two equal or almost equal halves in each step until the base case is reached.

-> Create a function merge that counts the number of inversions when two halves of array are merged, create two indices i and j, i is the index for first half and j is an index of the second half.

if a[i] is greater than a[j], then there are (mid-i) inversions.

because left-subarray a[i+1], a[i+2] ... a[mid] will be greater than a[j].

-> Create a recursive function to divide the array into halves and find the answer by summing the number of inversions in first half, number of inversions in the second half and no. of invs

by merging the two.
→ the base case of recursion is when there is only one element in the given half.
→ return the answer.

code part 1 -

```
arr → given array/list
n → len of arr

def mergeSort (arr, n):  # to use inversion count
    temp_arr = [0] * n
    return countInv (arr, temp_arr, 0, n-1)

def countInv(arr, temp_arr, left, right):
    # this fun will count inversion
    # count using mergesort.
    inv_count = 0
    if left < right:
        mid = (left + right) // 2
        inv_count += countInv(arr, temp_arr
                              left, mid)
        inv_count += countInv(arr, temp_arr,
                              mid+1, right)
        inv_count += merge (arr, temp_arr,
                            left, mid, right)
        # note: 'merge' is a different
    # fun that will merge two subarrays
    # in a single sorted subarray.

    return inv_count.
```

(2) Sliding Window Maximum [Maximum of all subarrays of size k]

Approach :-

The idea is very basic run a nested loop, the outer loop which will mark the starting pointing of the subarray of length k, the inner loop will run from the starting index to index+k, k elements from starting index and print the maximum element among these k elements.

Algorithm :-

→ Create a nested loop, the outer loop from starting index to n-k'th elements.

The inner loop will run for k iterations.

→ Create a variable to store the maximum of k elements traversed by the inner loop.

→ Find the max. of k elements traversed by the inner loop.

→ Print the maximum element in every iteration of outer loop.

Implementation:-

arr → given array
n - len of arr

```
def getMax (arr, n, k):
    max = 0

    for i in range (n-k+1):
        max = arr[i]

        for j in range (1, k):
            if arr[i+j] > max:
                max = arr[i+j]
        print (str(max) + " ", end = '')
```