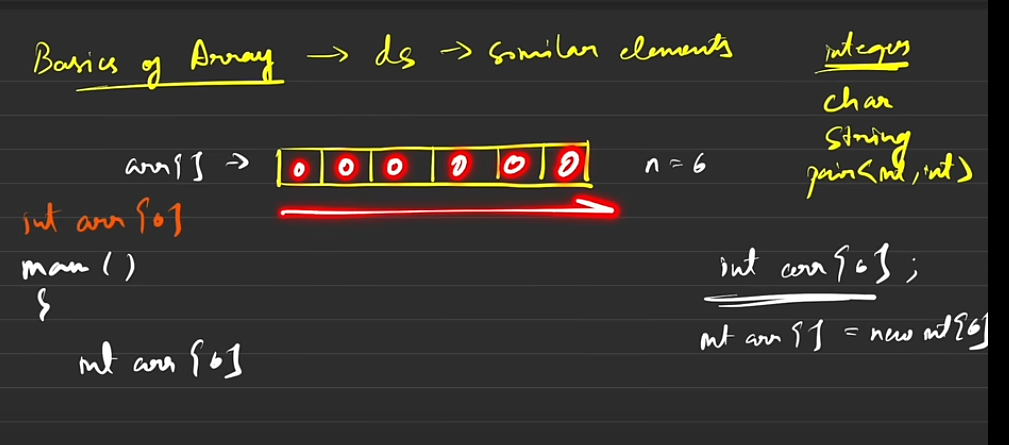
**ARRAY BASICS**

* Array is nothing but a data structure that contains similar elements only
* Now, this means that it can contain one particular type of data only
* Now, it can contain elements of any datatype but all elements that it contains must be of same datatype.
* Ex- It can datatypes like int, char, string, pair<int,int> etc.



Array Declaration in C++ and Java respectively

Now, if you declare an array inside main() (without initialization) they contain garbage values but if we define them globally before main then it is initialized with zeroes

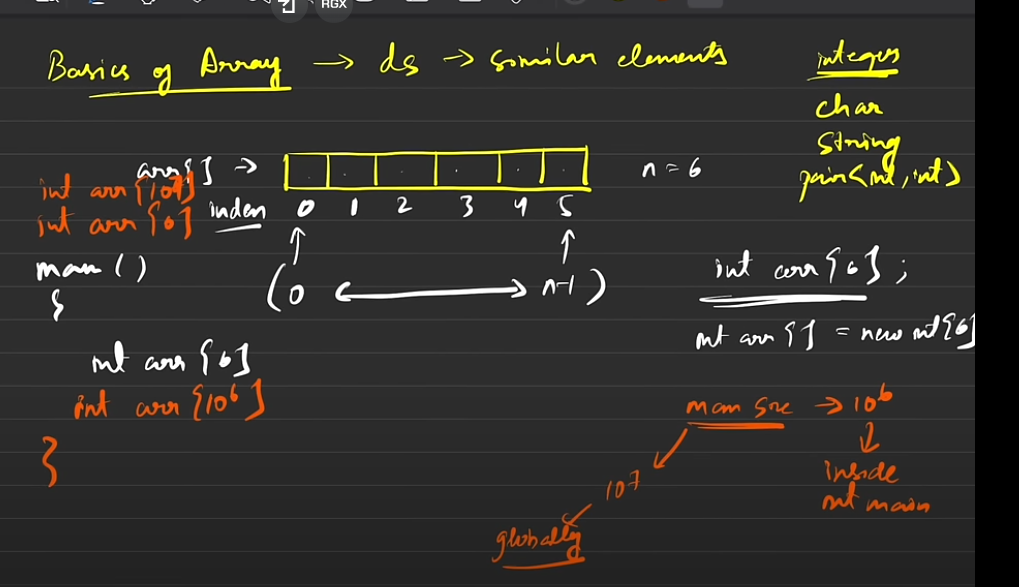
This happens in both C++ and Java

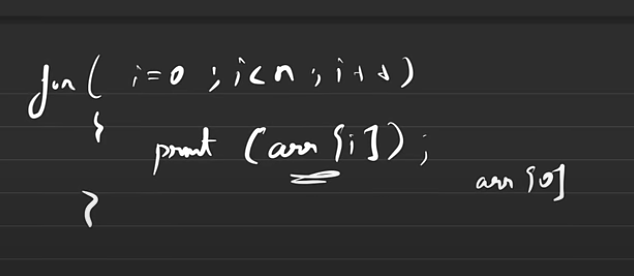


The Max. size of the array that can be declared is as follows:

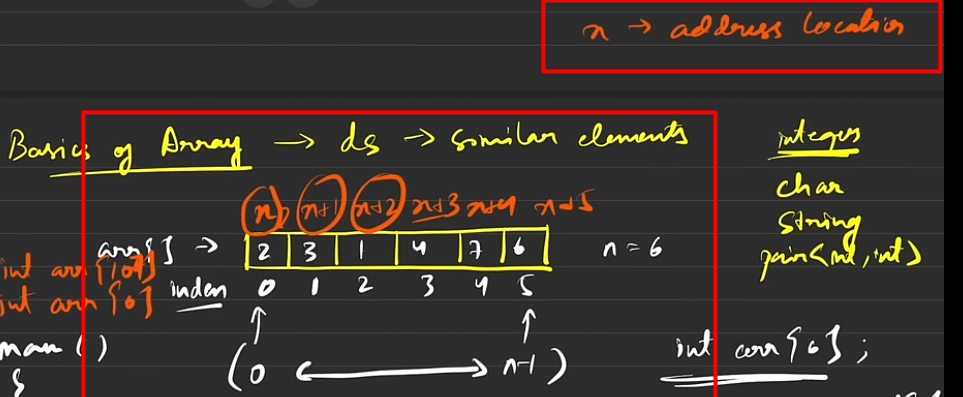
Inside main -> 10^6

Globally -> 10^7





Accessing an array through index and loops



Where is array stored in the computer?

Now, whenever you declare an array so it basically goes inside the computer’s memory and create a block of that size and the first element(having the index 0) is stored at some random ‘x’ address location (which cannot be predicted), but despite this we can predict that all other subsequent elements stored at the next indexes as they are always stored at (x+1), (x+2),(x+3) etc. memory locations. Thus, all the corresponding addresses are stored at contagious memory locations and since we cannot predict thereby addressing an array with memory address is not possible and we use indexes instead.

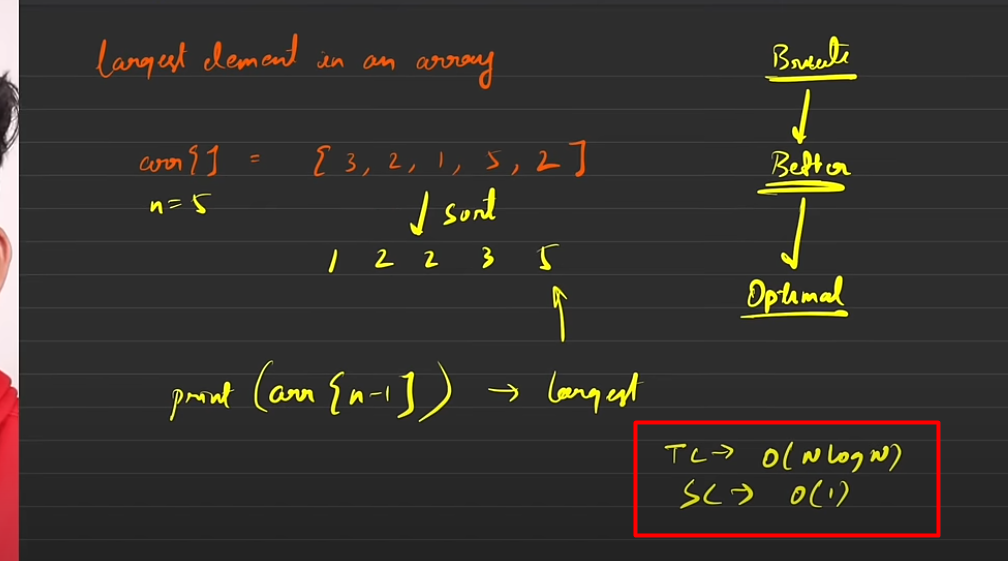
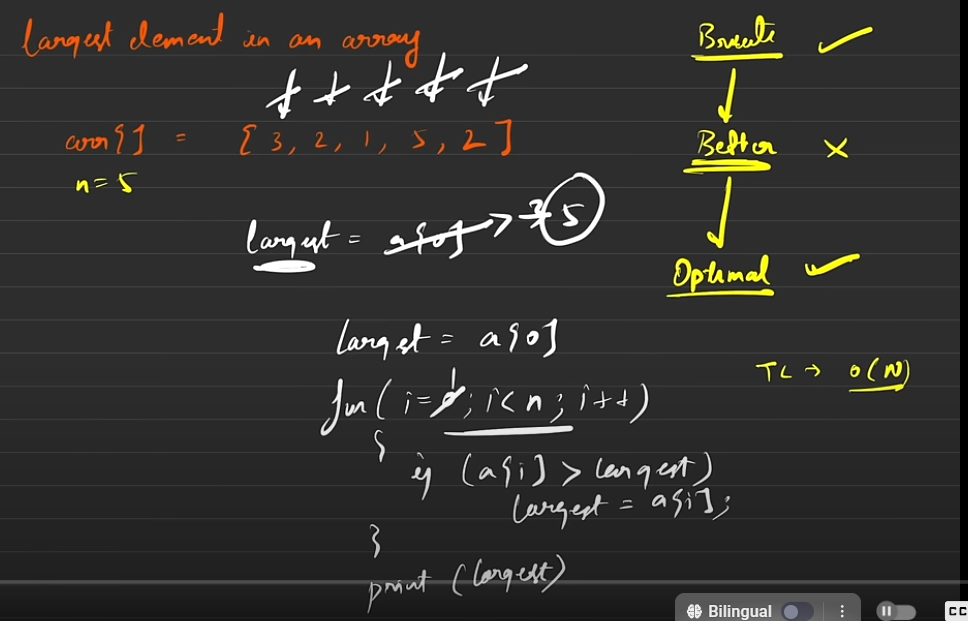


Now, Striver mentions that even if you know the optimal solution to hard problem, you shouldn't directly answer it to the interviewer instead you must build on that question, ask the interview about test cases and all the other required stuff and must understand the question thoroughly and then try to show the brute force approach first (a brute force approach is nothing but the first solution that comes to your mind and then you have to just optimise it further) followed by better solution and then optimise the better solution further to get the most optimal solution. (If you wanna add one layer after better solution like more better than then you can do that too)(The key is to show that you can think logically and critically and your approach to a problem is efficient and genuine (Even if you have to fake it then fake it))

But do to for harder problems and not for the easier ones

Q-1

Brute-Force Approach



While solving you might come across problems that use vectors or lists instead of arrays or arraylist in Java. Now, all the problems that can be solved using arrays can also be solved using them since although they are not array but they also store elements of same datatypes only(In fact we would even have more control over these DS(s)).

Now this, is categorized as the optimal solution as we have optimized the TC as O(N) now.

Tip: You can even start the loop from 1 to n now, as a[0] is already stored

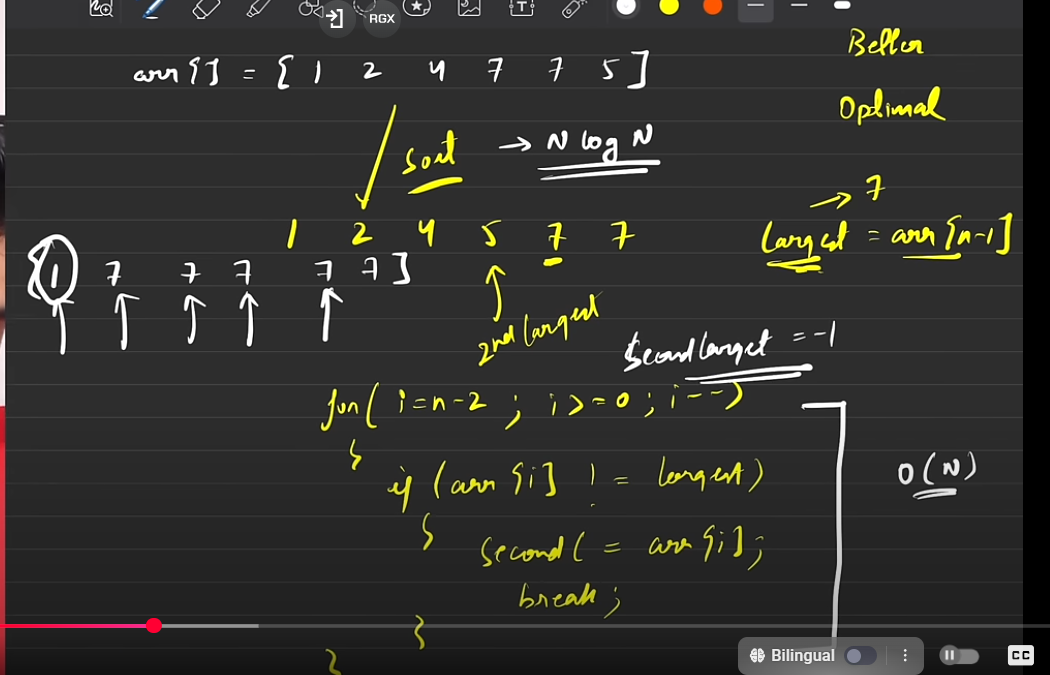
(although it won’t affect much but seeing Striver point out his told me that I must always look for marginal gains even the smallest ones if possible

Also, remember that in many cases although you have brute and optimal solutions, you still may or may not have a better solution and this question is among those cases.

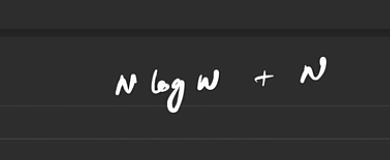
Here, the TC occurs from sorting i.e. merge or quick sort algo. etc

And SC is taken from quicksort (as of now) (As we have ignored the recursive space). If it would have been another sort say merge sort then it would have been O(n) SC.

Optimal Approach

Q-2

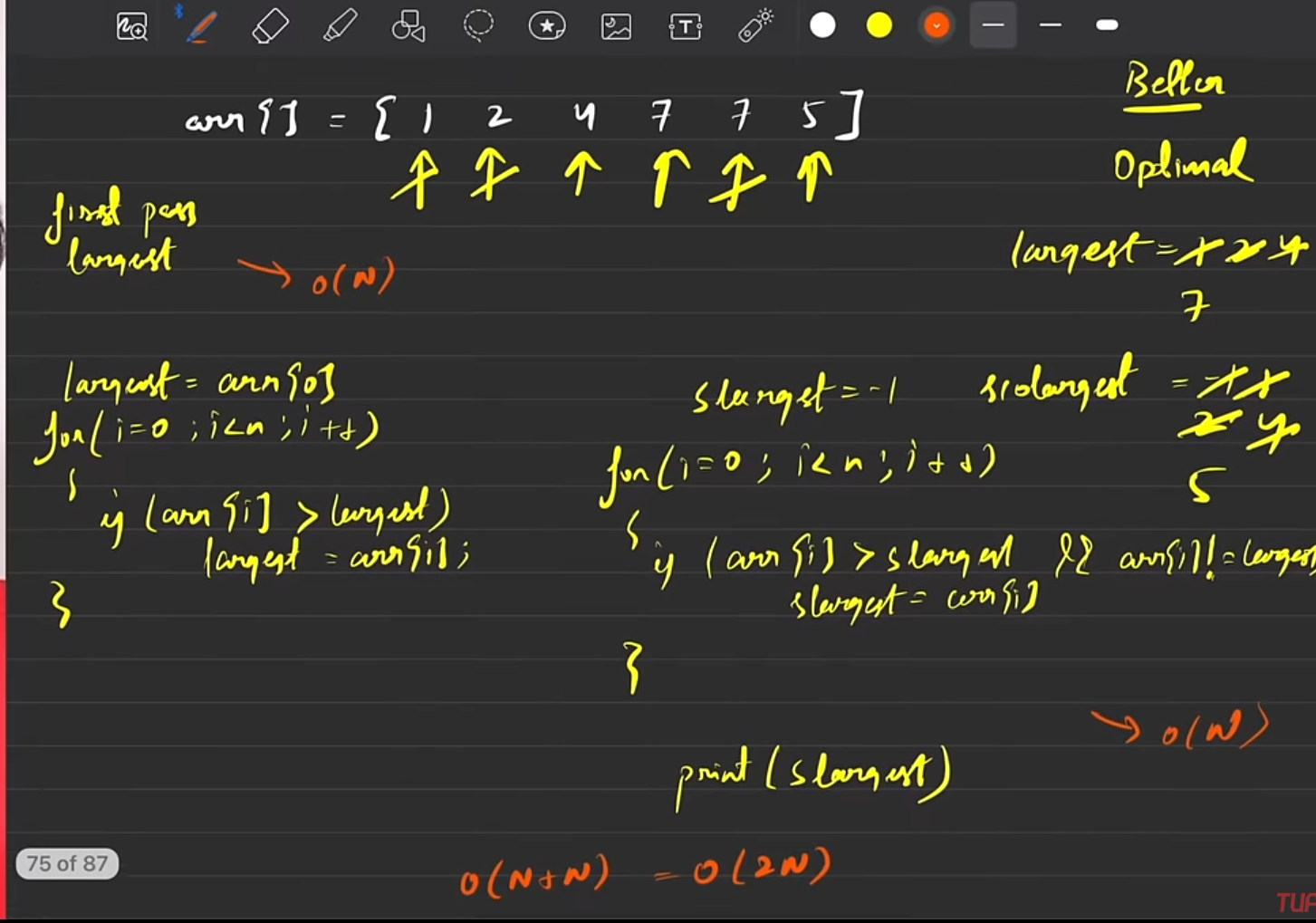
Brute-Force Approach

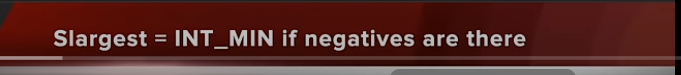
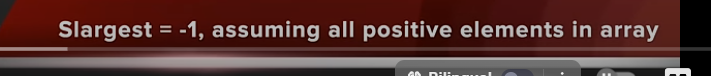
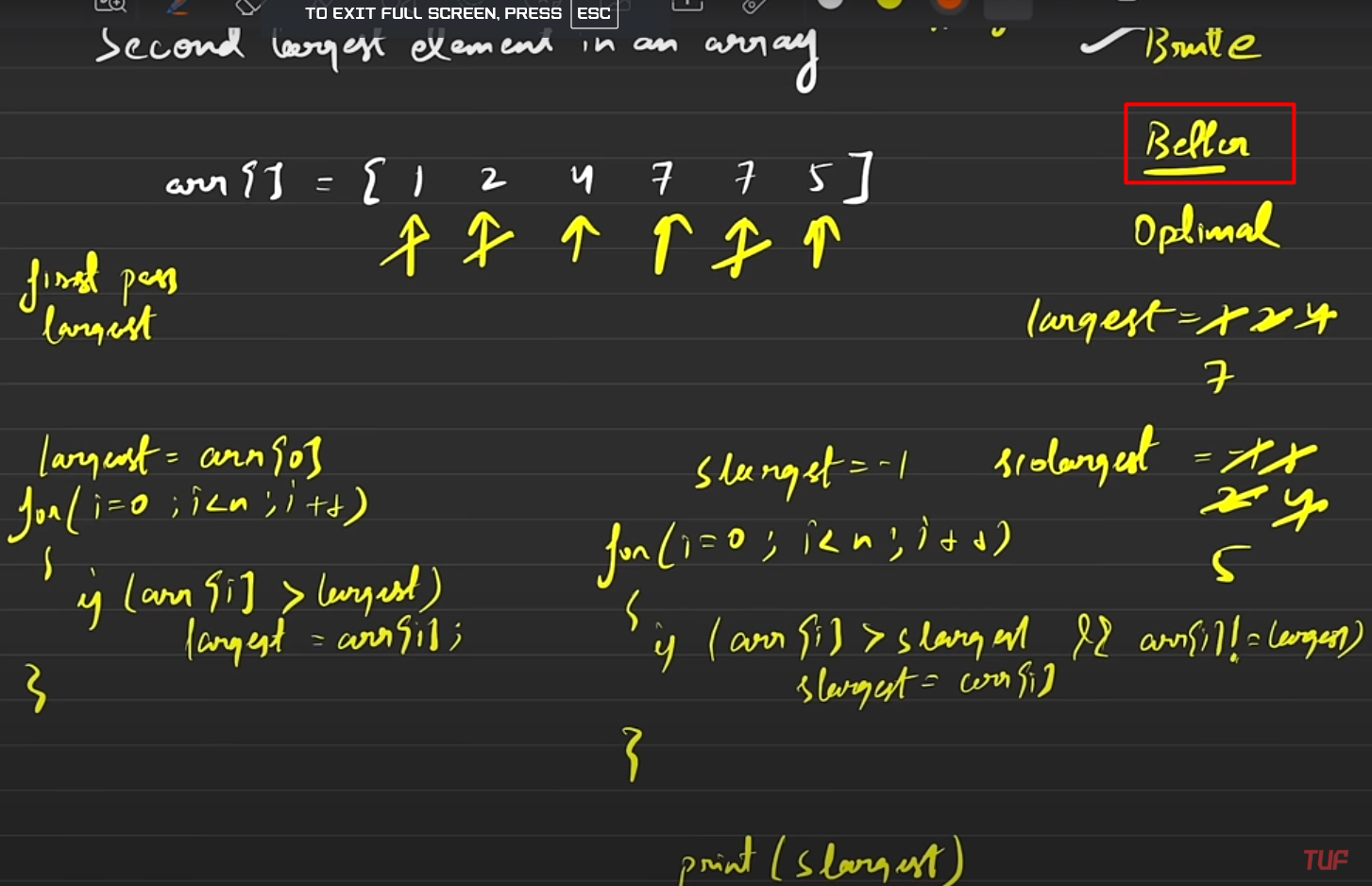


Better-Approach - TC

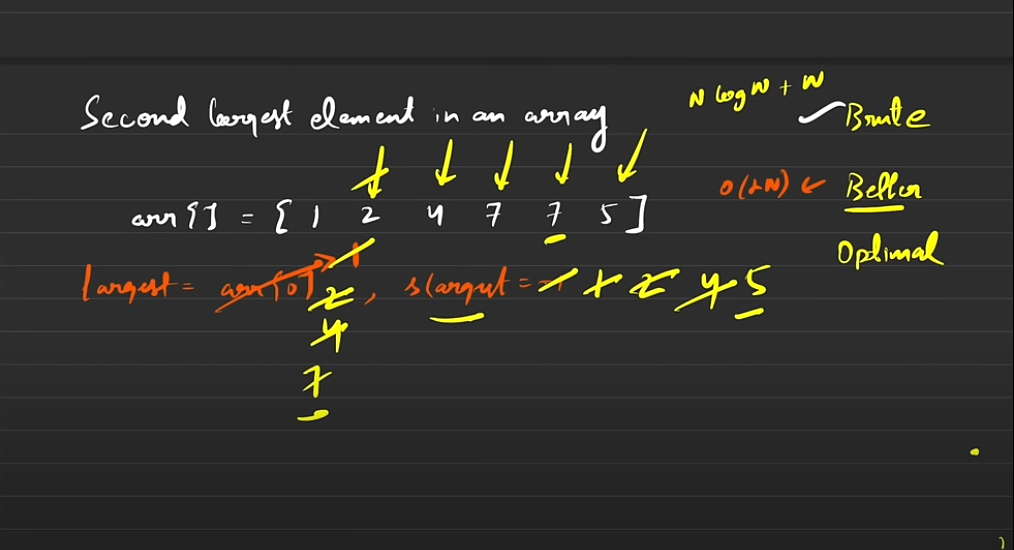
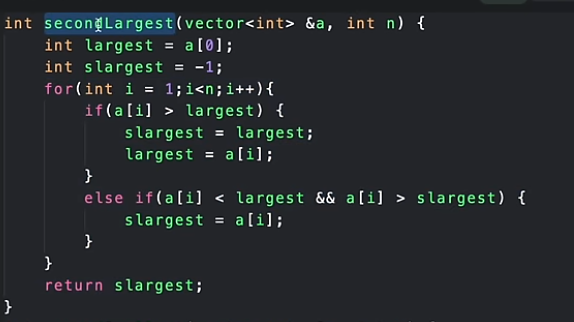
Better-Force Approach

Brute-Force TC





Optimal-Approach



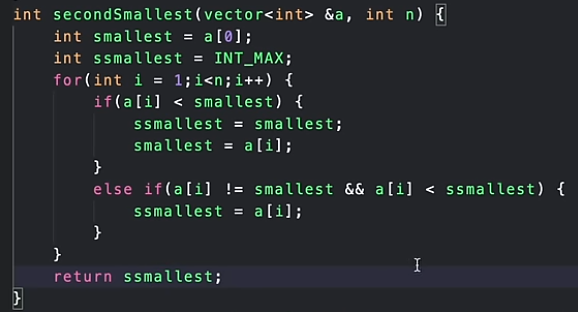
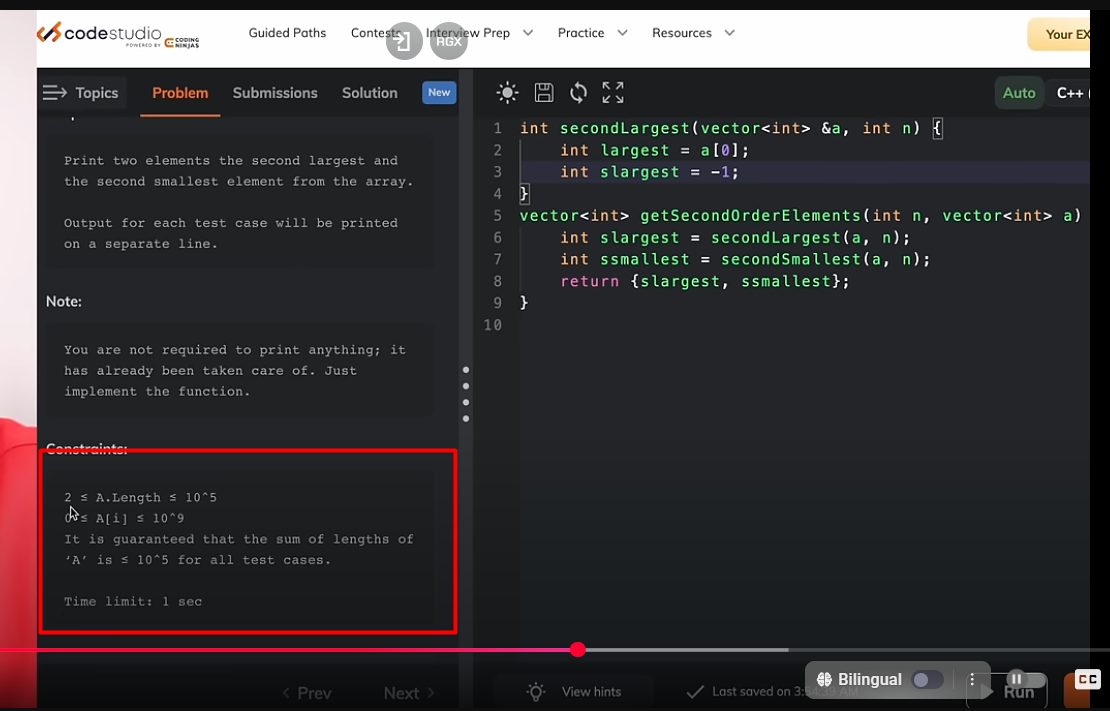
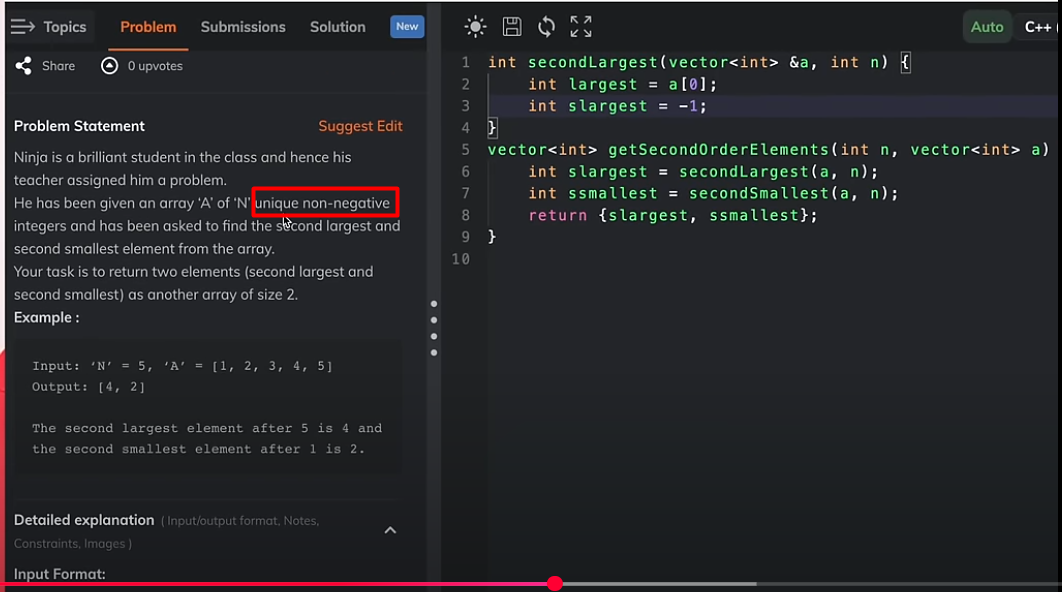
**Main Logic** = “If someone becomes the largest then previous one becomes the second largest.”

Points To Remember:

1. Assume the max as first element and secmax as -1
2. If one element is greater than the max then it becomes the max and the previous max becomes the secmax (Main Logic)
3. If any element is equivalent to max then don’t do anything (Imp)
4. Even if you have found the max it doesn’t mean that the previous secmax is the actual second largest element (there might be cases where we haven’t reached it yet) thus always check if the element that is smaller than the max, is greater than secmax or not. If yes, then make it the new secmax without affecting the max(as max must be already larger).

Optimal-Approach – TC -> O(N) as we only doing a single pass

Striver says that while doing any coding rounds you must do proper and logical naming of variables and functions and don't just give x, y etc. as names, this is to be done in order to stand out in an interview.

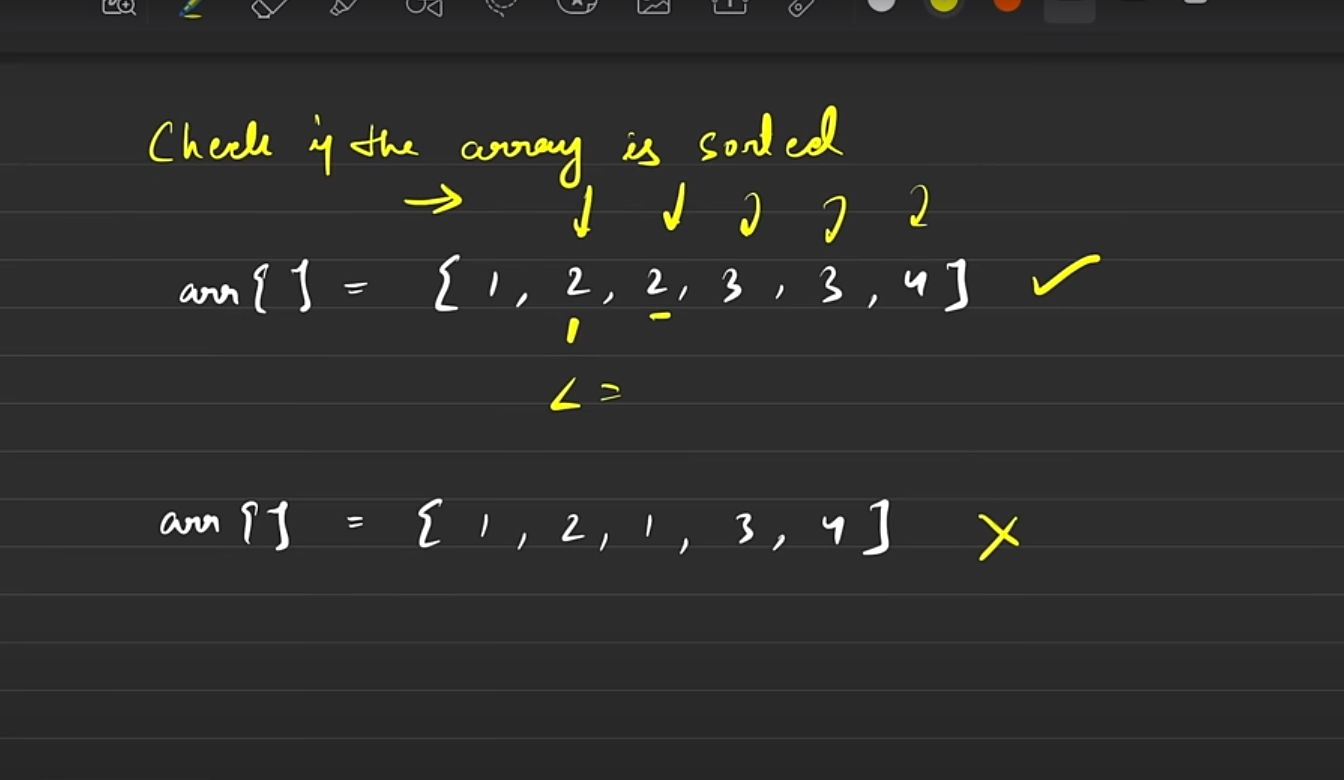


Another thing that, can be noticed is that in question he first went on for what was asked and wrote the code for it first like we were asked for return 2 elements so he firstly went on writing this function of what we have to return and returned a vector

And then he went on to write his own custom functions to figure the outputs that he has to written(here he hasn’t started writing the upper funcs yet)

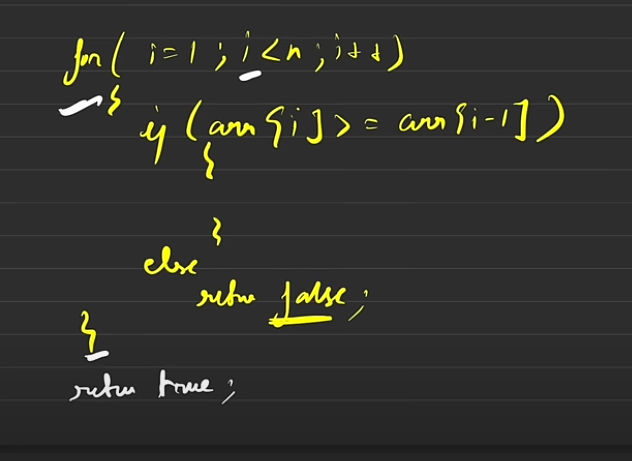
Now, although this irrespective of the question, but I wanted to drive your attention to it, while doing the question on Code-forces, I want you to look at the way he had approached the question and how he thinks. By the highlighted lines he conclude that since array has to be of minimum size 2 and all the elements are unique and non-negative. He figured out that this means that all the testcases will not have duplicates in this question will have a second element for sure. Although he still wrote for -1 but it is good to know how he thought intuitively

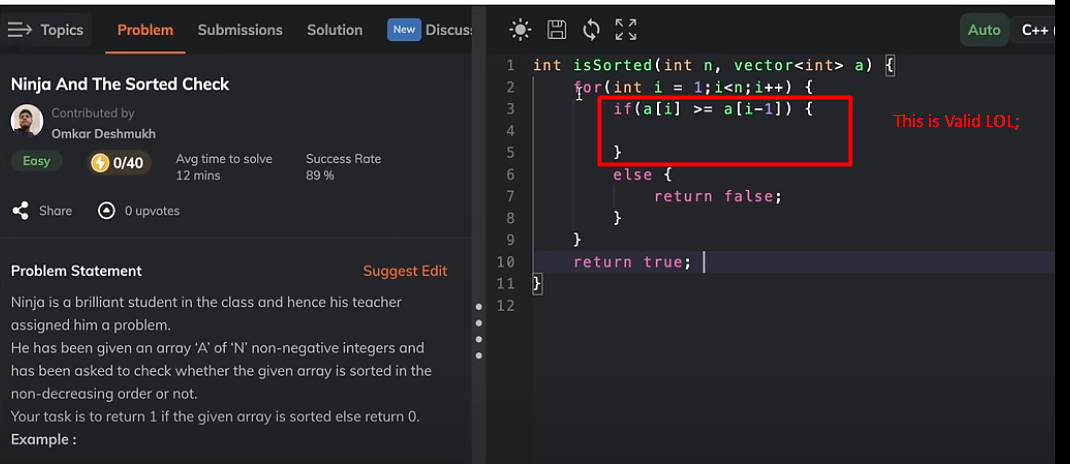
Optimal-Approach (For Second Smallest) (Same TC and SC)

Q-3

We don’t need to categorize the solutions as brute, better or optimal as this is a very straightforward question and thus, we just need to follow one main logic. (Remember, we have to check the sorting in non-descending order (that’s why the first array given here is sorted but not the other one).

**Main logic: ->Traverse through the array and check if the previous element is <= next element**





Q-4

This, is a popular question that is asked in interviews (i.e. find and remove duplicates in array or find the unique elements in the array)

Now, we have to keep somethings in mind:

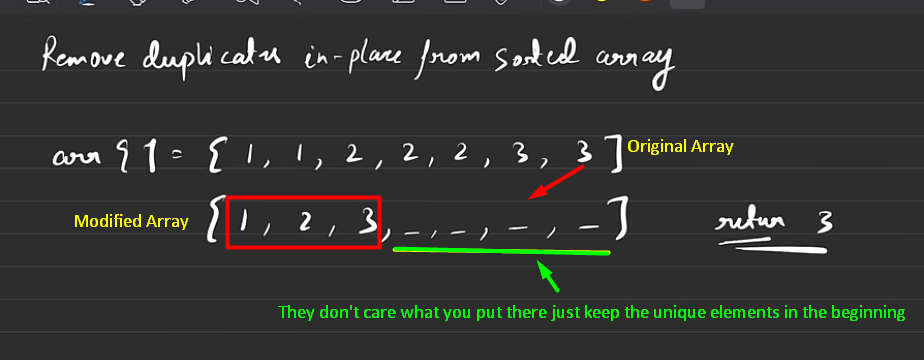
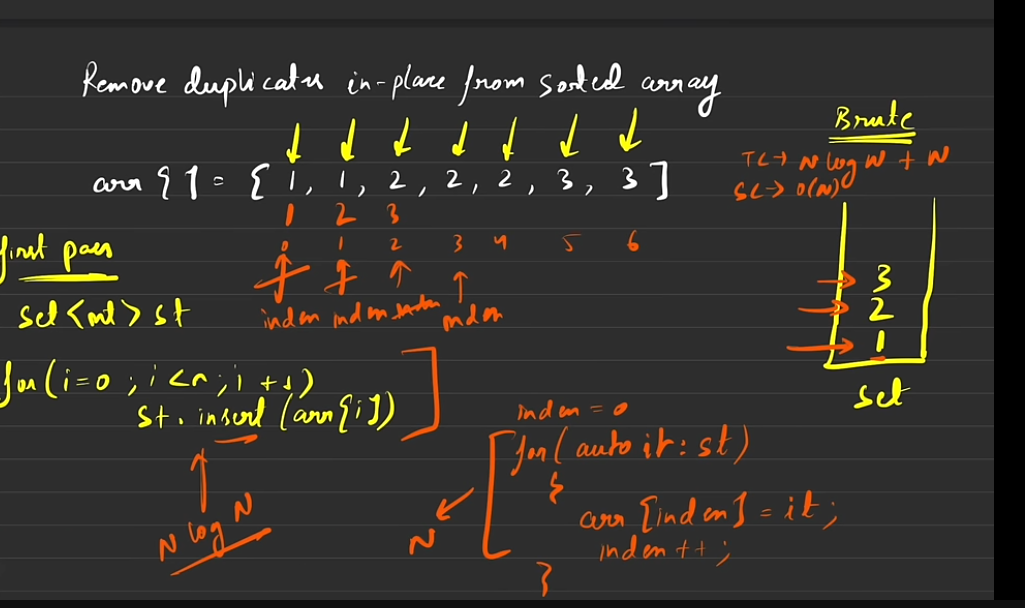
->The array given is sorted (usually in ascending order)

->You have to modify the same array (that is what is meant by ‘in-place’ here)

-> The unique (how many they may be) should be there from starting indexes of the array and you can fill whatever in the last indexes (they don’t care)

-> You are required to tell the number of unique elements present in the array as well (usually done through the index of the last element after modifying)

->So, basically modify and tell the number of unique elements what the question is asking you to do

1

Brute-Force Approach

Now, for the Brute-Force solution:

Whenever, the word ‘unique’ comes in minds, we know that STL we require is none other than Sets (as it can only contain unique elements and rejects the duplicate ones).

So, we will insert the elements in a set in our first pass, then after the unique elements are filtered, then we’ll make a second pass where elements from the set are refilled in same array starting from index 0. And the size of the unique array(i.e. the no. of unique elements) will be none other than (last unique element’s index +1)

**TC->** N log(N) for the first pass because insert operation takes log(N) time in set and since we are doing it for n elements (as the loop traverses over N elements) so it becomes N log(N). Now, for the second part since the loop traverses again N times so its TC is O(n) and for both the passes the overall TC becomes O(N log(N) + N).

SC-> Now, we are taking an extra set to complete this solution, which in worst case can have N elements (as all elements can be unique (without duplicates) in the worst case). Thus, SC becomes O(N).

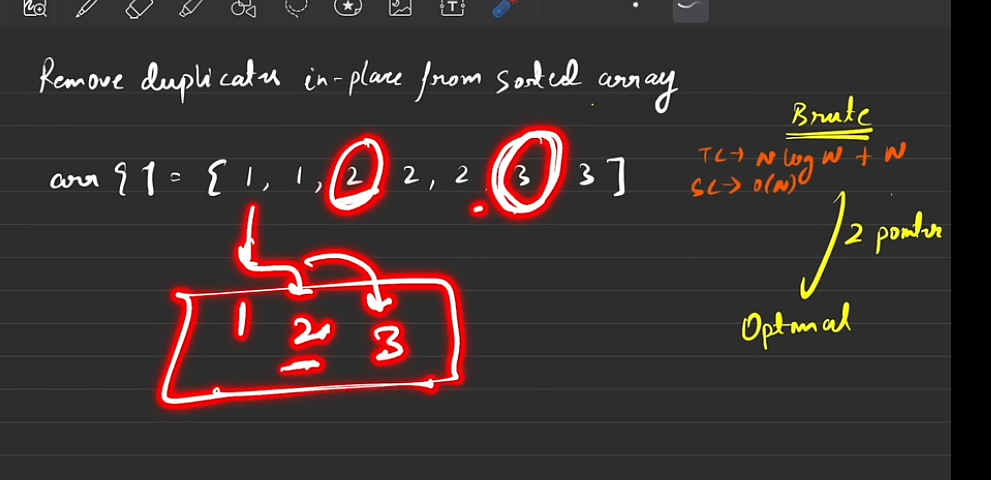
Remember, that whenever this question comes up to you, the you don’t have to say the optimal solution directly but instead drive the interviewer. Start from the brute-force approach and then move on to the optimal solution

Optimal-Approach

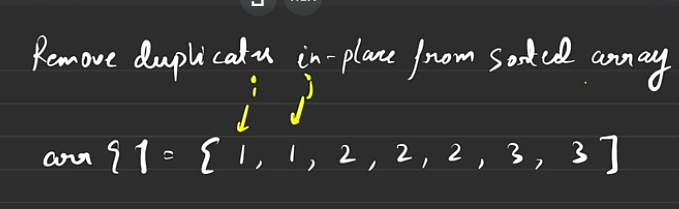
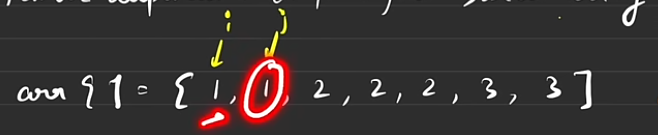
Now, in order to optimize the above brute-force solution we’ll use something known as Two-pointer Approach.

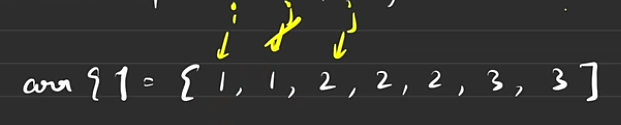
Now, here we will traverse the same array, follow the following logic:

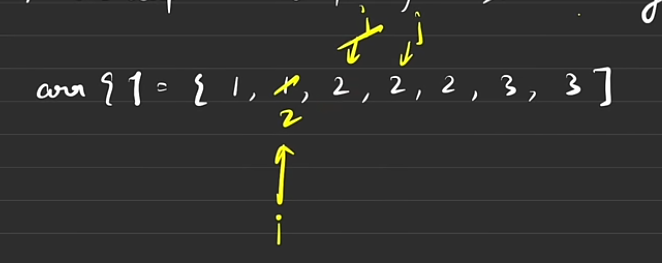
“Since the array is sorted, so every first unique element will be at its unique position, now we’ll start from 1 unique element and place it on the starting index 0, and then traverse ahead and ahead and check for the next element that is not equivalent to the my previous unique element and once found, I will place it at the next index (of the previous unique element) and will move with the same logic for all elements to find the unique ones”. Thus, main logic is that the next position of the previous unique element must be filled someone who is not equivalent to it (and it would only be the next equivalent element) (beautiful logic out there).

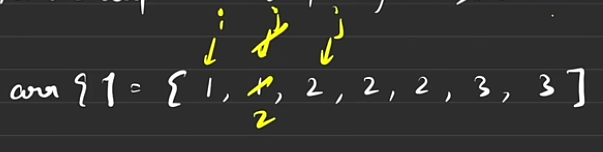
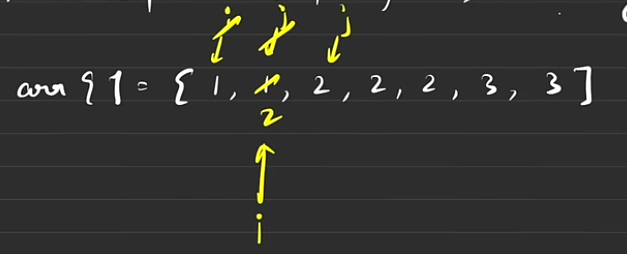


Thought Process

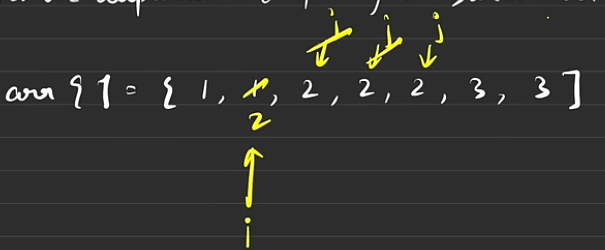


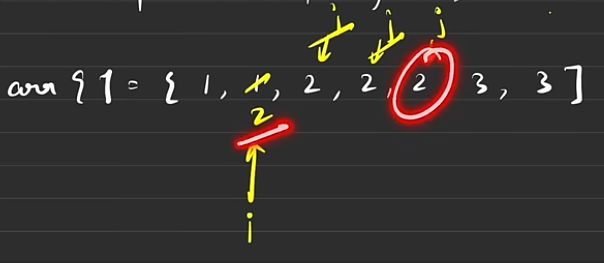


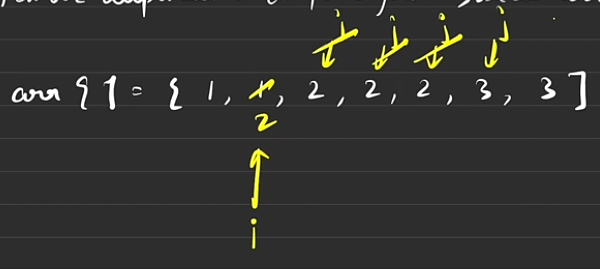
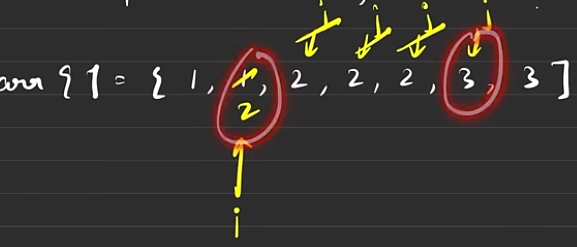
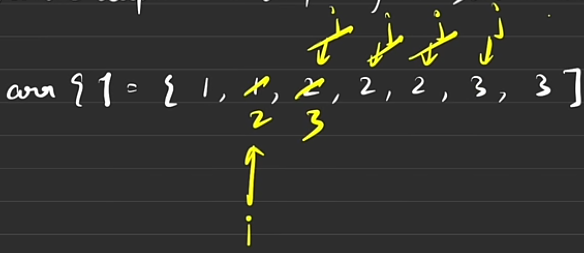


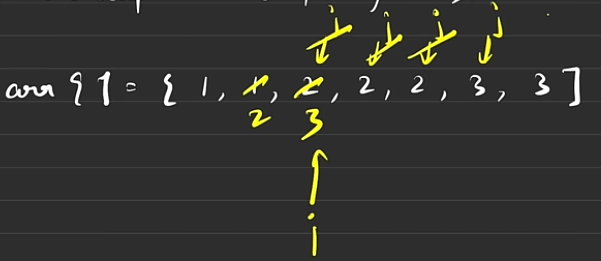


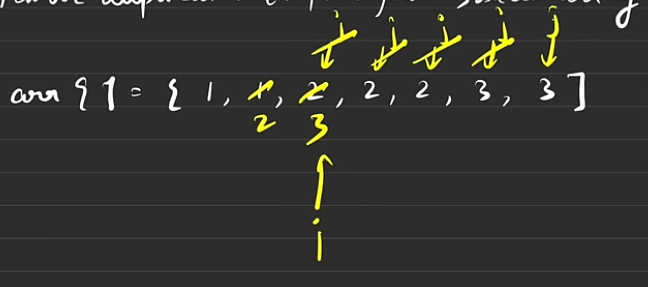
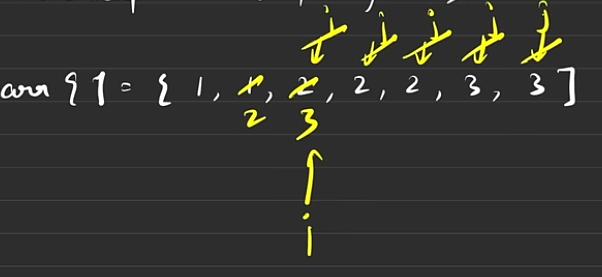


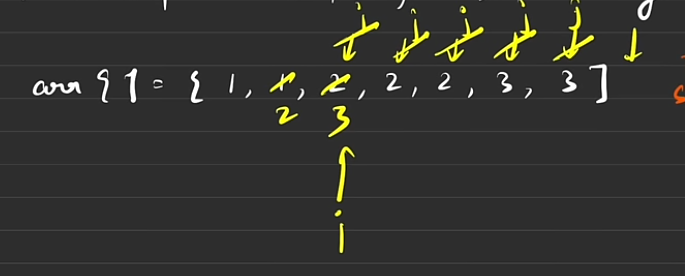


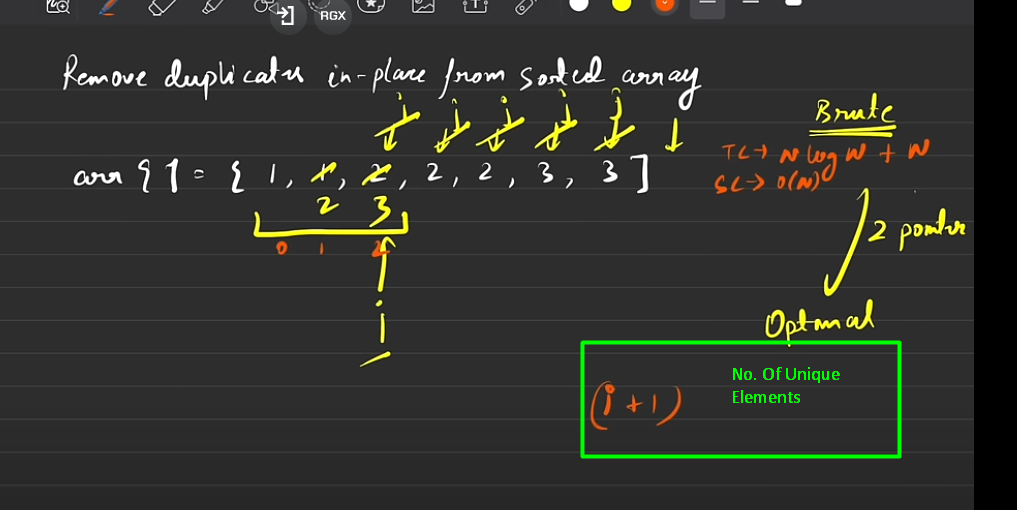








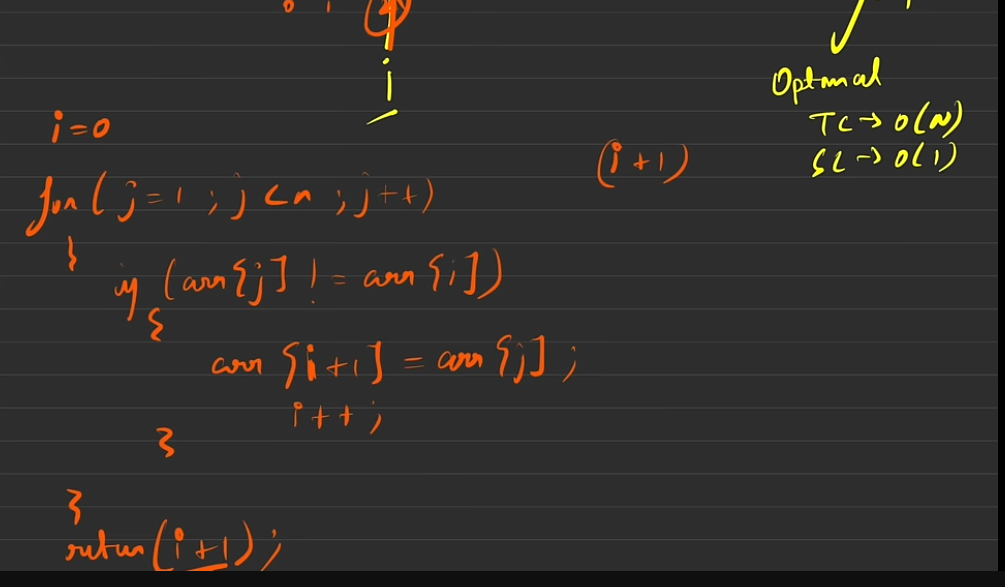




Follow this Horizontal First and Vertical Approach to look at this flowchart. Now, here the i pointer basically keeps track of the unique elements and only moves when a non-equivalent to previous element (i.e. the new unique element) is found and placed next to the previous unique element (and then i takes the index of that new unique element).

Now, j pointer basically traverses the whole array, it moves to the next element if that element is equivalent to the previous unique element. If a new unique element is found then it triggers i pointer (which as I explained earlier, the new unique element is copied to the index next to the previous element and it moves to the index of the new element).

So its like saying, i pointer is fixed at the first element (by default) then the j pointer goes on and asks every element if its equivalent to the element pointed by the i pointer (i.e. previous unique element), the if the element says yes then j pointer moves to the next element while the i pointer stays still at the same element. But if the elements says no (meaning it’s the new unique element), then i pointer says to the element “take my front position” meaning that the element (pointed by the current j pointer) to move to the next index position to i pointer i.e. (i+1). Once the new unique element moves next to the previous unique element the i pointer updates and shifts from the previous unique element to the new unique element and j pointer also shifts to the new element (since the current element has been moved/copied) and asks the question again. This cycle continues until j pointer reaches out of the array.



TC-> Here, it is O(N) since j pointer traverses the whole array

SC-> here , it is O(1) as we are modifying the same array and no extra space was used (Thus, in true sense it is the actual in-place solution as everything is done within the same array itself and this is one of the reasons which makes it optimal.