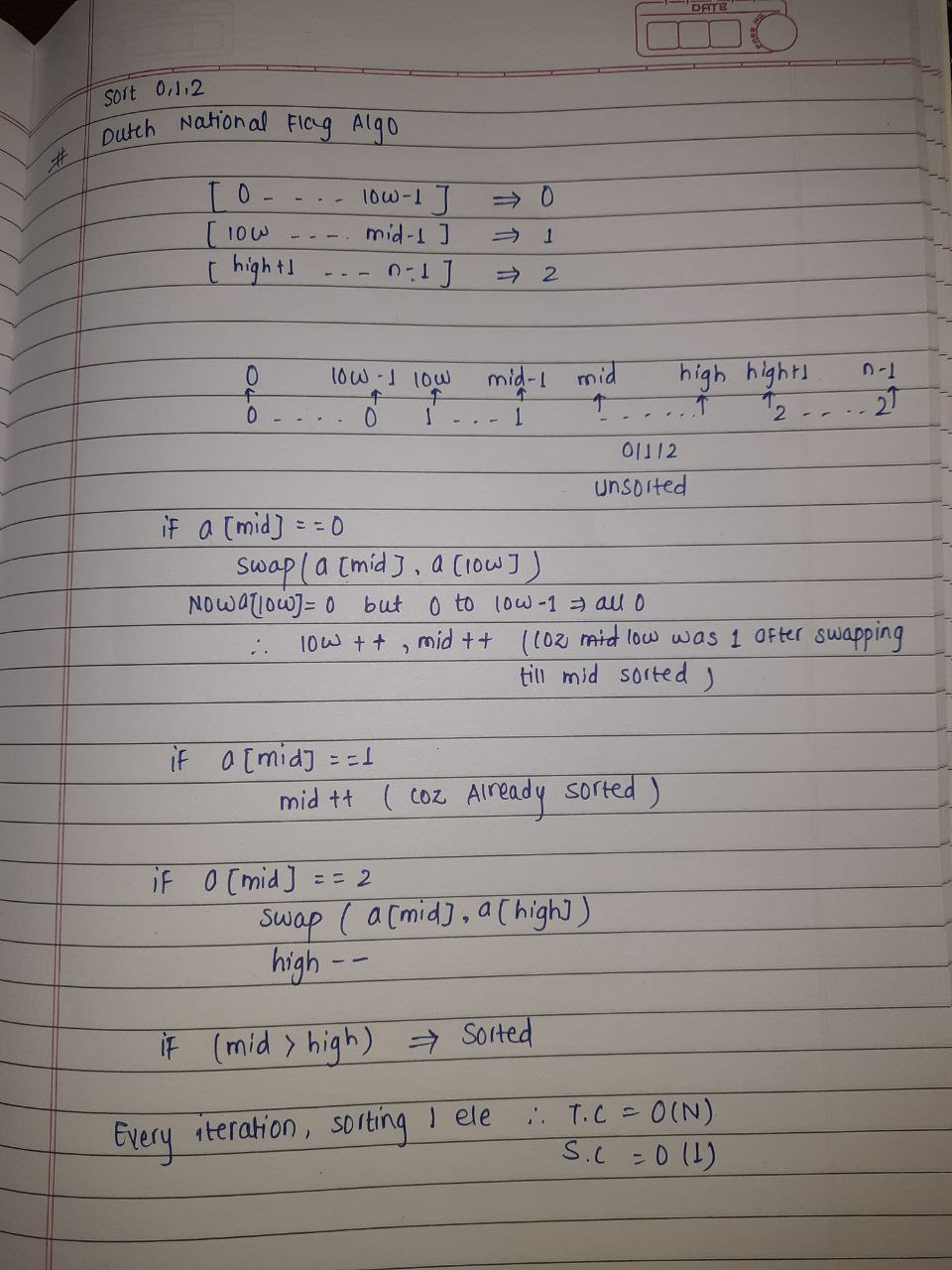
**Sort 0,1,2**

A1 : Sort the array

A2 : take 0 1 2 count and create array according to counts O(2N)

A3 :

**Dutch National Flag Algo**



**Majority Elements ( freq of ele > n/2 )**

A1 : Direct compare O(**N2**)

A2 : store freq in map and compare TC : NlogN + N SC N

A3:

**Moore's Voting Algorithm**

Take the element if matches c++ if not c—if c=0 start with new element

And the last element will be your majority element

TC : O(N+N) another N is for checking if not N only

**Maximum SubArray Sum**

A1 : Traverse through all subarrays and find max sum O(N3) and to O(N2)

A2 :

**Kadane's Algorithm**

If sum<0 sum=0 if sum>0 add the elements and keep tracking maximum

**Rearrange Array Element by Sign**

A1 : Take the +ve element and -ve element in 2 separate arrays and merge sequence wise. TC O(2N) SC N

A2 : Take 2 variables posIndex and negIndex start from 0 and 1 resp. and if nums[i]<0 insert at negIndex else posIndex only when given that pos and neg numbers are equal i.e n/2 TC N SC N

If given pos and neg are unequal In numbers then the brute approach of above is the optimal.

Just start with smaller size array and then add the leftovers of bigger array

**Leaders in an Array**

Leader is the element which has all small elements to the right of it.

A1 : Traverse for every element I to n j i+1 to n that do you get any large number than a[i] if yes no leader else leader

A2 : Traverse from end if current element is greater than maxi then push the element in the leaders array and the array at the end will be sorted.

TC N SC 1

**Longest Consecutive Sequence**

A1 : Take the element and traverse for element + 1 in the whole array and keep the track of longest while you keep getting element + 1

TC N2

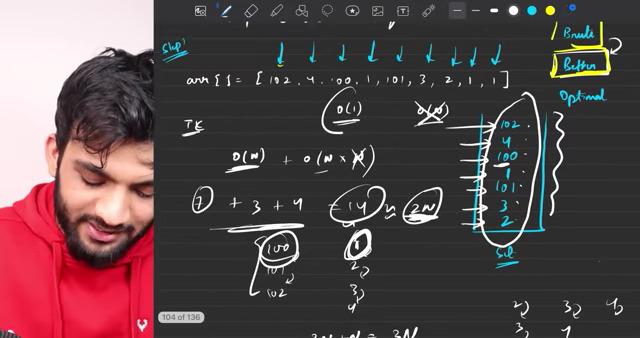
A2 : Sort the array after sorting keep track of lastEle if arr[i]-1==lastEle c++ and If not c=1 change the lastEle to that arr[i] and keep changing to longest as max(longest,cnt)

TC NlogN + N

A3 : put all elements in unordered set , traverse the set that if element -1 exits in set then continue and if we did not then this is the starting point keep c++ while the element +1 is present and keep longest updating using max()

TC O(N) for inserting+O(N + 2N) for traversing here the inner while loop will run approx. 2N times

Under the assumption that set will take O(1) for operations SC O(N)



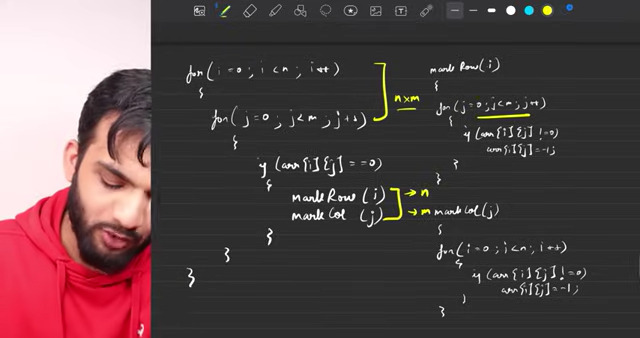
**Set Matrix Zeros**

If a[i][j] == 0 mark the whole row and col as 0

**A1** : if a[i][j]==0 mark the row and col as -1 && a[i][j]!=0 after that again traverse thro array and change the -1 to 0

If we initially replace it with 0 then the ambiguity might have arisen.

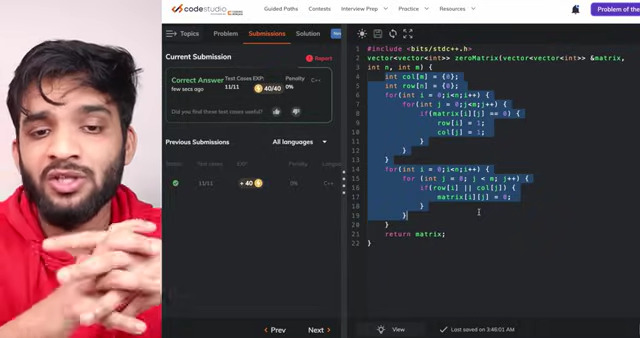
TC : O(N\*M \*(N+M))+O(N\*M)



A2 : Instead of traversing each row and col we will keep track of rows and cols which contains 0 and after tracking we will traverse the matrix and will update the elements, for that two arrays will need of n and m size resp to keep a track

T.C : O(2\*N\*M)

S.C : O(N) + O(M)



A3: Here we are keep tracking of 0's in given matrix only in 0th row and 0th col but (0,0) will overlap so we will take one variable to represent col1

now change the array elements according to 0th row and 0th col so first start iterating from 1st row and 1st col to avoid (0,0) field

for (0,0) field the 0th row and 0th col depends on it if we change change the 0th col, but the 0th row depends on it so first change the 0th row and then with the help of col0 change 0th col

TC: O(2\*N\*M)

SC: O(1)

**Rotate Matrix / Image by 900**

A1: Observe the pattern how the answer is differed by org matrix

i.e a[i][j] => ans[j][n-i-1]

TC : O(N2) SC : O(N2)

A2: Take transpose of matrix and reverse the every row

Transpose : diagonal elements will be same.

I 0->N j i+1->N swap(a[i][j],a[j][i])

TC: O(N/2 \* N/2)+ O(N\* N/2)

**Majority Elements ( freq of ele > n/3 )**

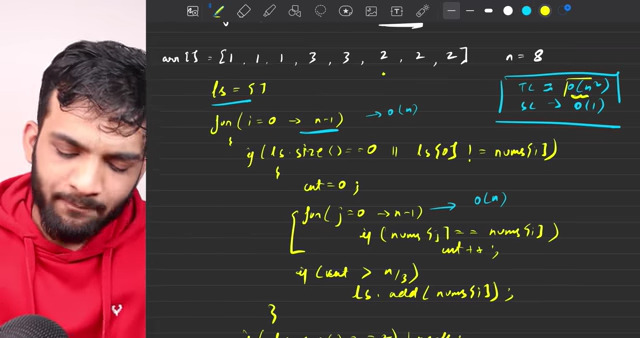
In this case there can be maximum 2 elements freq > n/3 and minimum 0

How maximum 2 ?

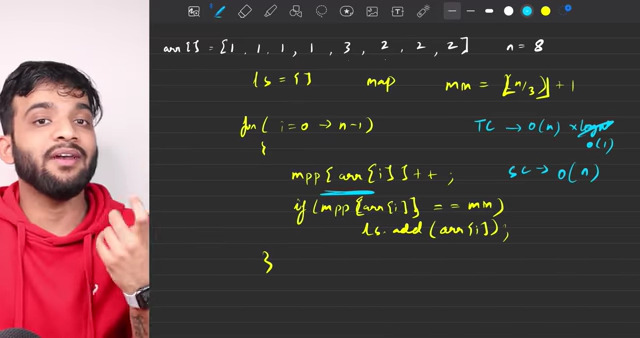
Lets take n=8 8/3 = 2 we want greater than 2 so has to be 3 times

Then 3 + 3 + 3 = 9 but our N=8 so only 2 element greater than n/3 can be present in array.

A1: compare element if got any element n>3 put it in the array.



A2: Use map to store the freq.



A3 : Same approach as n/2 just add conditions and 1 edge case of while assigning new element the other one must not be equal to that current element.

**3 Sum**

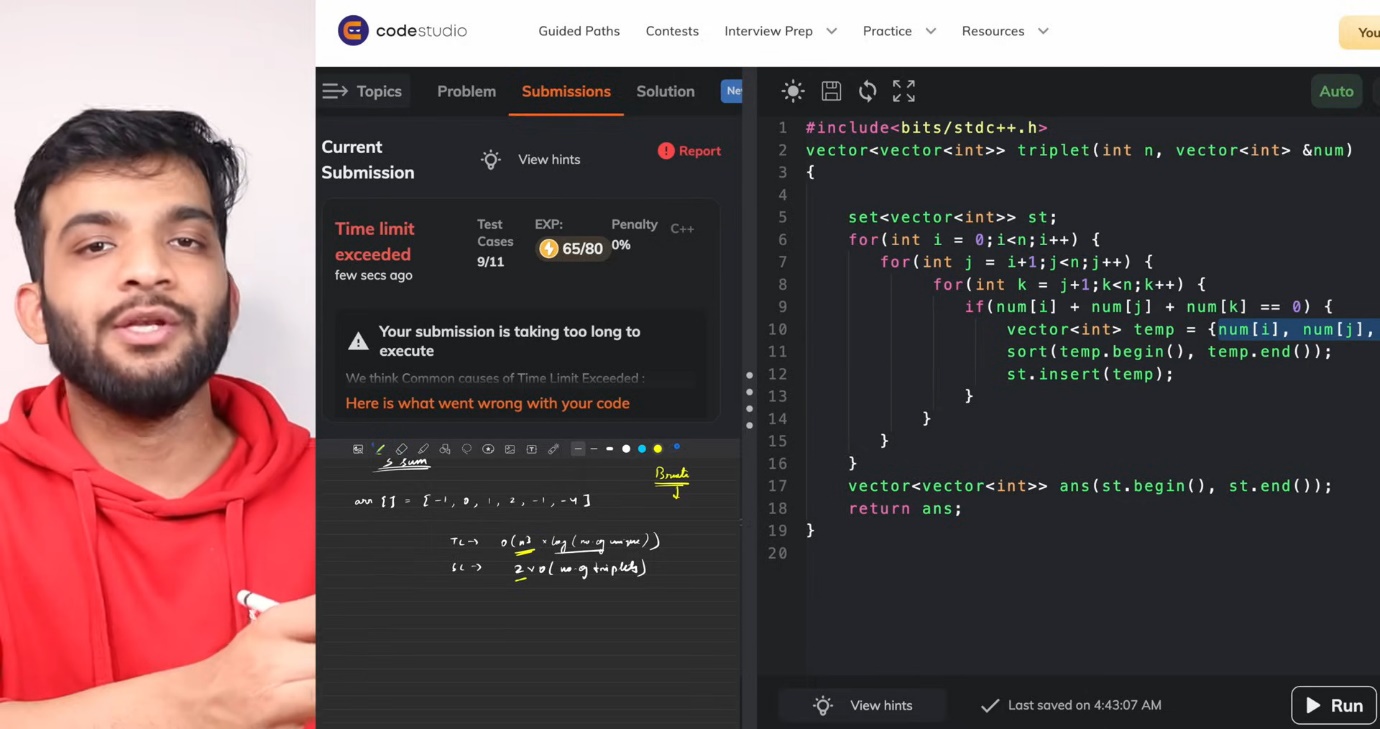
Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

A1 : Traverse for all triplets I 0->n j i+1->n k j+1 ->n

TC : O(N3 \* log(No. of Unique triplets))

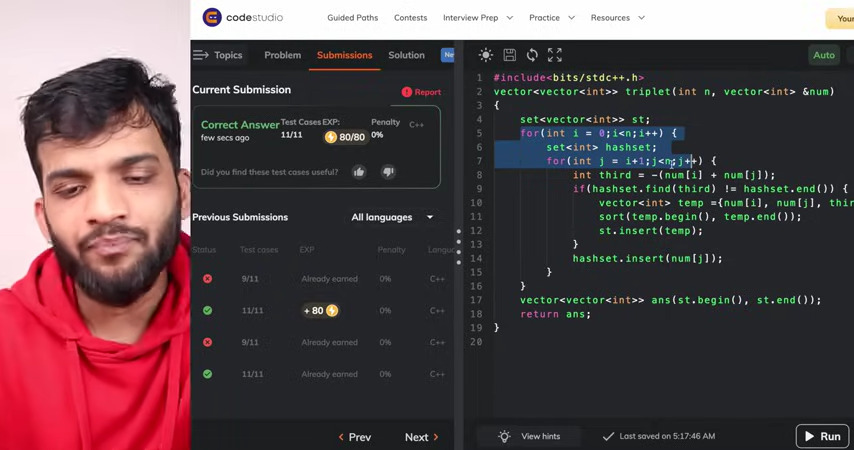
SC : O(2\* No. of Unique triplets)



A2: a[i]+a[j]+a[k] = 0 a[k]= - (a[i]+a[j])

If a = [-1 0 1 2 -1 -4]

Now If a[i]=2 a[j]=-4 -(2+(-4))=2 and 2 is present in map but that 2 we are using in our equation so to avoid this while traversing from I to j put only those in hashmap.



TC : O(N2 \* log M) log M due to the size of set if unordered could be the 1

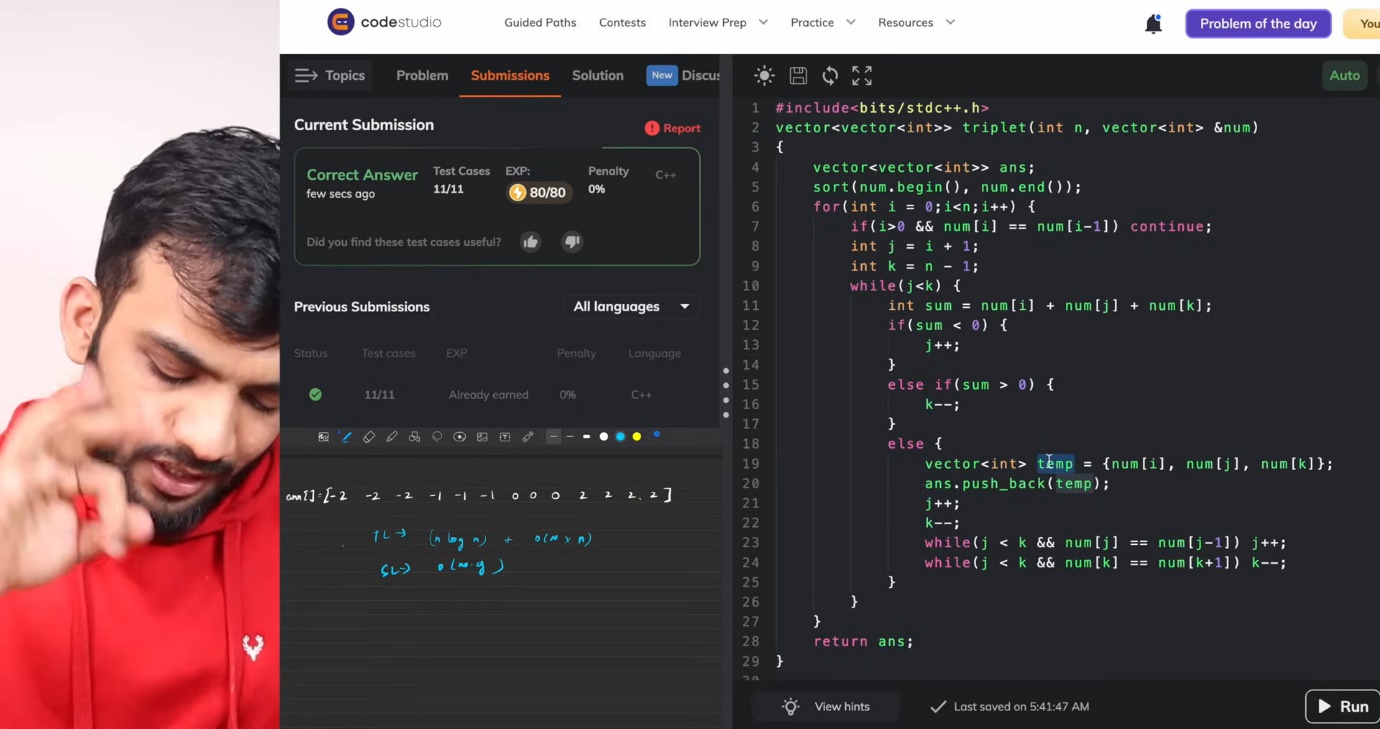
SC : O(N) for map +2\*O(No. of Unique triplets)

A3:

Sort the array

Take 3 pointers I,j,k keep I constant and move j and k

If I is same as previous continue and same for j and k in their moving manner

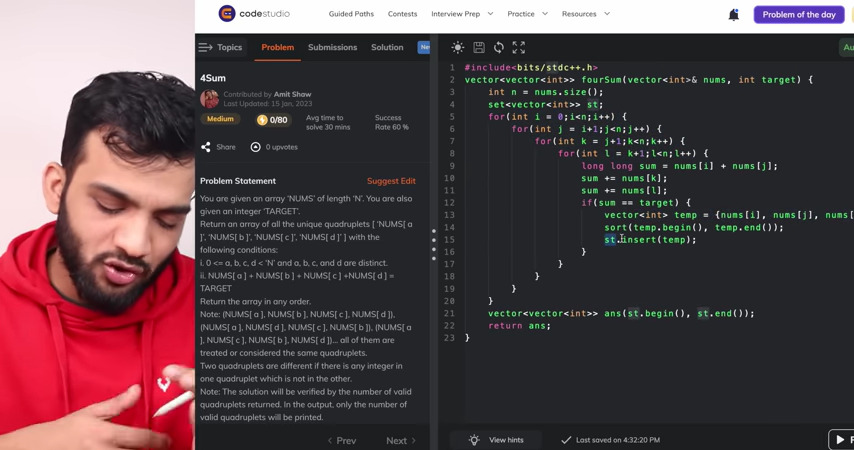


TC : O(N log N) to sort + O(N2)

SC: O(1)

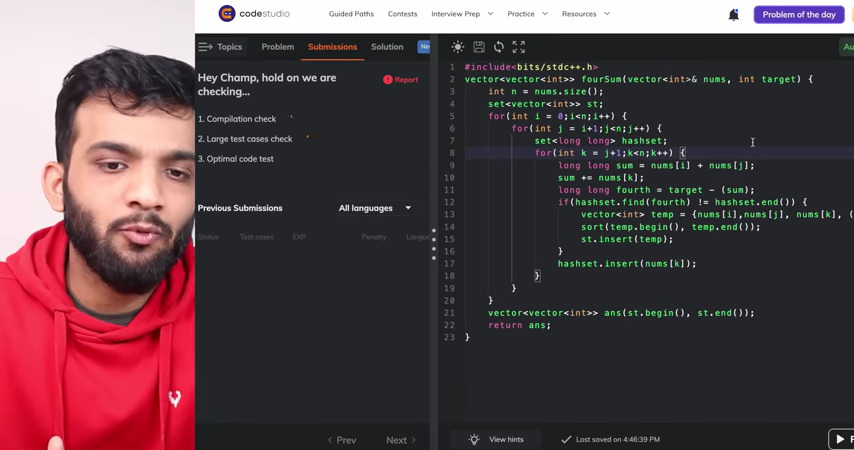
**4 Sum**

A1:



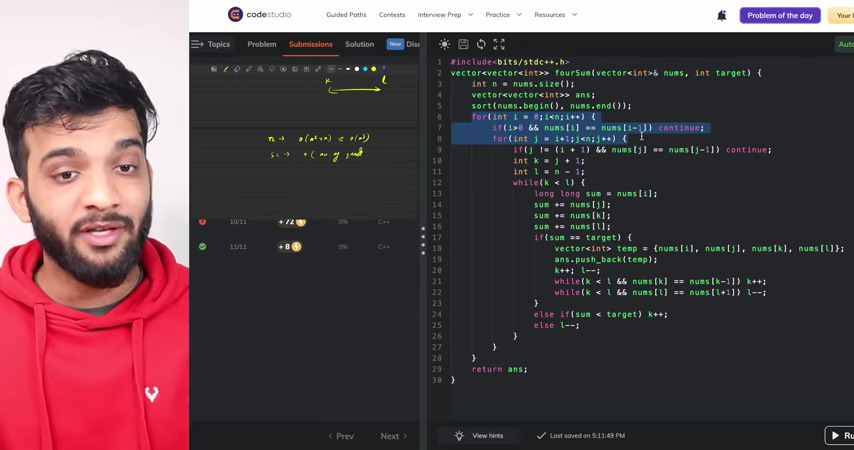
TC : O(N4) SC : O(No. of Quads\*2)

A2: Same as A2 of 3 Sum



TC : O(N3 \* log M) SC : O(N) for map+ O(No. of Quads\*2)

A3: Same as A3 of 3 Sum



TC : O(N log N) to sort + O(N3)

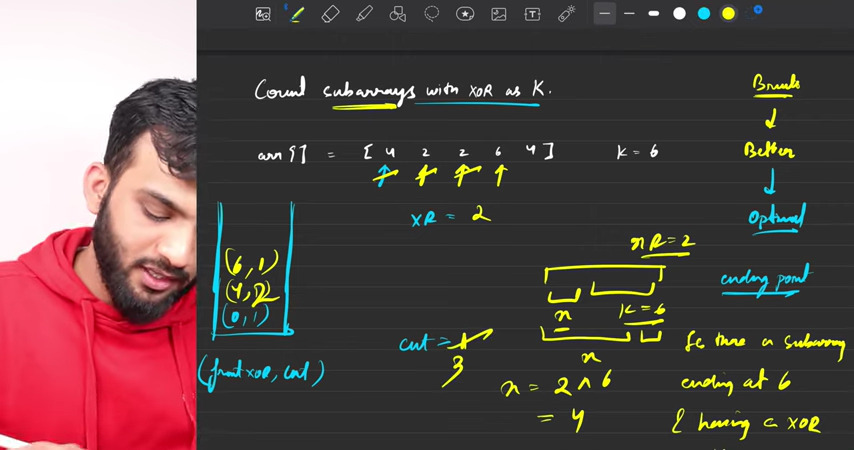
SC: O(1)

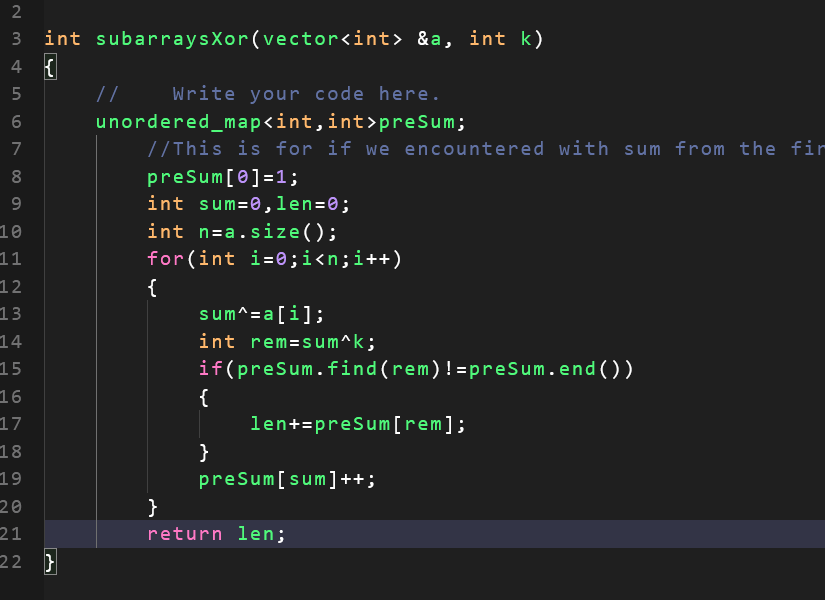
**Count Subarrays with XOR as K**

A1:Generate all the subarrays and calculate the XOR and compare

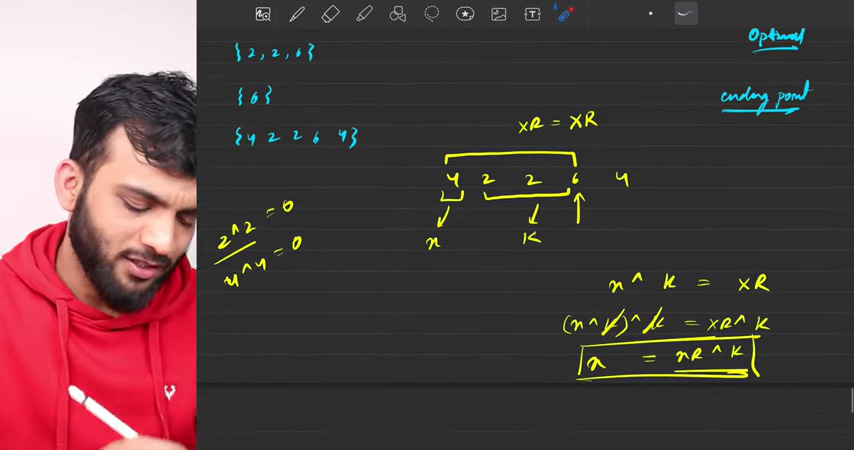
TC : O(N3) => O(N2)

A2: Same we did it in subarrays with sum K





Why sum^k ?

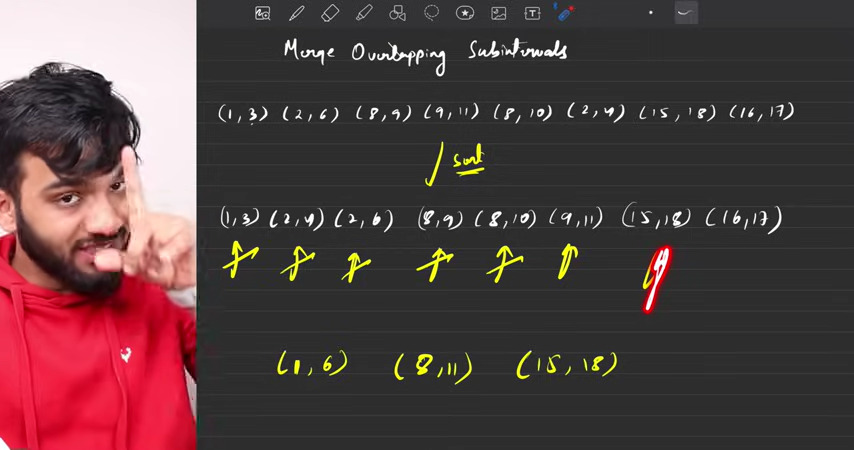
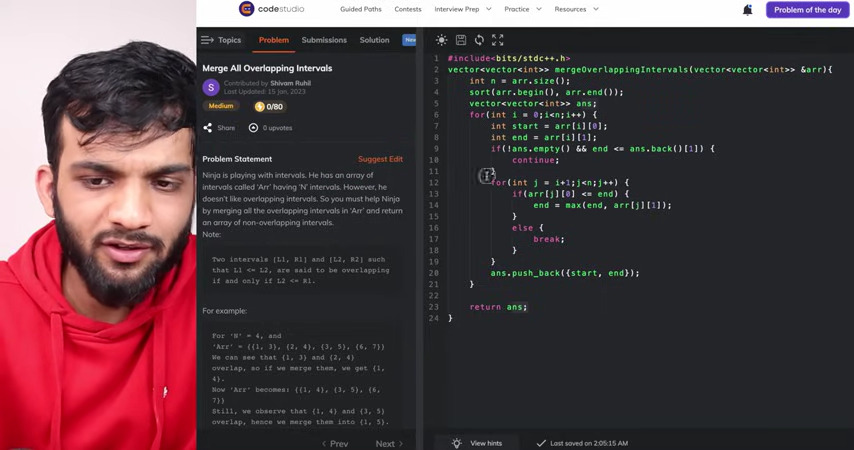


**Is there a subarray ending at 6 and having XOR as K**

**Use the formula and search for the x**

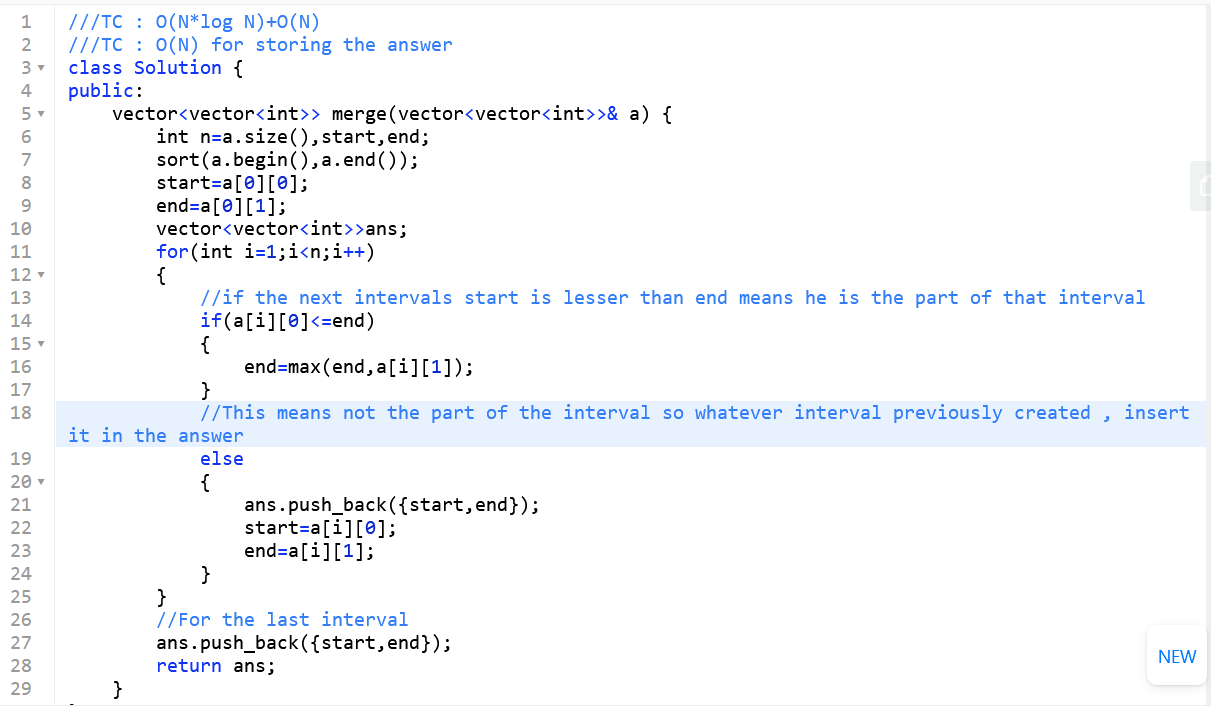
**Merge Intervals**

A1 : check if current start is lesser than the previous end or not



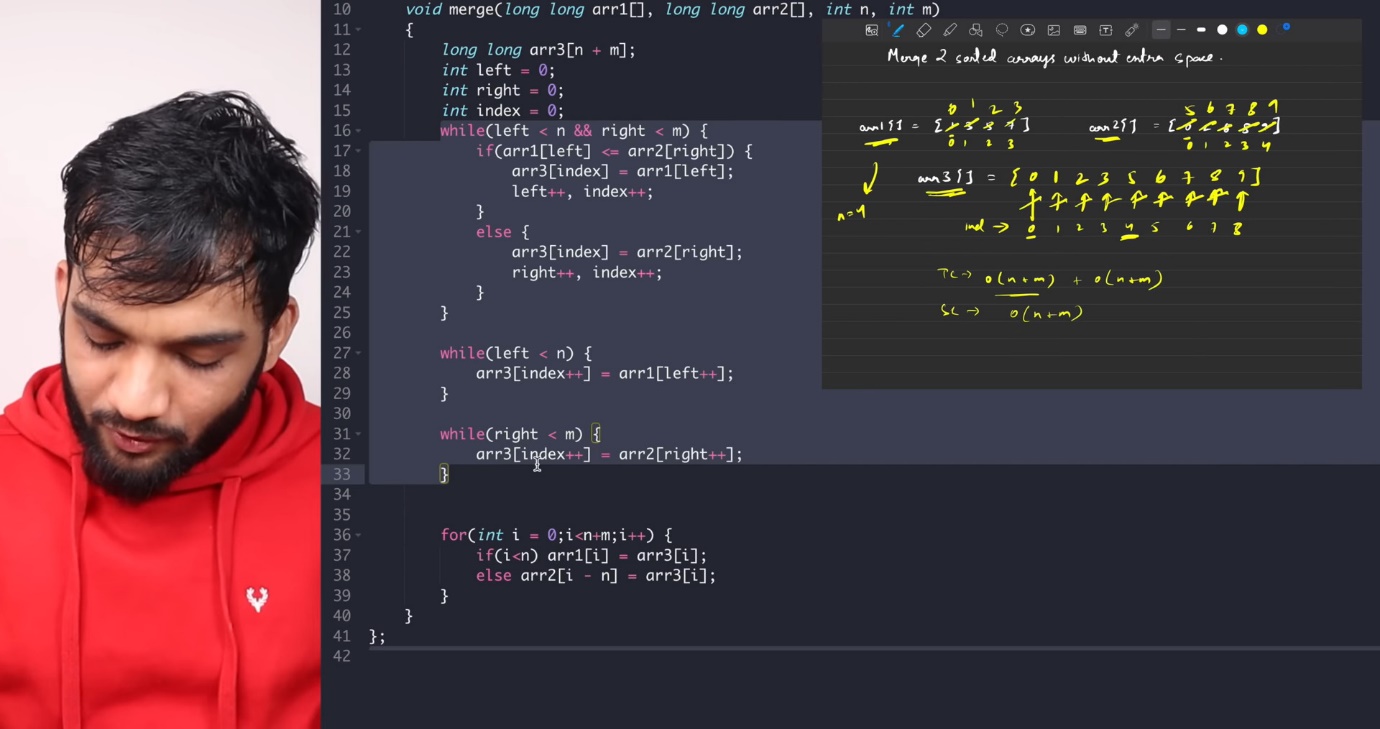
TC : O(N log N) + O(2N) as in that loop every element is getting traverse 2 times

A3: Instead of traversing for each interval , while traversing only update the intervals

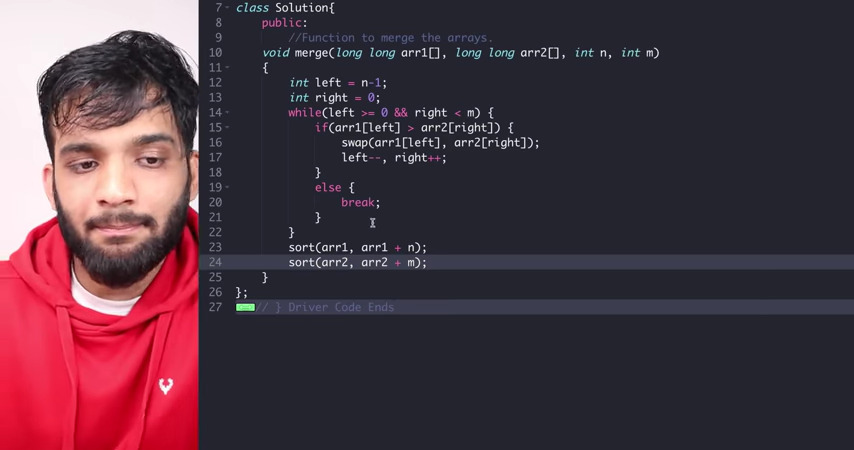


**Merge Two Sorted Array without using Extra space**

A1: Using 2 pointer Approach but with extra memory



A2: keep pointer at end for 1st array and pointer at start for 2nd array, keep comparing arr1[i]>arr2[j] if yes swap it, this will make the elements in their correct array but not in the sorted manner, for it sort the arrays



TC : O(min(N,M)) + NlogN + MlogM

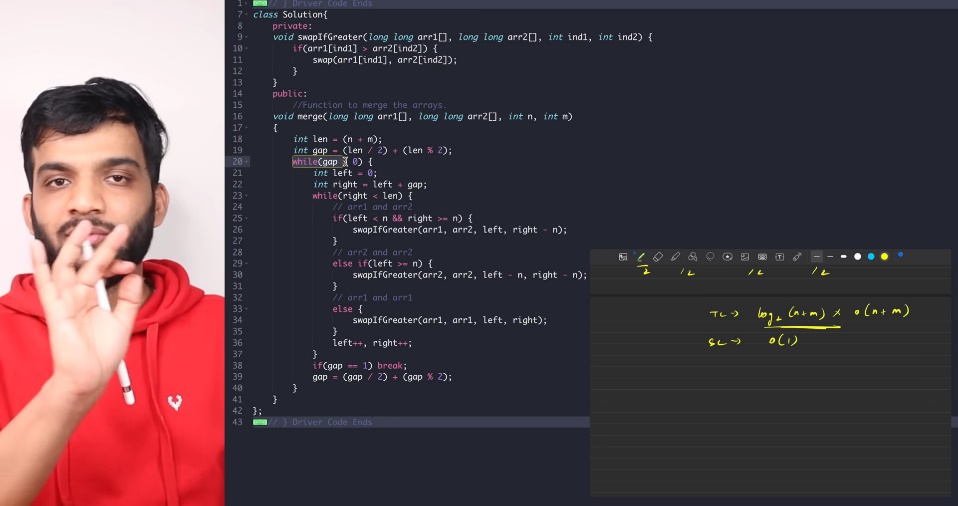
Why min(N,M) ?

If we move all the elements from one array to other so it will take minimum of both

A3: Gap Method

Gap=ceil (m+n)/2

And keep that gap in between I and j and compare a1[i] and a2[j] and swap , while(gap!=0)

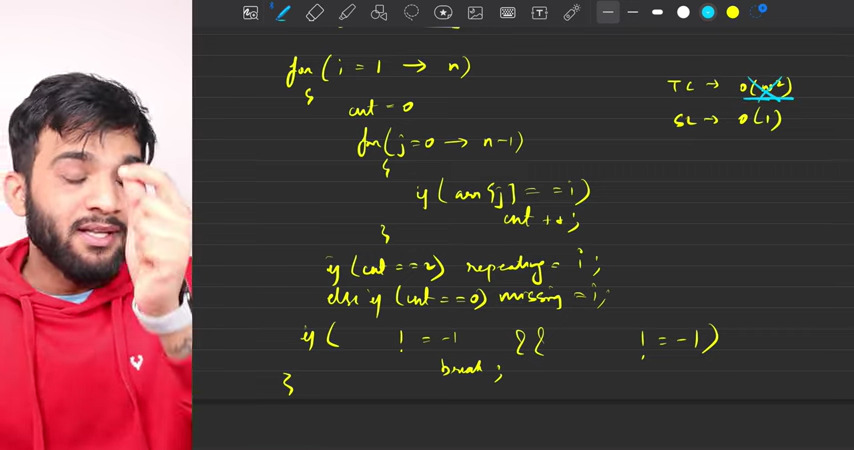


TC: log2(N+M) + (N+m) SC 1

Find Missing and Repeated Number(2 times)

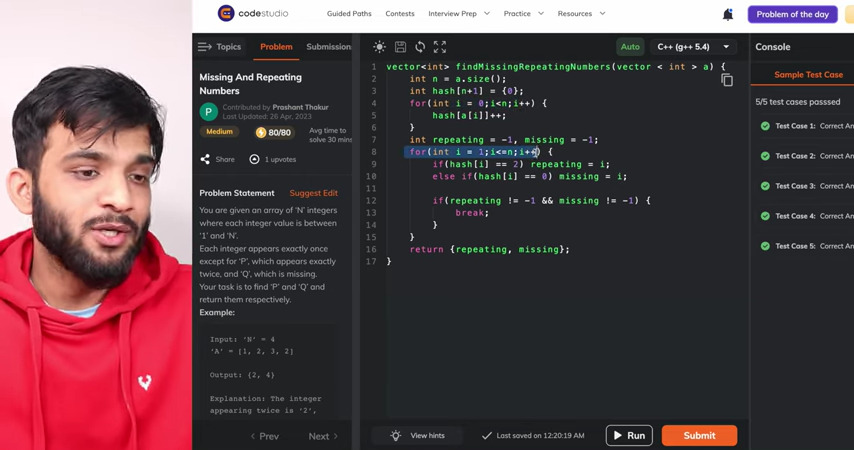
Given 1 to N numbers

A1: Declare 2 variables and keep track of counts of elements



TC : O(N2) SC: O(1)

A2: Create a Array of n+1 size and keep count of every element in that array by index if visited array element is 0 then missing no. and if 2 repeating number



TC : O(2N) SC : O(N)

A3: Here you have to create two simultaneous equation to get the value of x and y

X = repeating no. y=missing no.

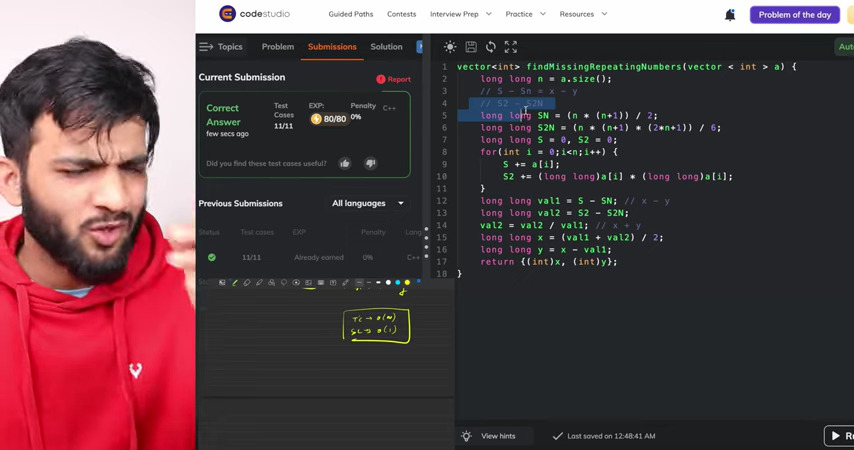
x-y=Sum of array – sum of 1 to n

x2-y2 =sum of squares of array elements – sum of squares of 1 to n

sum of 1 to n: n\*(n+1) / 2

sum of squares of 1 to n : n\*(n+1)\*(2n+1) / 6

After solving all this equations the code will match



TC : O(N) SC : O(1)

**Count Inversions**

I<j && a[i]>a[j]

Given an array of integers. Find the Inversion Count in the array.  Two array elements arr[i] and arr[j] form an inversion if arr[i] > arr[j] and i < j.

***Inversion Count***: For an array, inversion count indicates how far (or close) the array is from being sorted. If the array is already sorted then the inversion count is 0.  
If an array is sorted in the reverse order, then the inversion count is the maximum.

**A1:** Traverse a whole array for each index and find element if found c++;

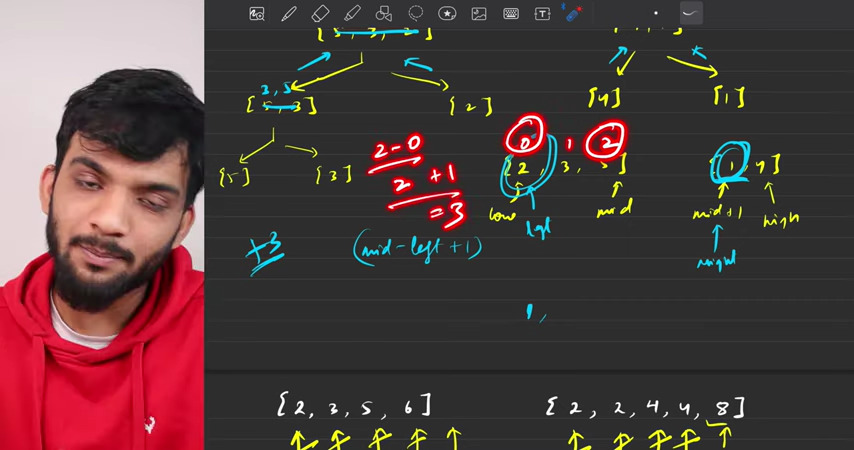
TC : O(N2) SC: O(1)

**A2:** Use the logic of merge sort while merging only compare the elements if elements is in sorted order at the time of comparing if a1 = 2 5 6 a2=1 4

2>1 so all elements ahead of 2 will make a pair i.e 2,1 5,1 6,1 move pointer to 4 when comes to i=5 , it will form pair 5,4 6,4

TC :O(N logN) SC : O(N) as we are storing as temporary

Why mid-left+1 ?



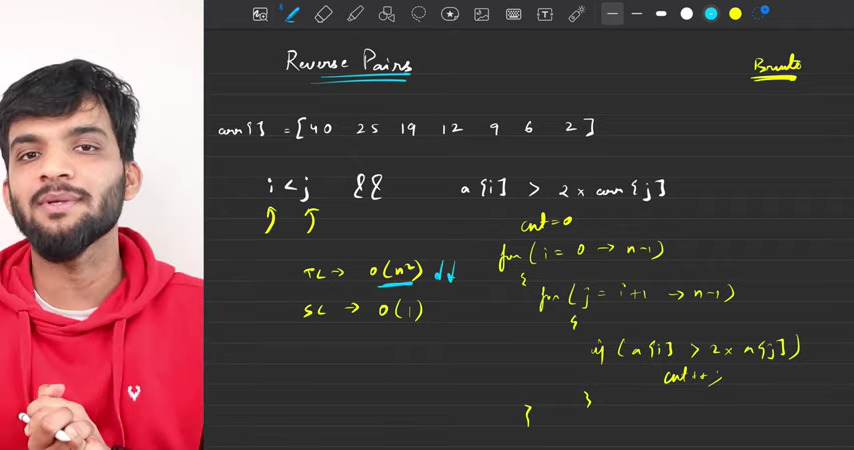
**Reverse Pairs**

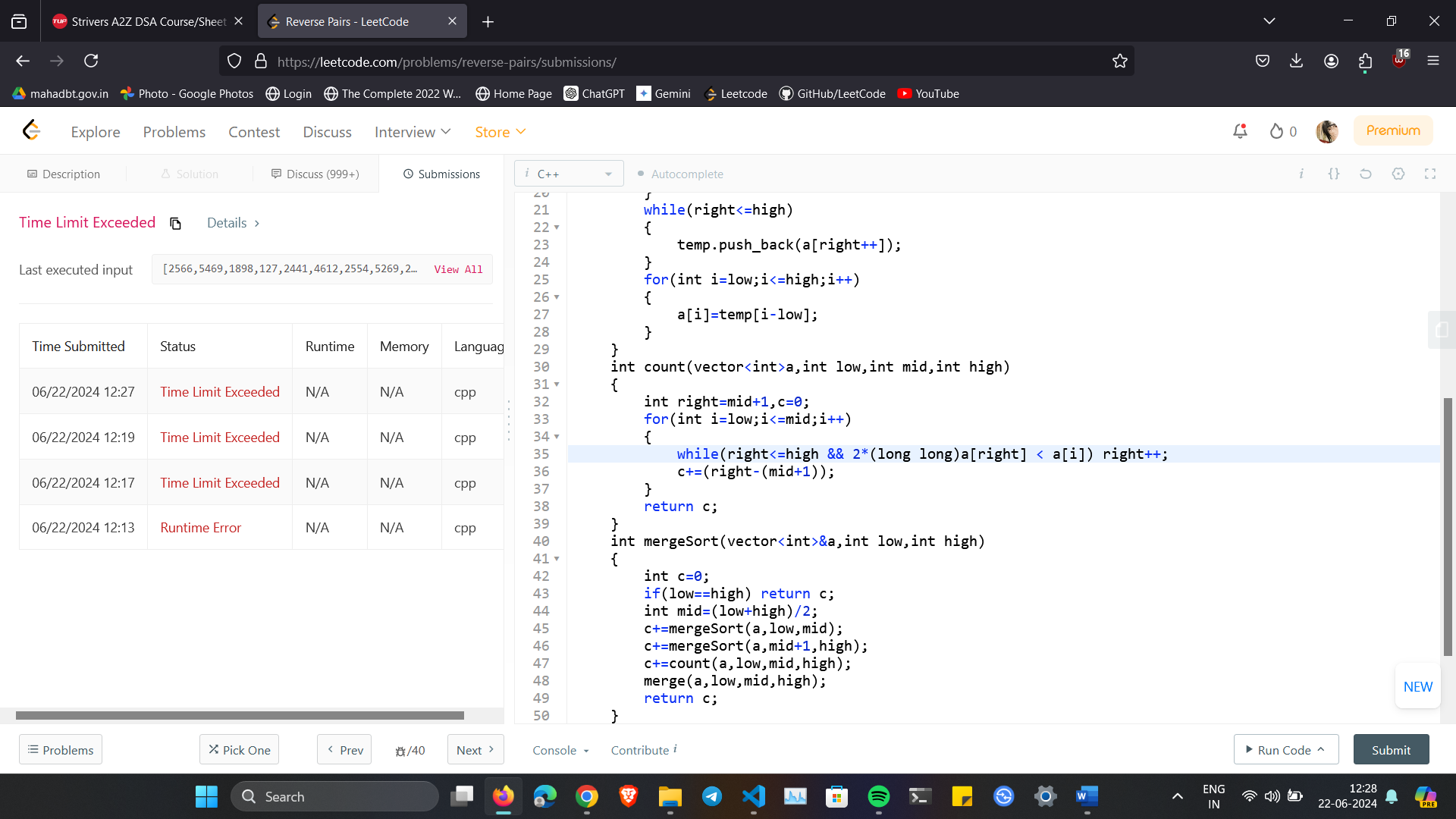
Given an integer array nums, return *the number of* ***reverse pairs*** *in the array*.

A **reverse pair** is a pair (i, j) where:

* 0 <= i < j < nums.length and
* nums[i] > 2 \* nums[j].

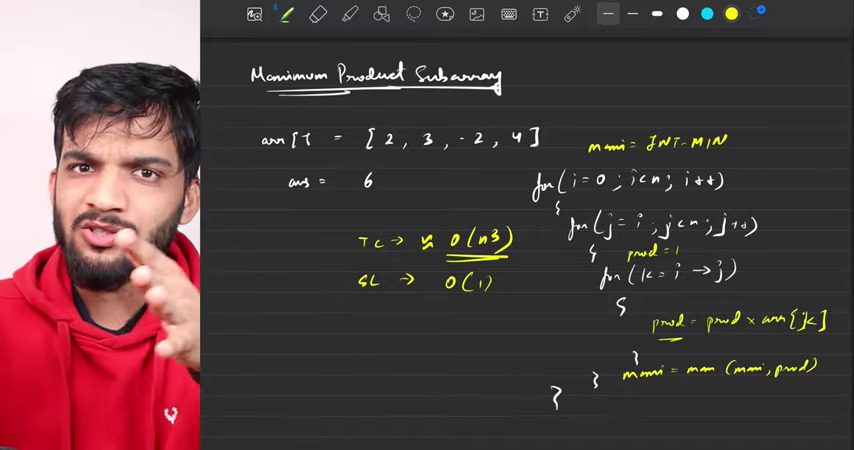
A1: Traverse for each element through whole array and check for condition that nums[i] > 2 \* nums[j]



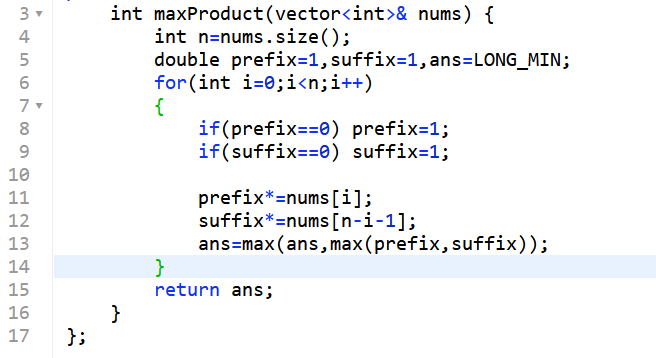
A2: 

**Maximum Product Subarray**

A1: N3 => N2



A2: iterate from start , iterate from end if 0 comes set pre and suff to 1 and keep tracking of max



TC : O(N) SC : O(1)