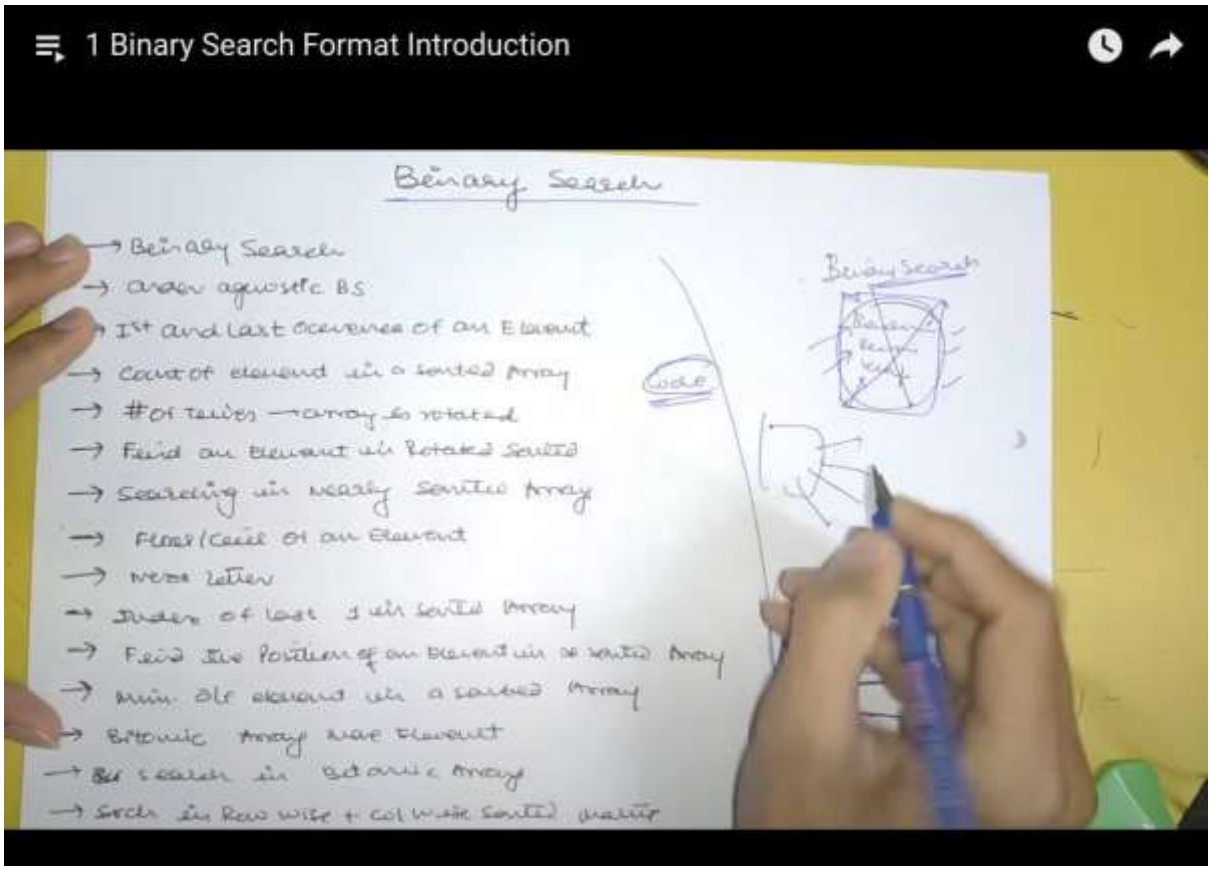
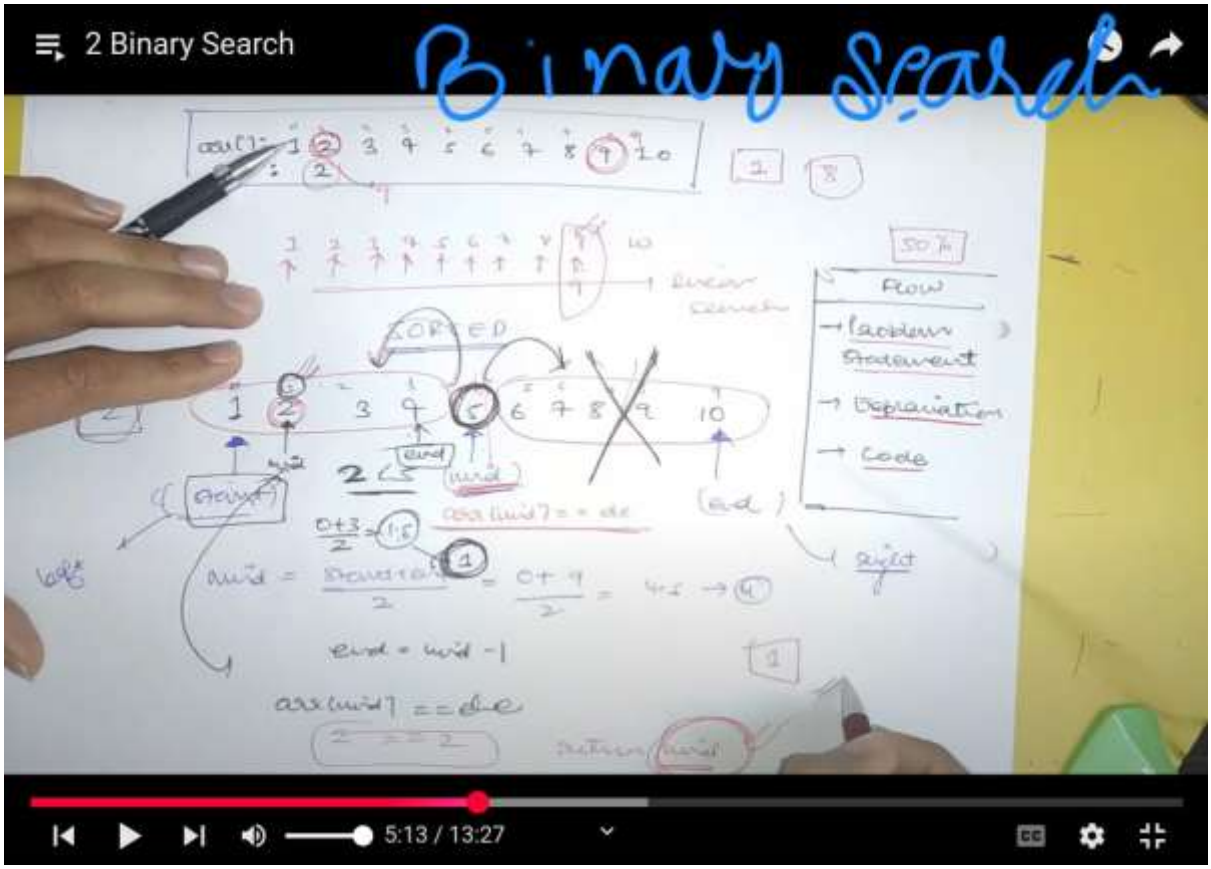
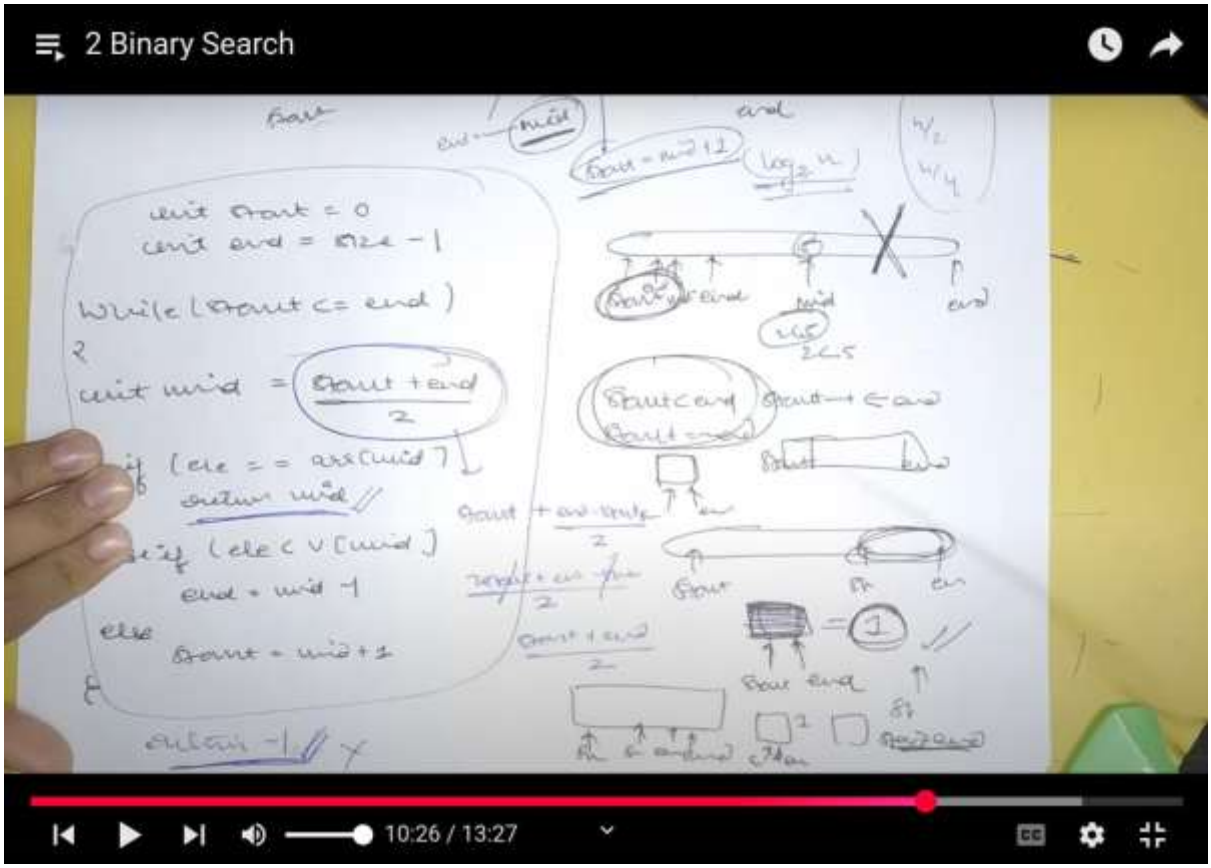
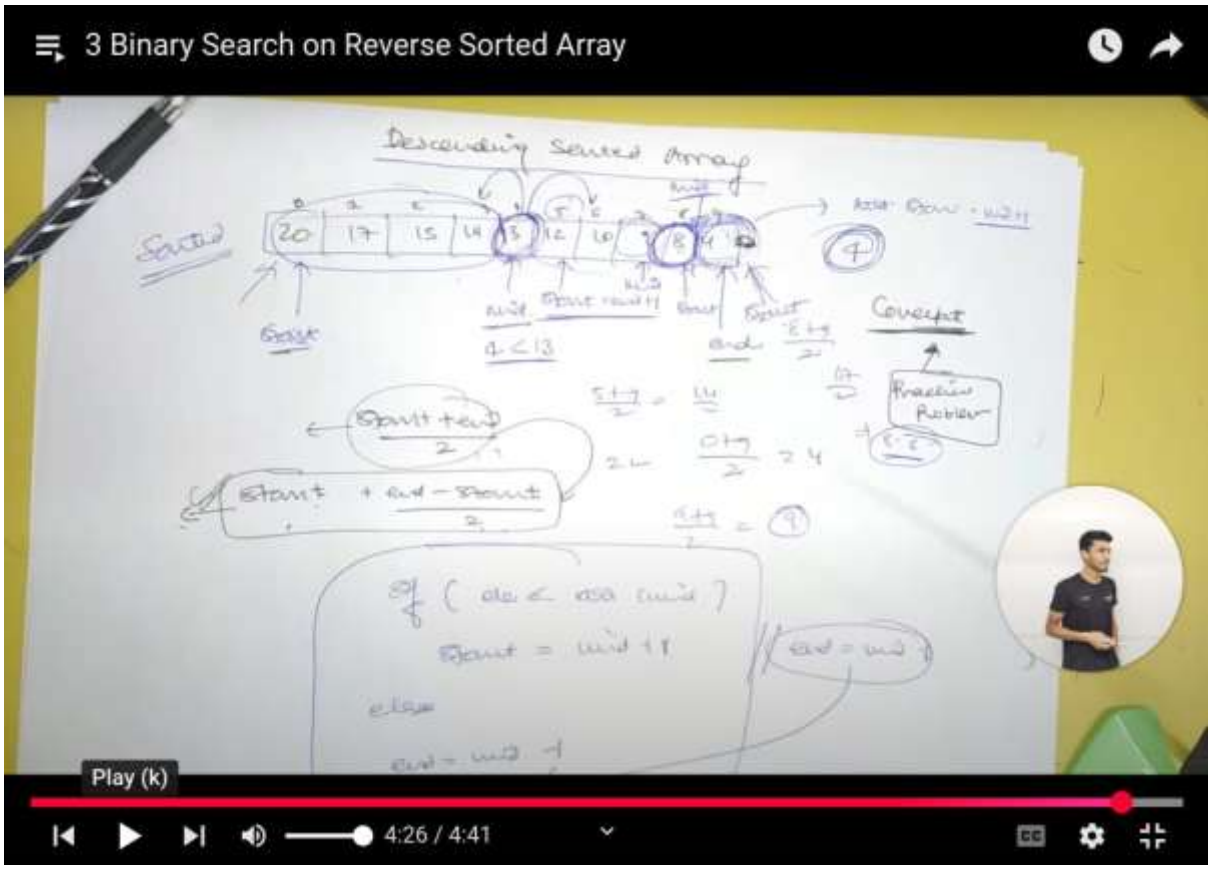
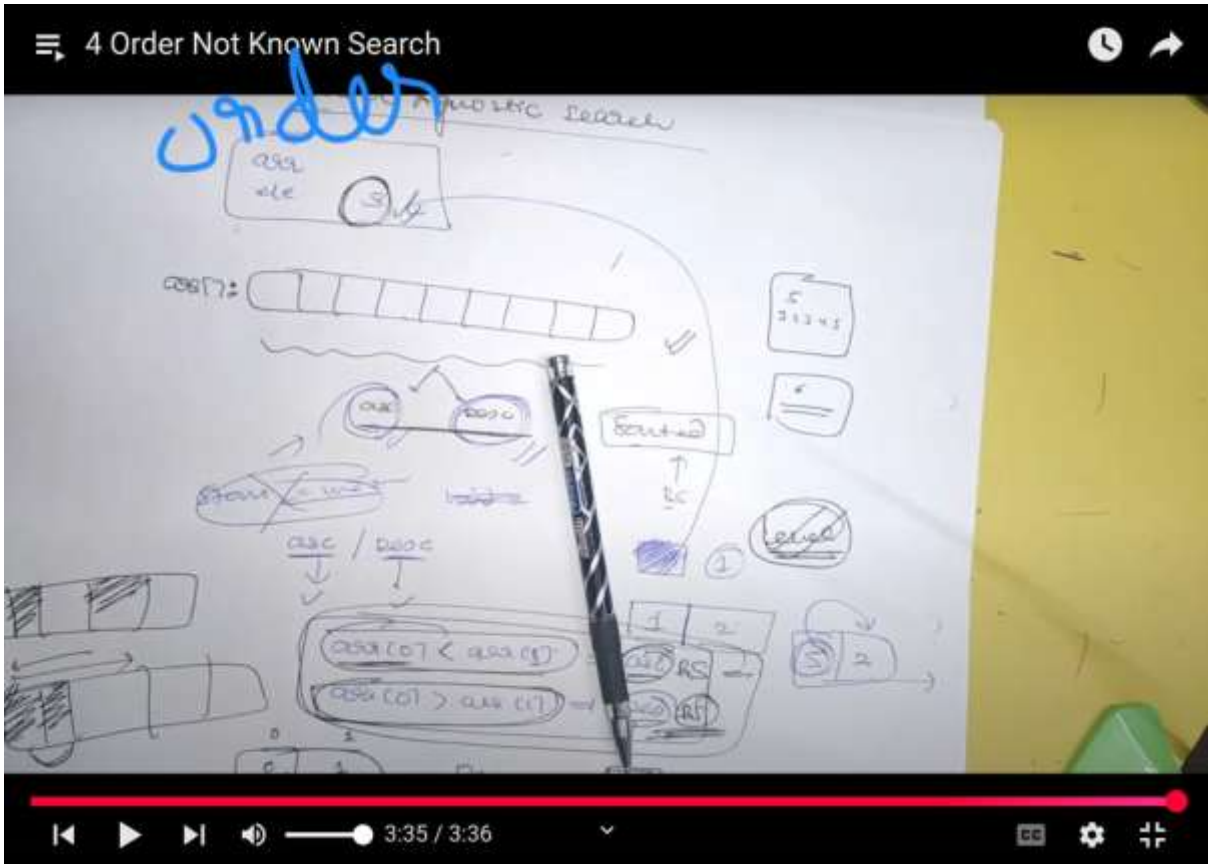
Binary Search

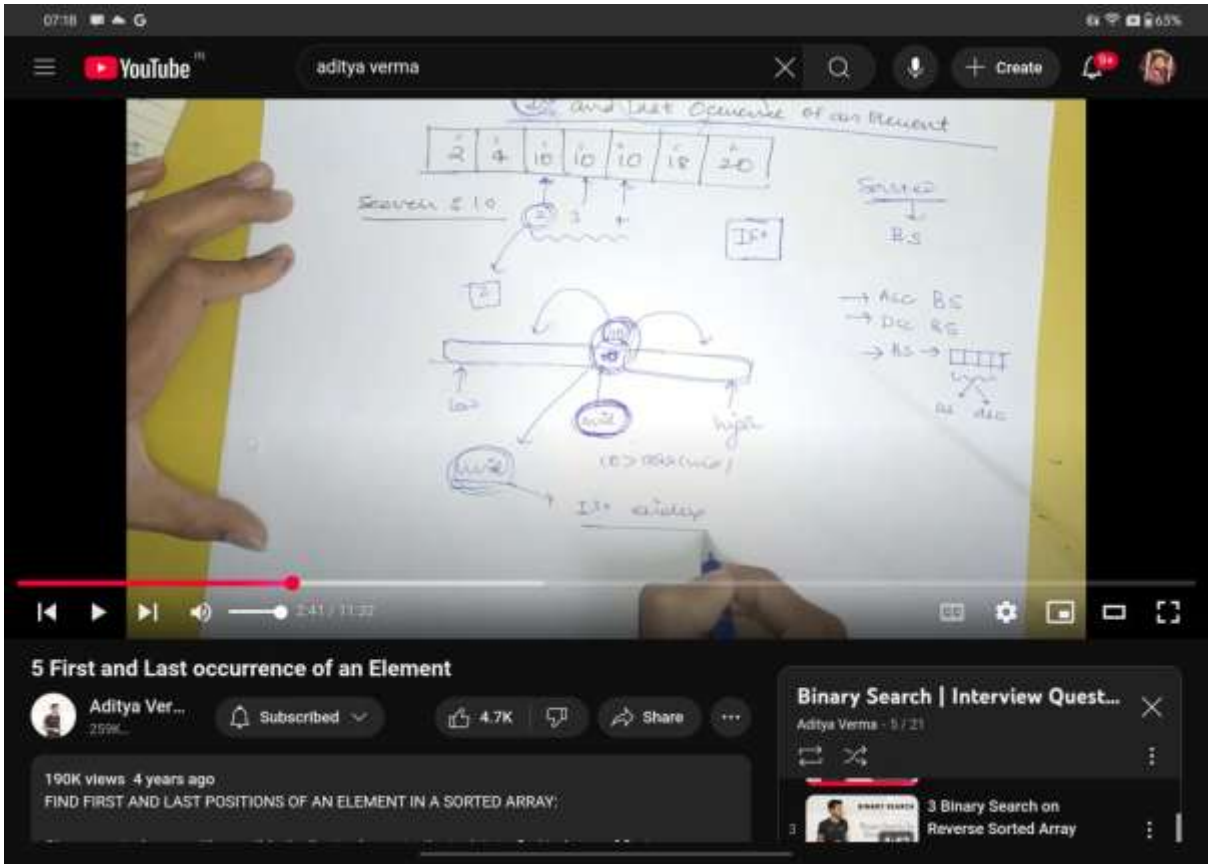


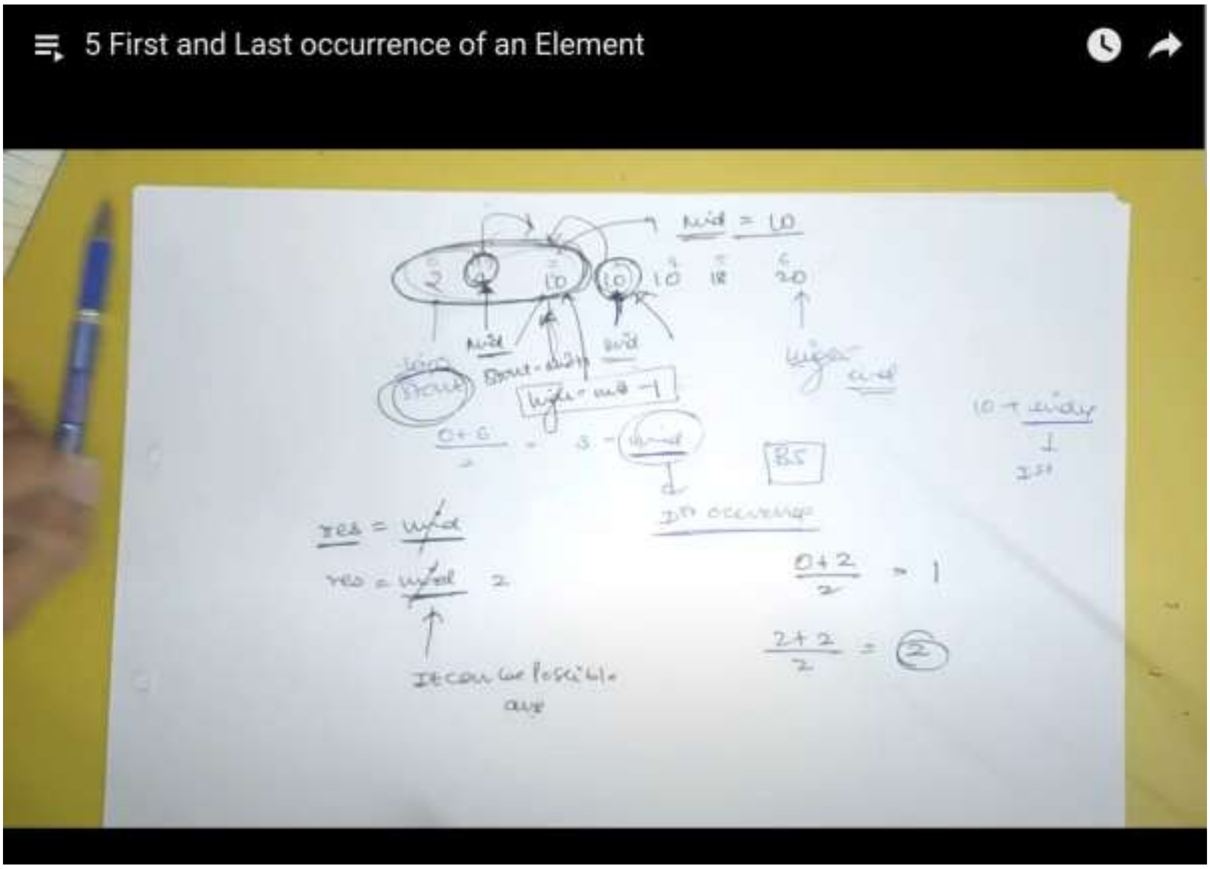


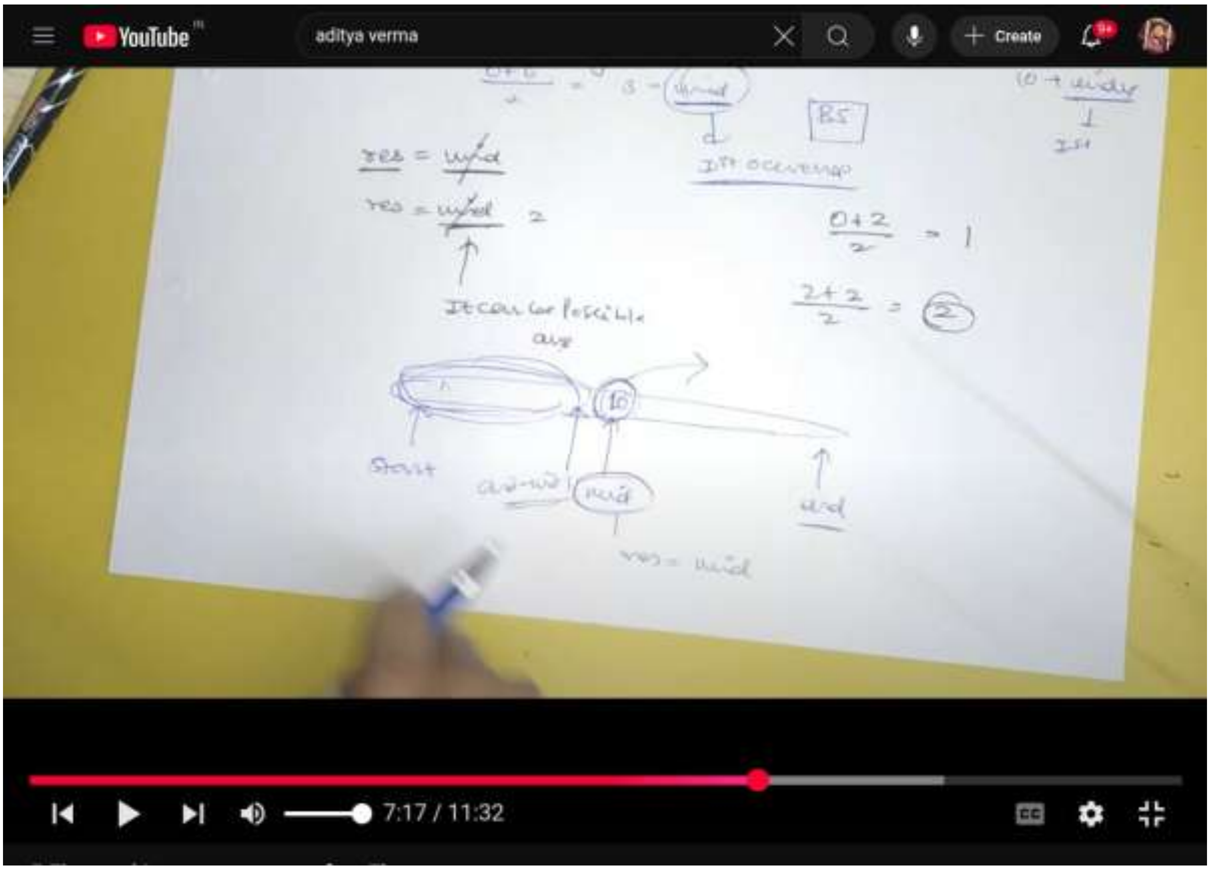

2 Binary Search
int start = 0
int end = size - 1
while (start <= end)
int mid = (start + end) / 2
if (ele == arr[mid])
return mid
else if (ele < v[mid])
end = mid - 1
else
start = mid + 1
return -1
10:26 / 13:27

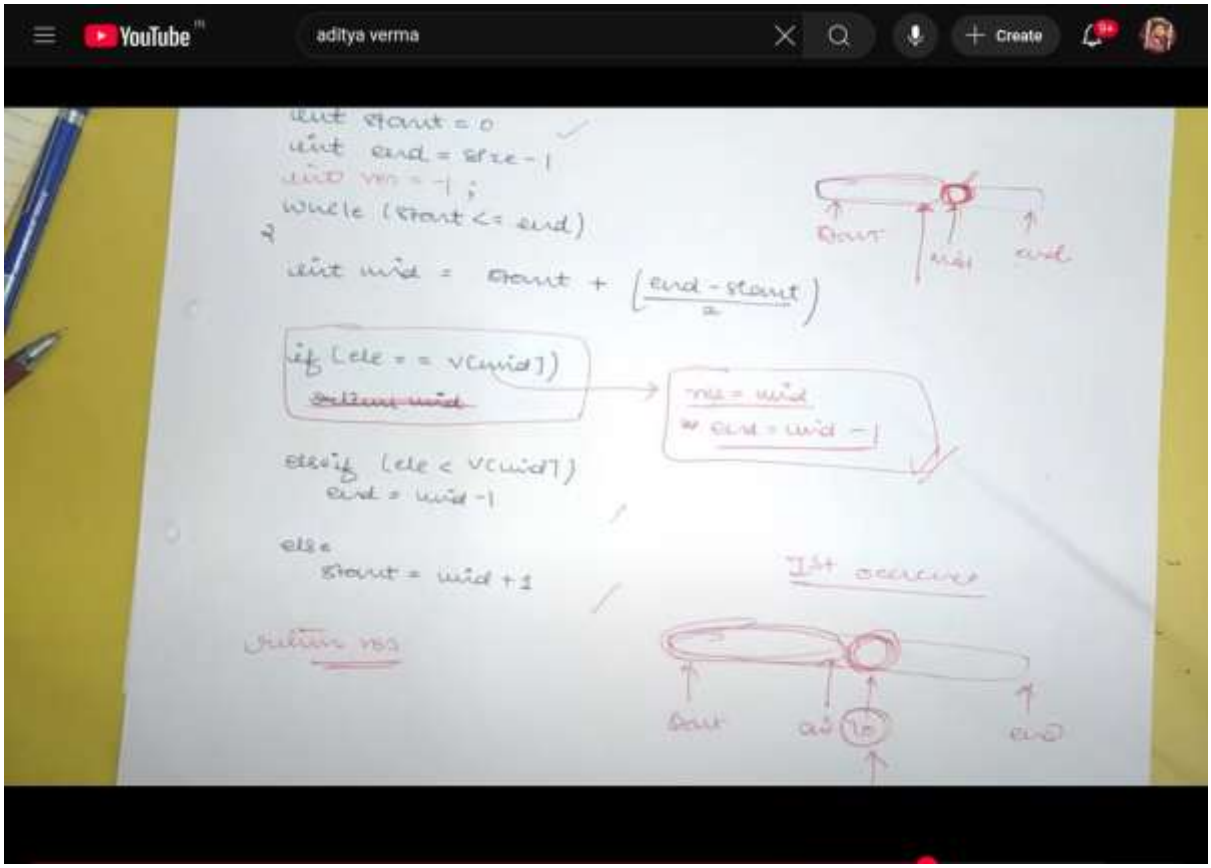

3 Binary Search on Reverse Sorted Array
Descending Sorted Array
Sorted
20
17
15
14
13
12
10
8
4
Coverage
Play (k)
4:26 / 4:41

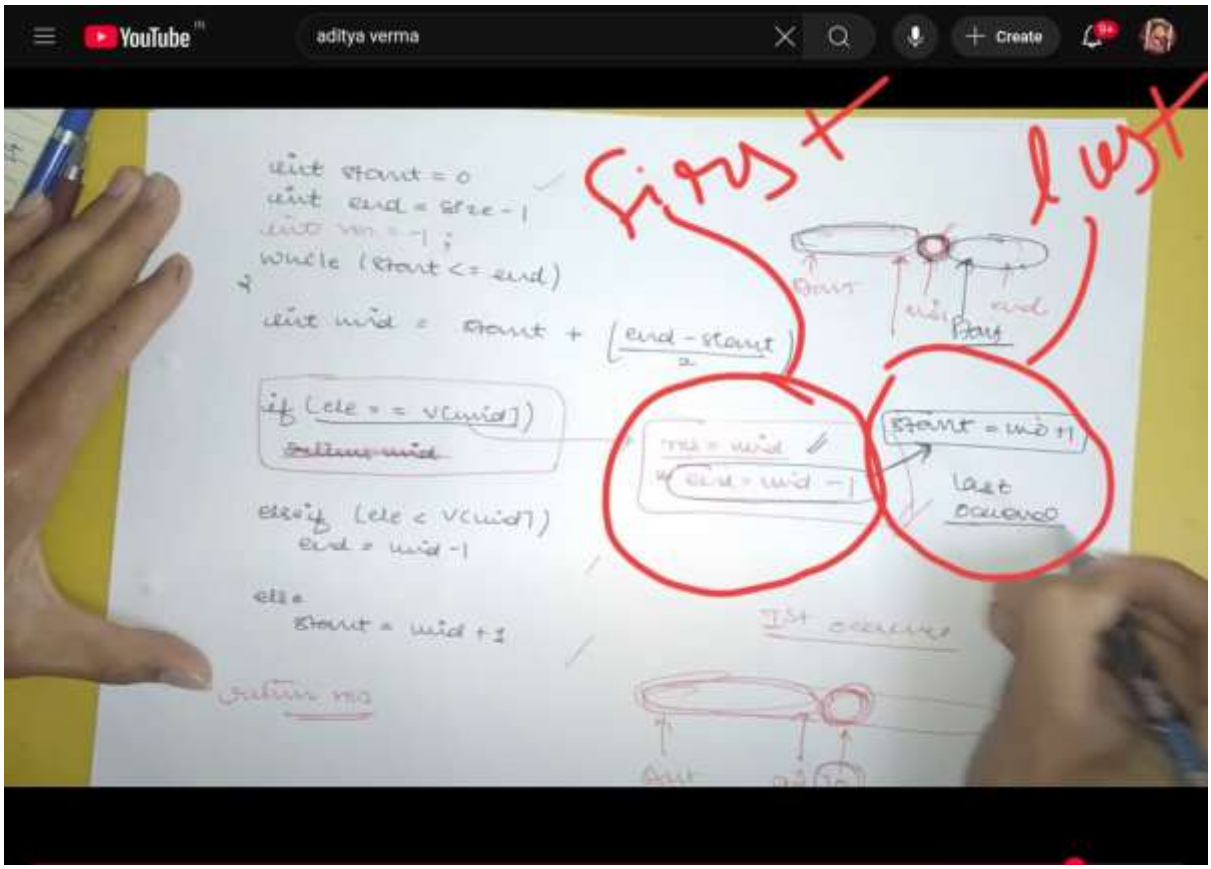

4 Order Not Known Search
3:35 / 3:36

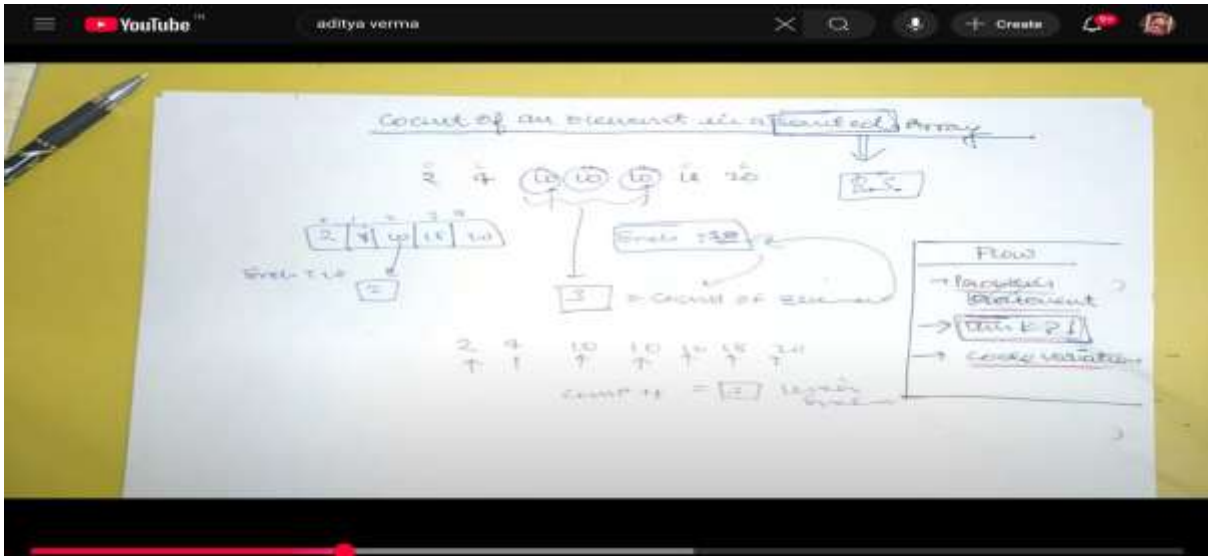

YouTube
aditya verma
Create
5 First and Last occurrence of an Element
Aditya Ver...
Subscribed
4.7K
Share
190K views 4 years ago
FIND FIRST AND LAST POSITIONS OF AN ELEMENT IN A SORTED ARRAY:
Binary Search | Interview Quest...
Aditya Verma
3 Binary Search on Reverse Sorted Array

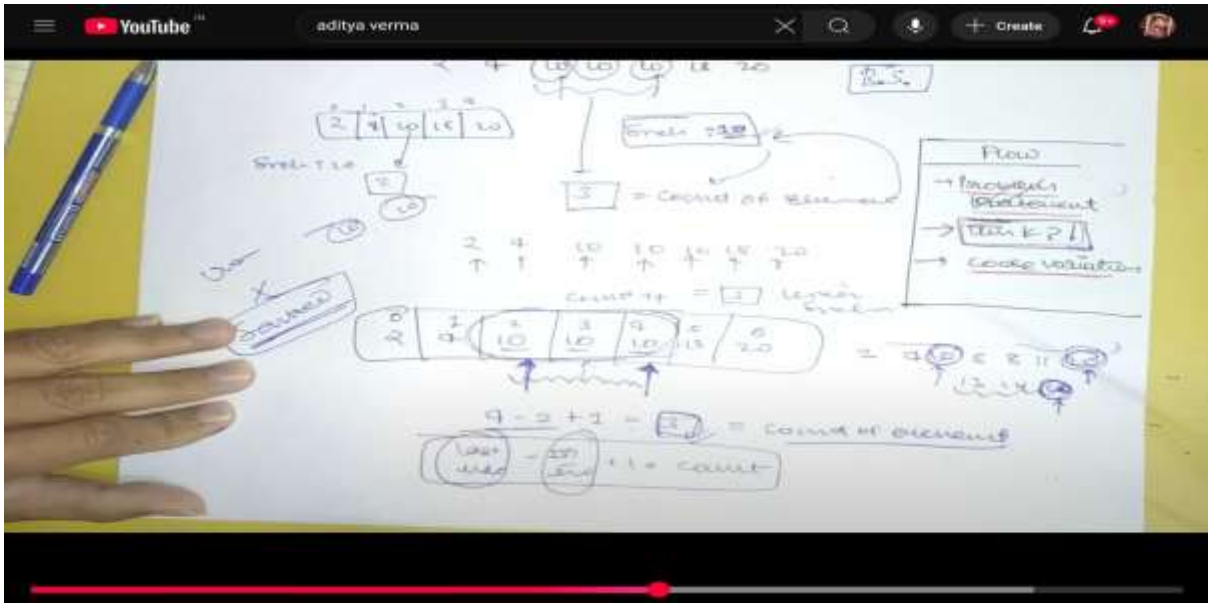

5 First and Last occurrence of an Element
mid = 10
BS
0+2
2+2
res = mid

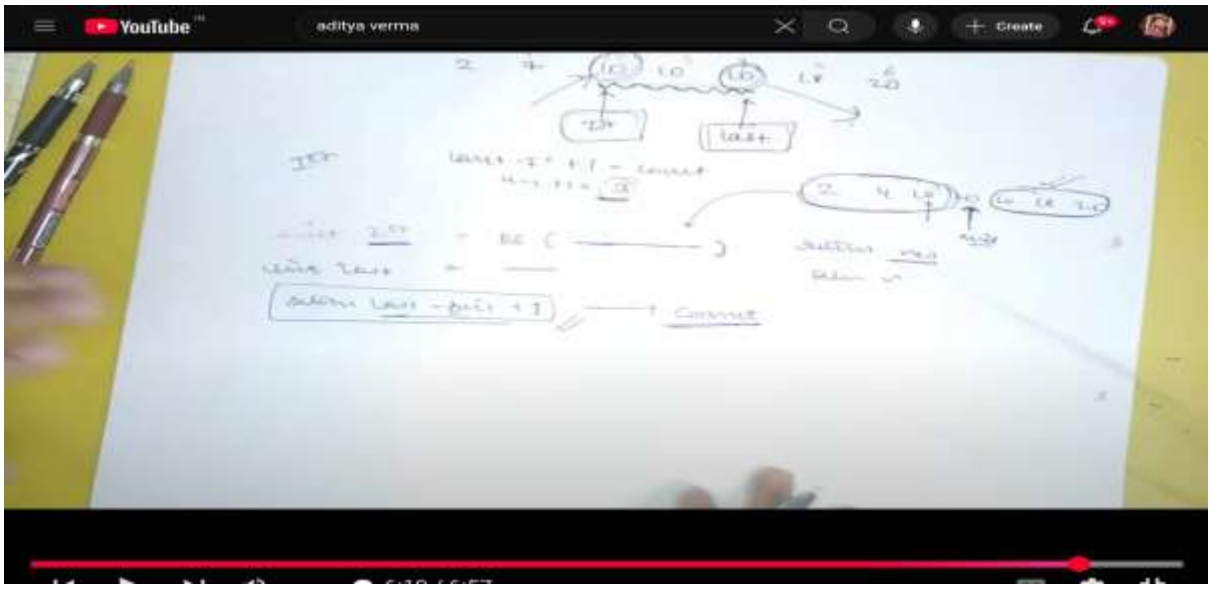

YouTube
aditya verma
Create
BS
res = mid
0+2
2+2
start
mid
7:17 / 11:32

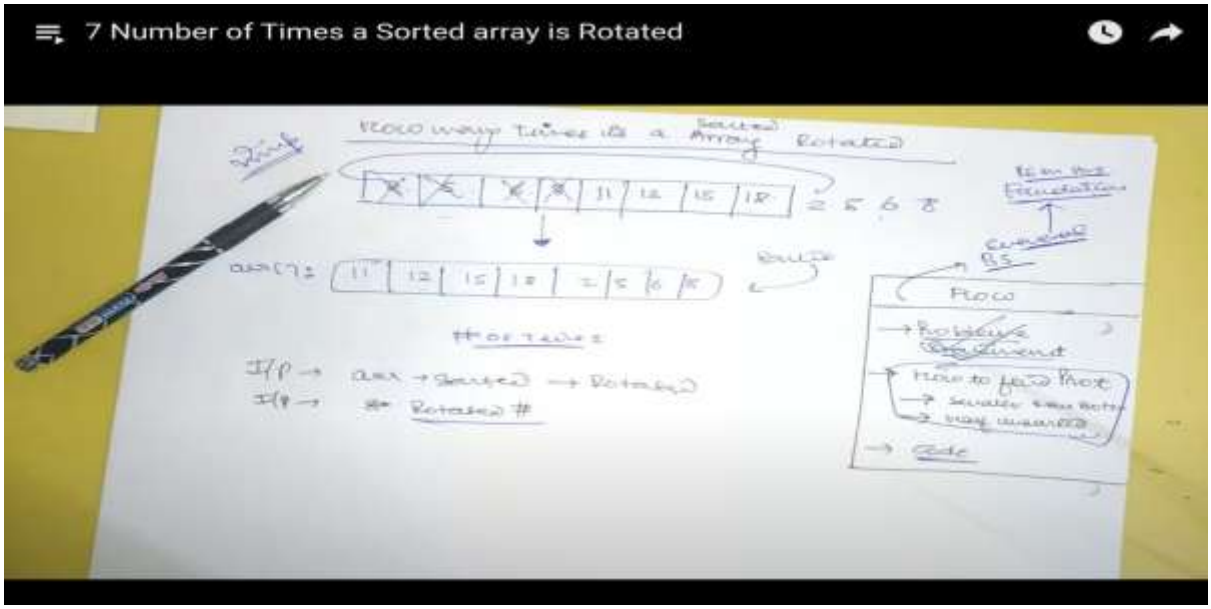

YouTube
aditya verma
Create
int start = 0
int end = size - 1
int ans = -1 ;
while (start <= end)
int mid = start + (end - start)/2
if (ele == v[mid])
ans = mid
end = mid - 1
else if (ele < v[mid])
end = mid - 1
else
start = mid + 1
1st occurrence
start
mid
end

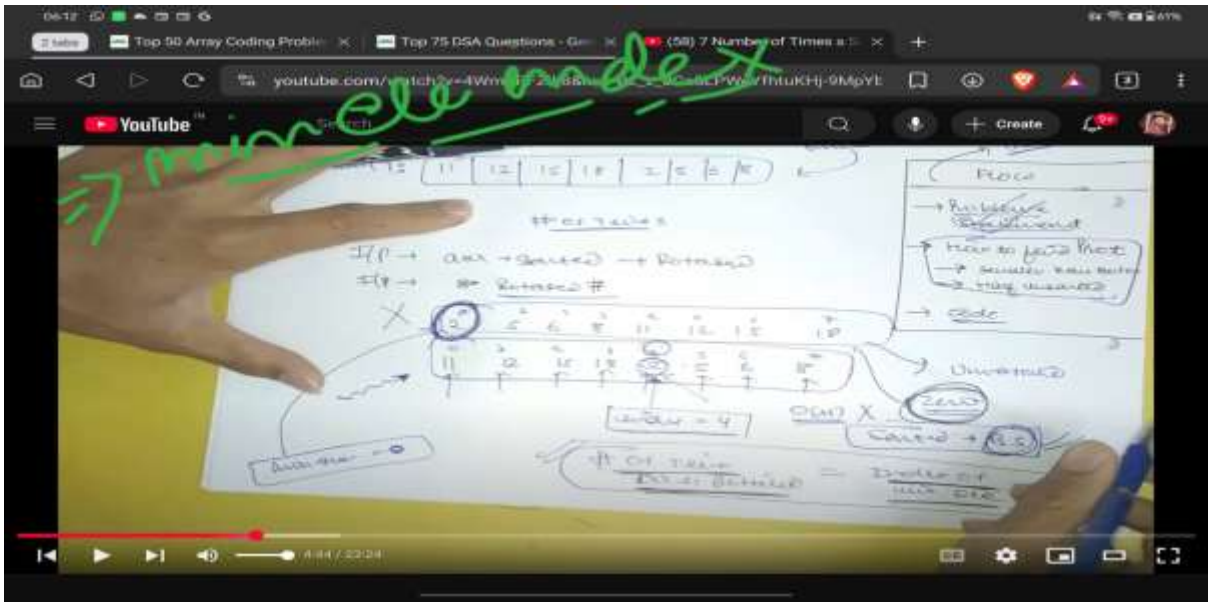

YouTube
aditya verma
Create
int start = 0
int end = size - 1
int ans = -1 ;
while (start <= end)
int mid = start + (end - start)/2
if (ele == v[mid])
ans = mid
end = mid - 1
else if (ele < v[mid])
end = mid - 1
else
start = mid + 1
1st occurrence
last occurrence
first
last
start
mid
end

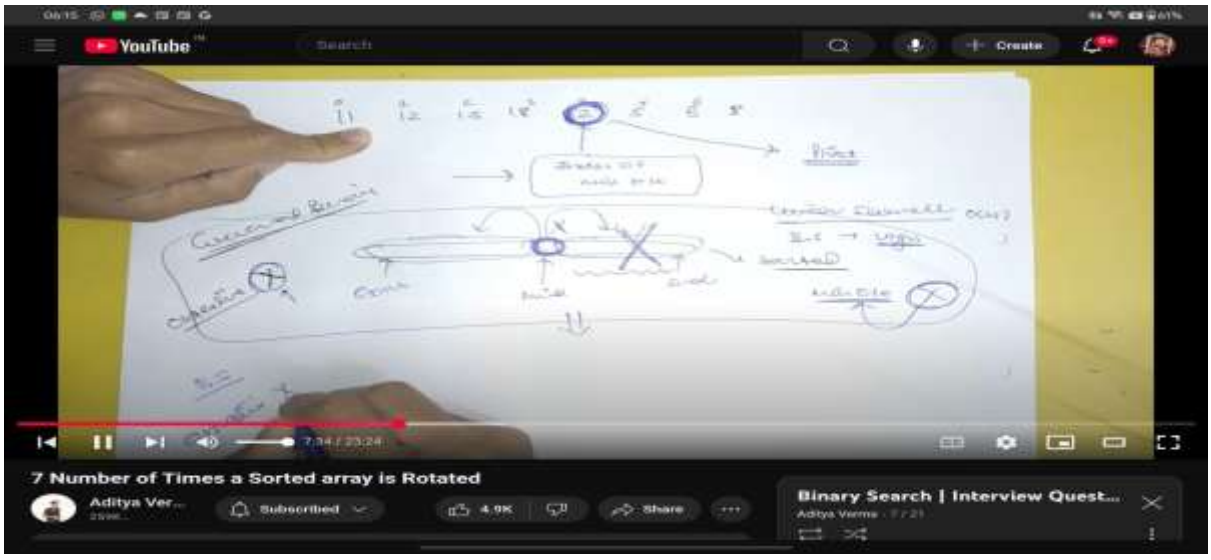

YouTube
aditya verma
Create
Flow

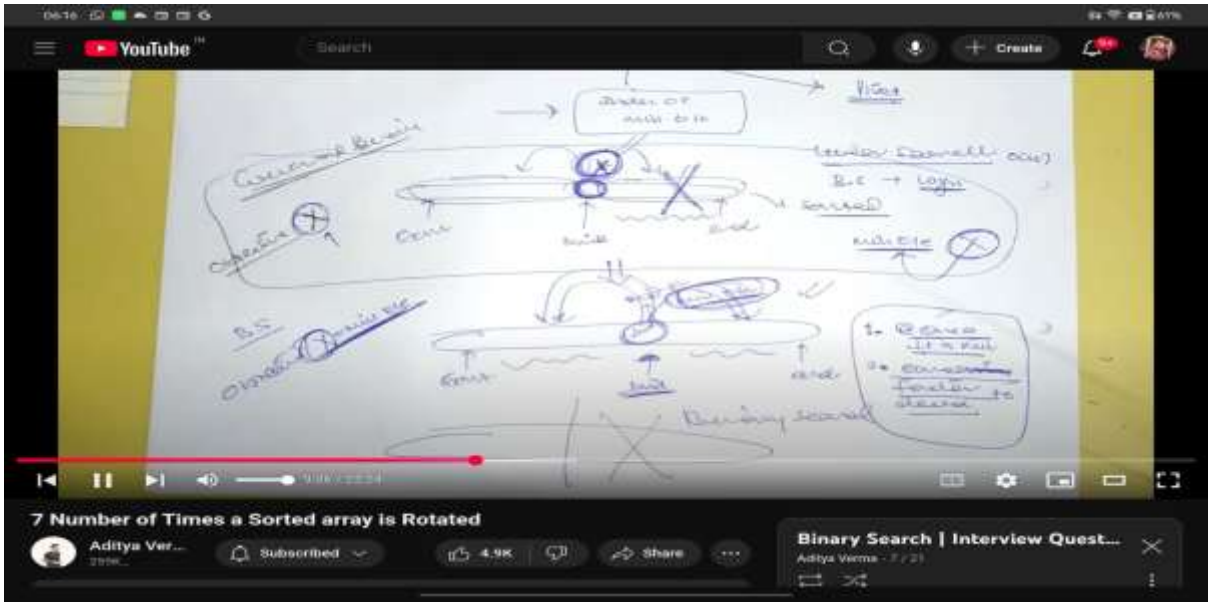

YouTube
aditya verma
Create
Flow

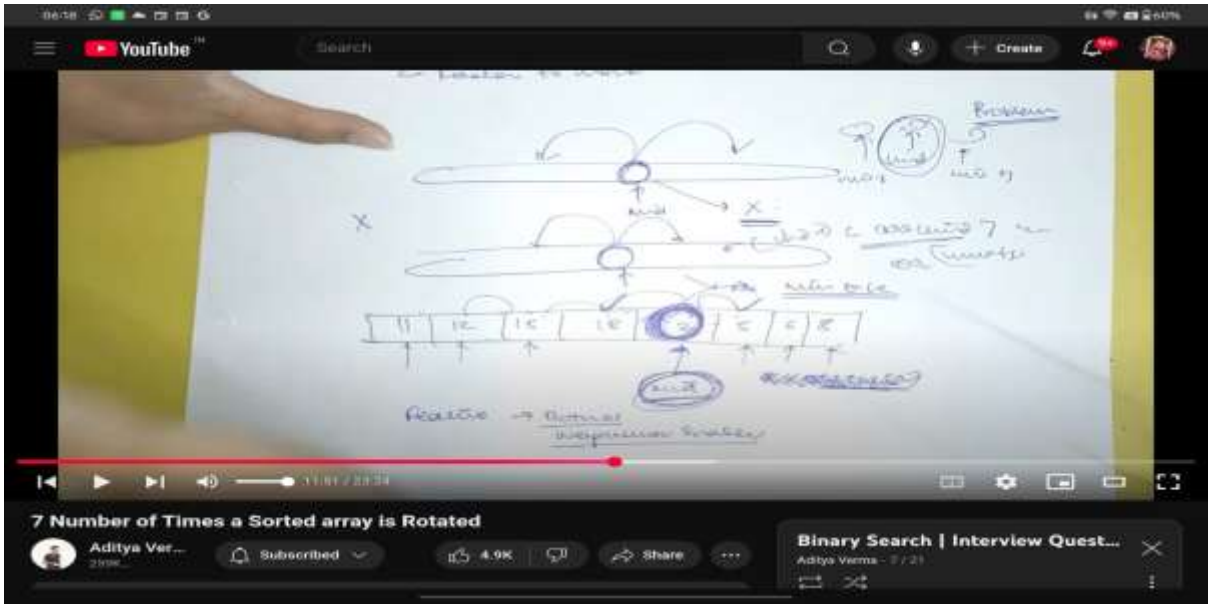

YouTube
aditya verma
Create


7 Number of Times a Sorted array is Rotated

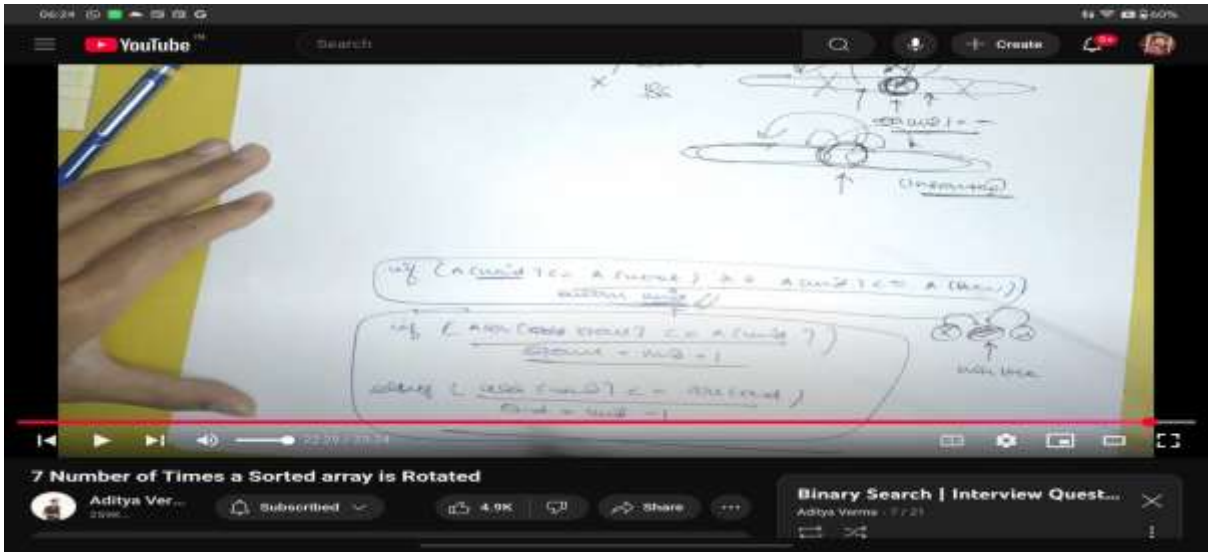

YouTube
Create
pivot element = index


YouTube
Search
Create
7 Number of Times a Sorted array is Rotated
Aditya Ver...
Subscribed
4.9K
Share
Binary Search | Interview Quest...
Aditya Verma

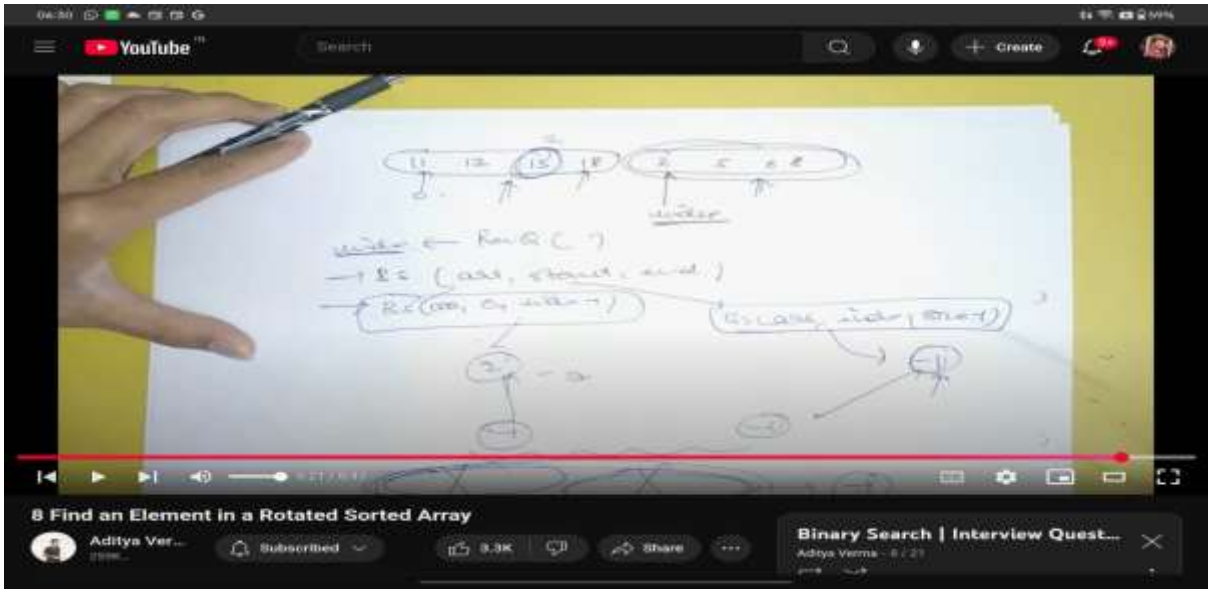

YouTube
Search
Create
7 Number of Times a Sorted array is Rotated
Aditya Ver...
Subscribed
4.9K
Share
Binary Search | Interview Quest...
Aditya Verma

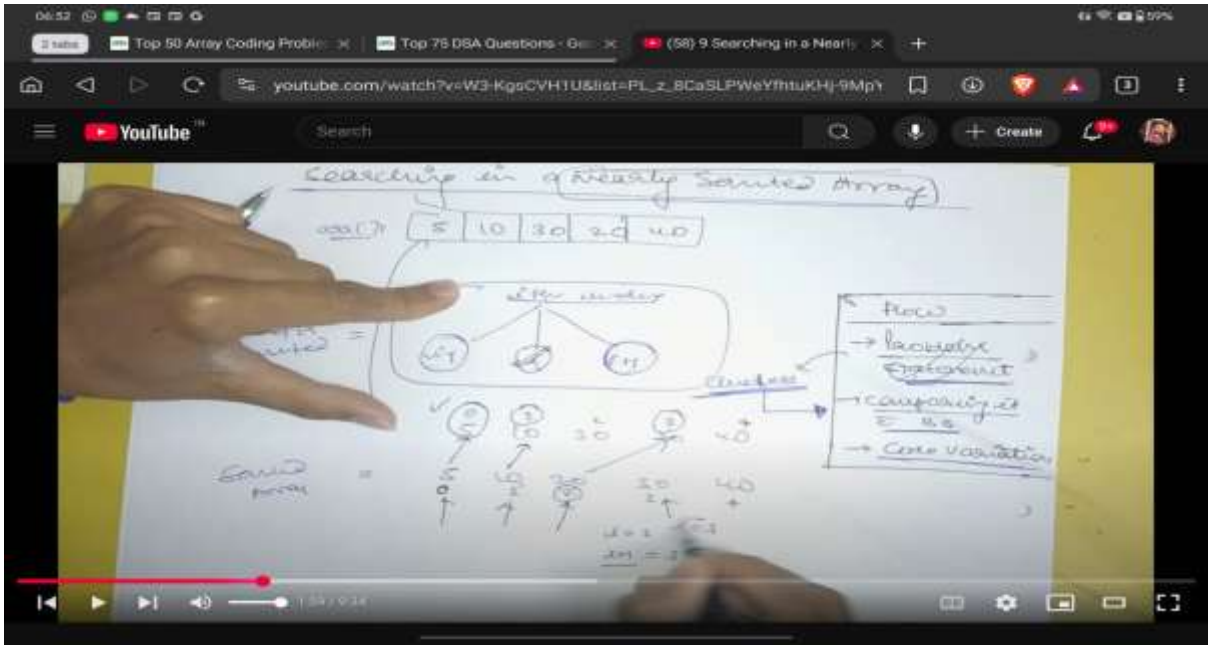

YouTube
Search
Create
7 Number of Times a Sorted array is Rotated
Aditya Ver...
Subscribed
4.9K
Share
Binary Search | Interview Quest...
Aditya Verma

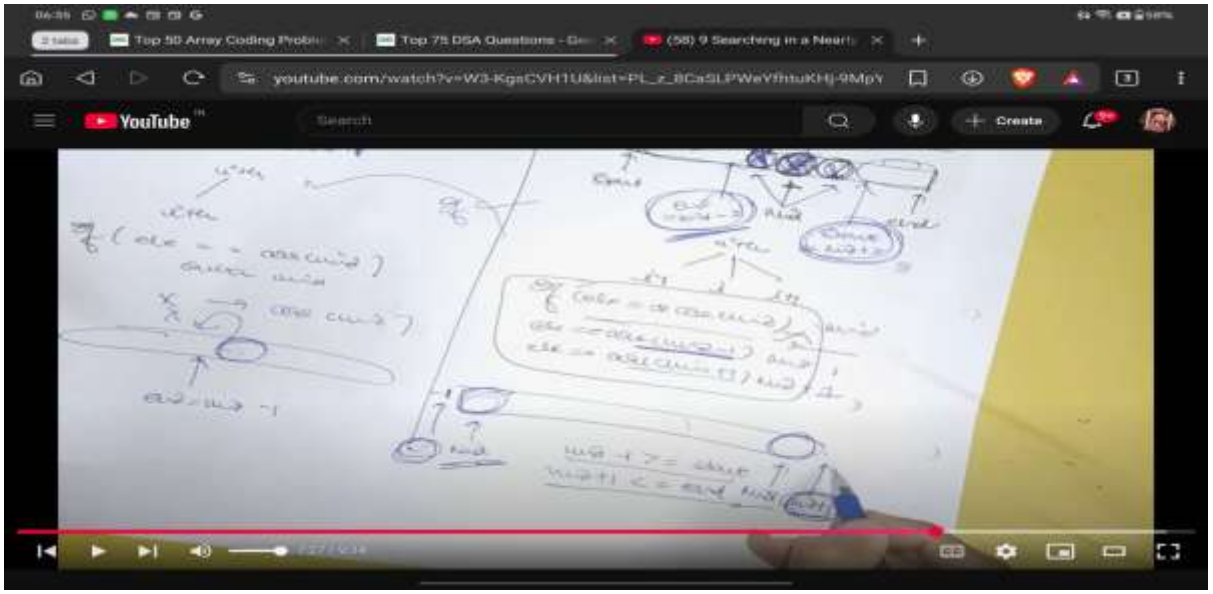

YouTube
Search
Create
7 Number of Times a Sorted array is Rotated
Aditya Ver...
Subscribed
4.9K
Share
Binary Search | Interview Quest...
Aditya Verma

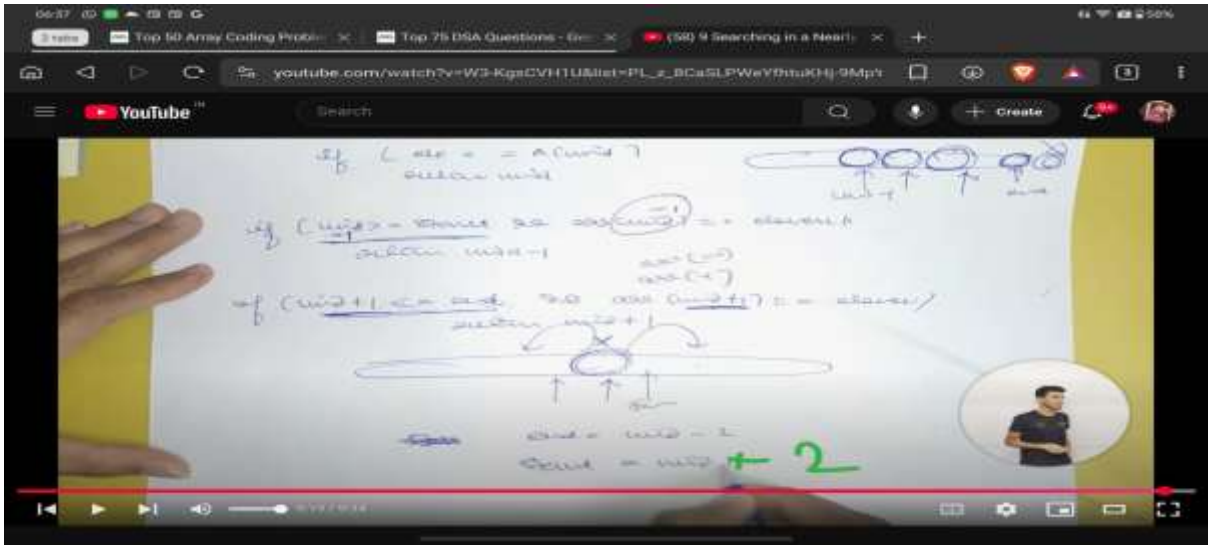

YouTube
Search
Create
7 Number of Times a Sorted array is Rotated
Aditya Ver...
Subscribed
4.9K
Share
Binary Search | Interview Quest...
Aditya Verma

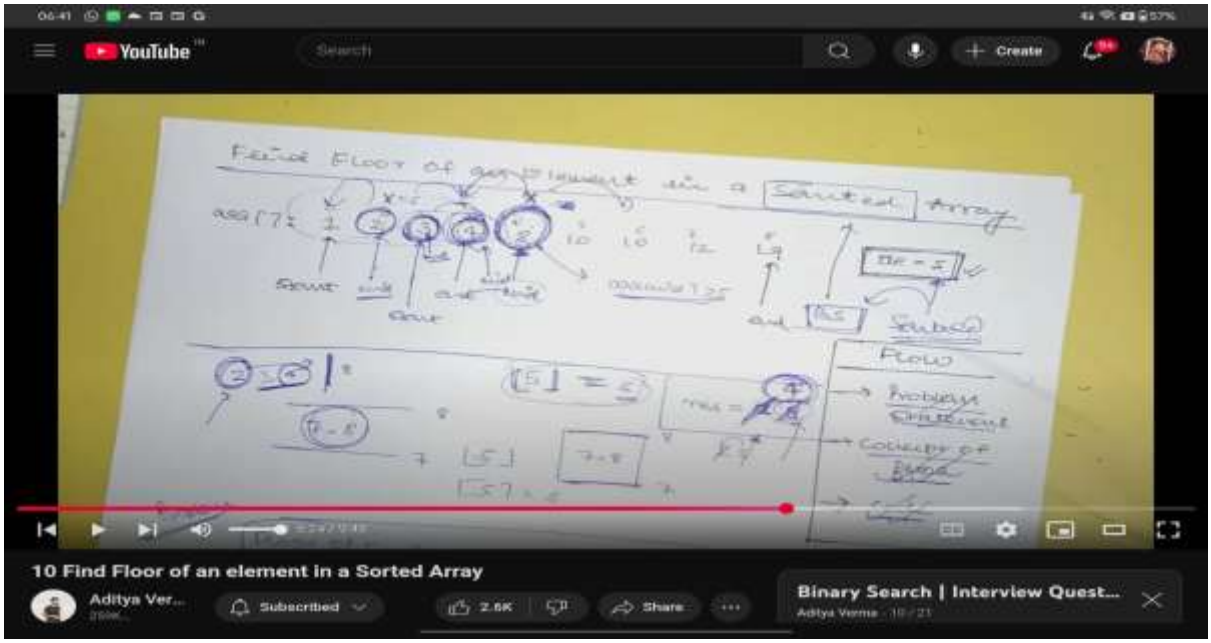

YouTube
Search
Create
Find an Element in a Rotated Sorted Array
Aditya Ver...
Subscribed
Share
Binary Search | Interview Quest...
Aditya Verma

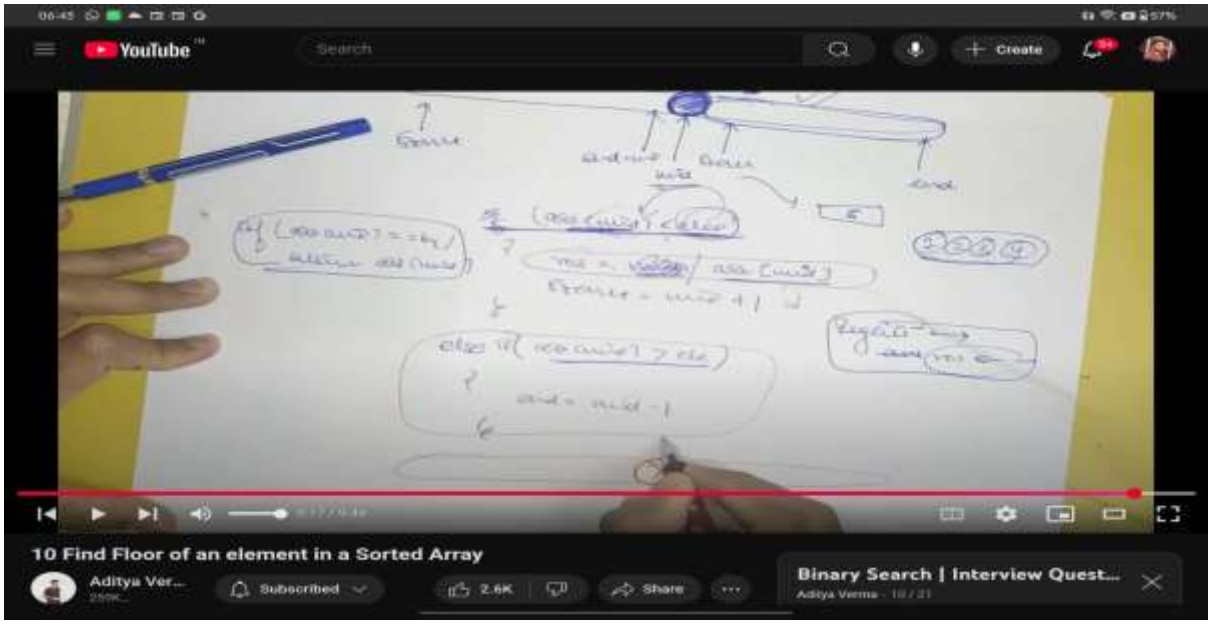

YouTube
Search
Create
8 Find an Element in a Rotated Sorted Array
Aditya Ver...
Subscribed
9.3K
Share
Binary Search | Interview Quest...
Aditya Verma

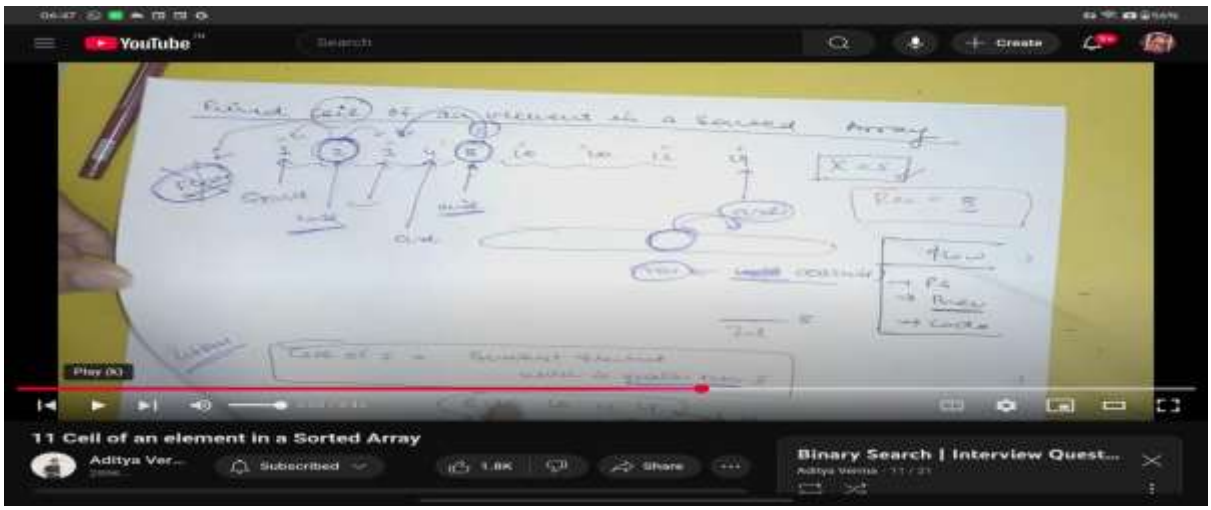

Top 50 Array Coding Proble
Top 75 DSA Questions - Ge
(58) 9 Searching in a Nearly
youtube.com/watch?v=W3-KgsCVH1U&list=PL_z_8CaSLPWeYfhtuKHj-9MpY
YouTube
Search
Create

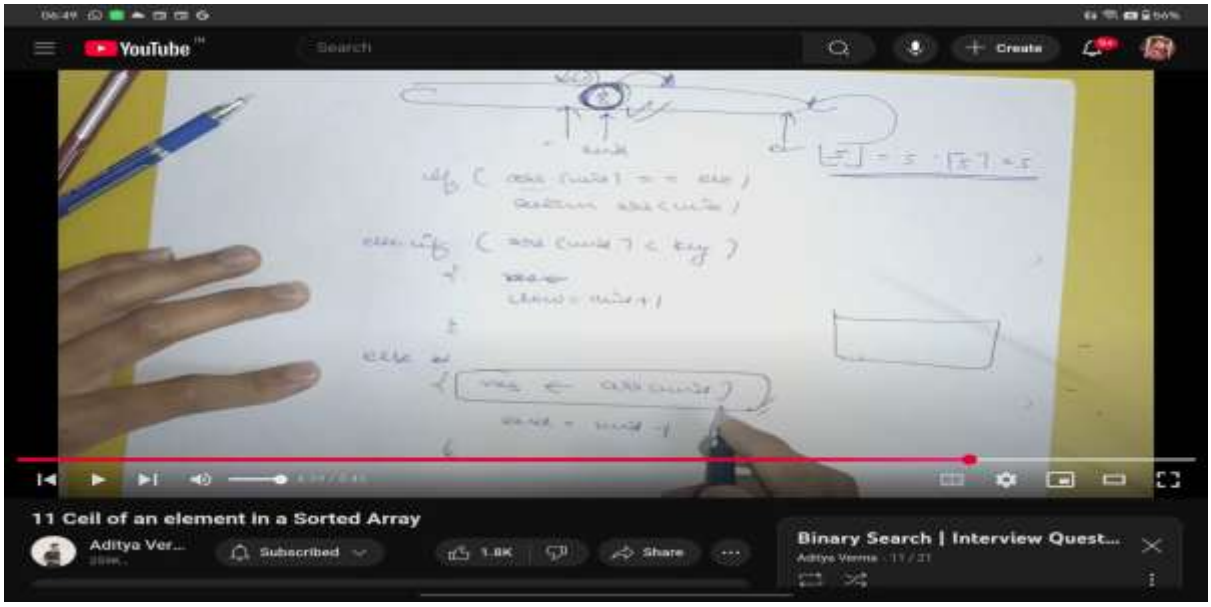

Top 50 Array Coding Proble
Top 75 DSA Questions - Ge
(58) 9 Searching in a Nearly
youtube.com/watch?v=W3-KgsCVH1U&list=PL_z_8CaSLPWeYfhtuKHj-9MpY
YouTube
Search
Create

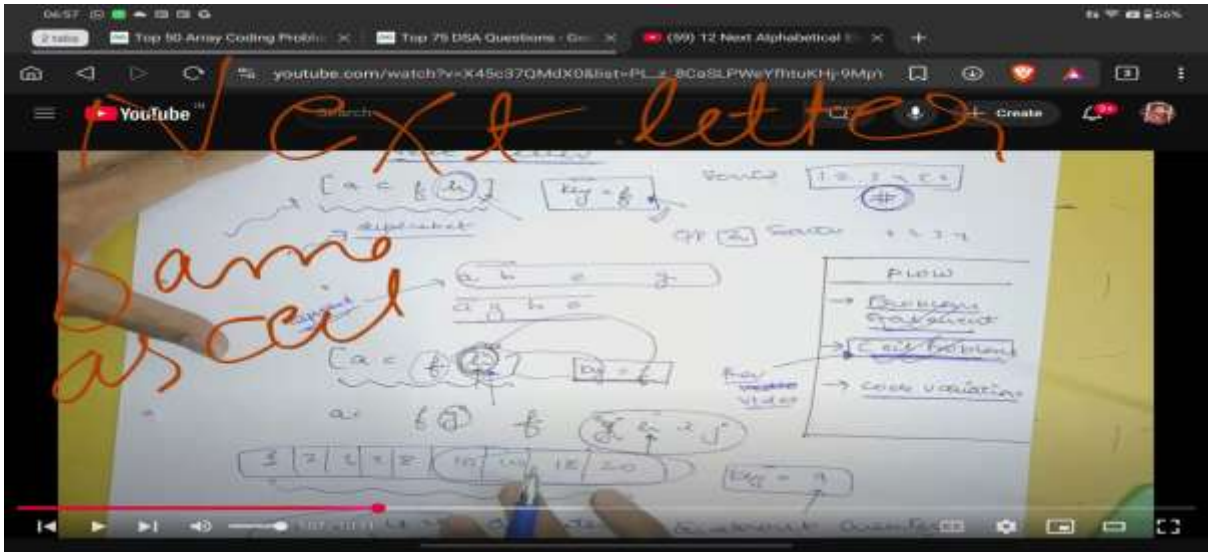

Top 50 Array Coding Proble
Top 75 DSA Questions - Ge
(58) 9 Searching in a Nearly
youtube.com/watch?v=W3-KgsCVH1U&list=PL_z_8CaSLPWeYfhtuKHj-9MpY
YouTube
Search
Create
+ 2

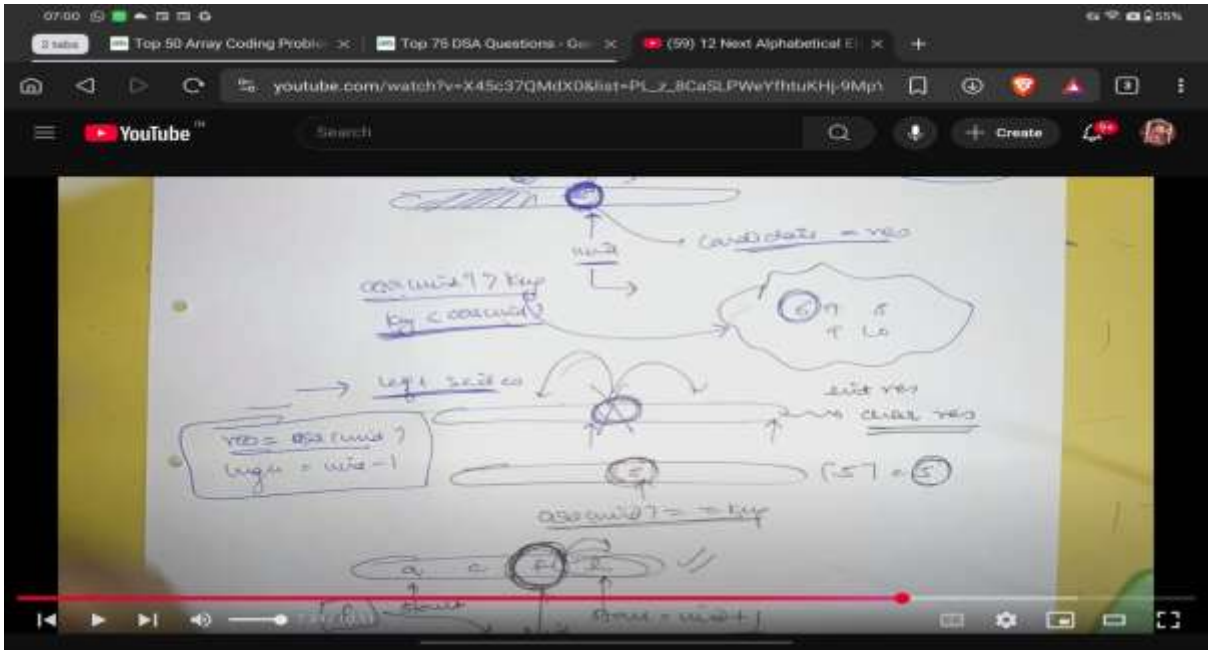

YouTube
Search
Create
10 Find Floor of an element in a Sorted Array
Aditya Ver...
Subscribed
2.6K
Share
Binary Search | Interview Quest...
Aditya Verma

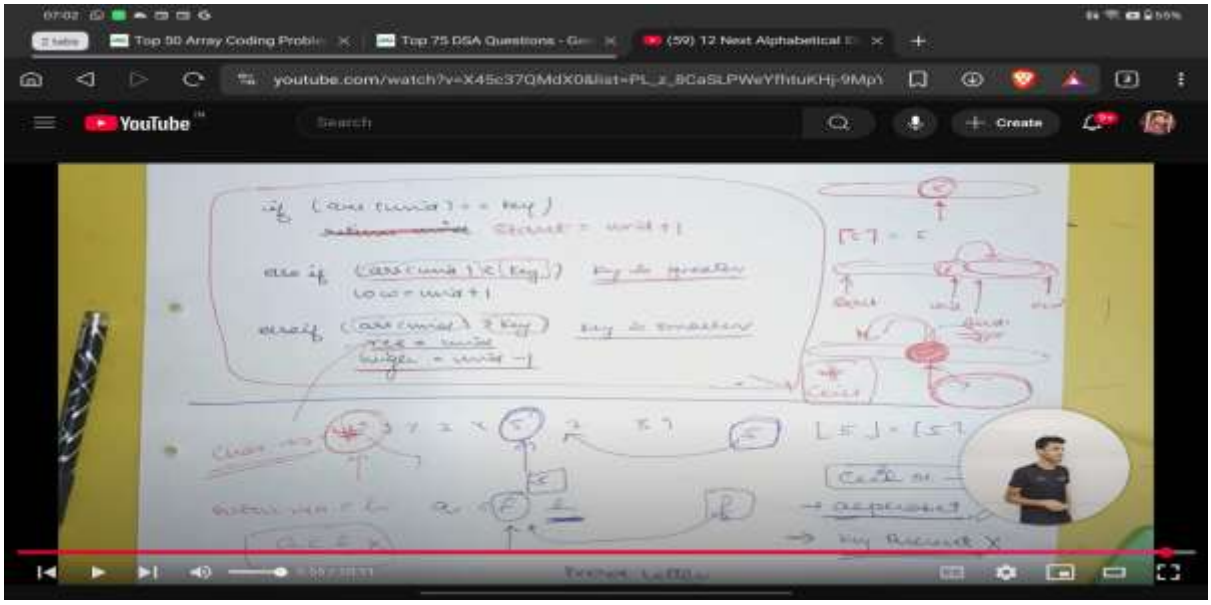

YouTube
Search
Create
10 Find Floor of an element in a Sorted Array
Aditya Ver...
Subscribed
2.6K
Share
Binary Search | Interview Quest...
Aditya Verma

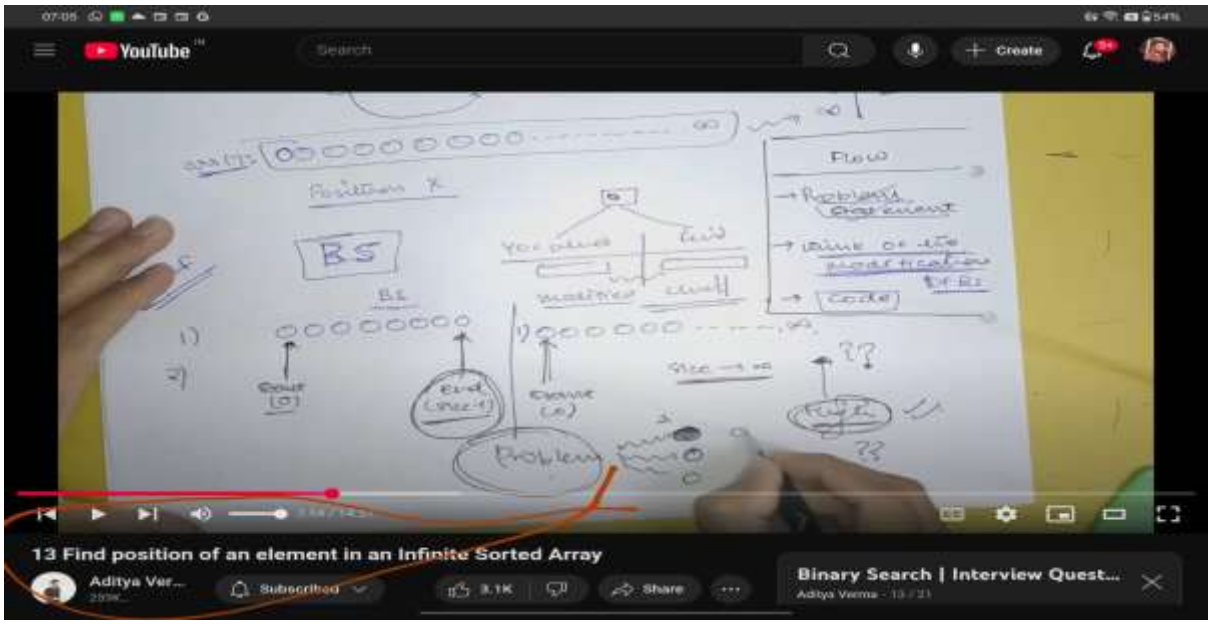

YouTube
Search
Create
11 Cell of an element in a Sorted Array
Aditya Ver...
Subscribed
1.8K
Share
Binary Search | Interview Quest...
Aditya Verma

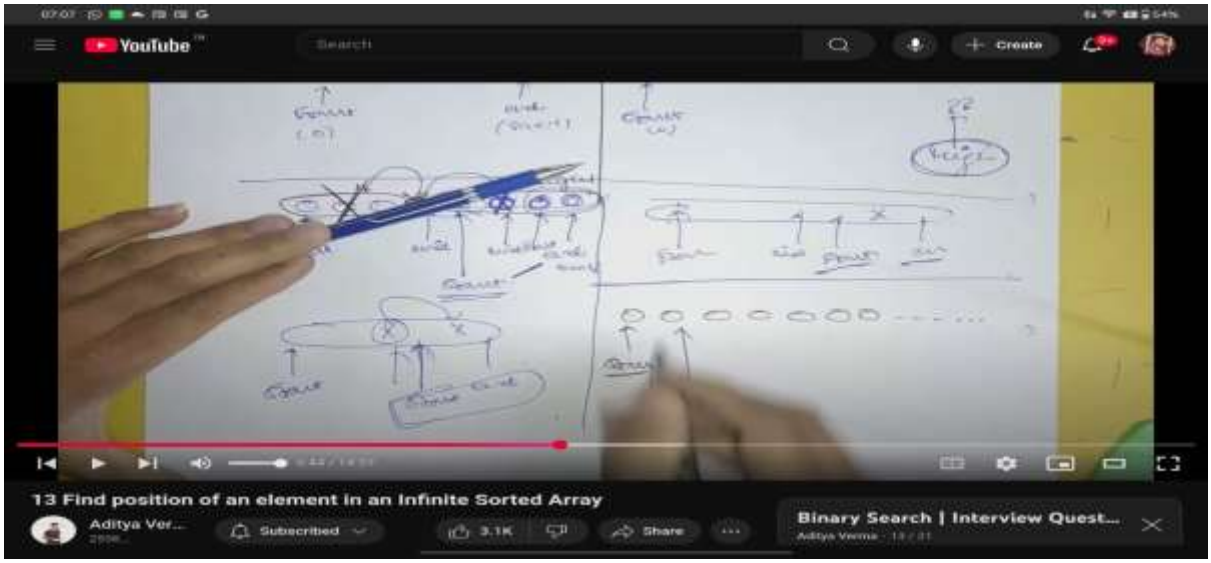

YouTube
Search
Create
11 Ceil of an element in a Sorted Array
Aditya Ver...
Subscribed
1.8K
Share
Binary Search | Interview Quest...

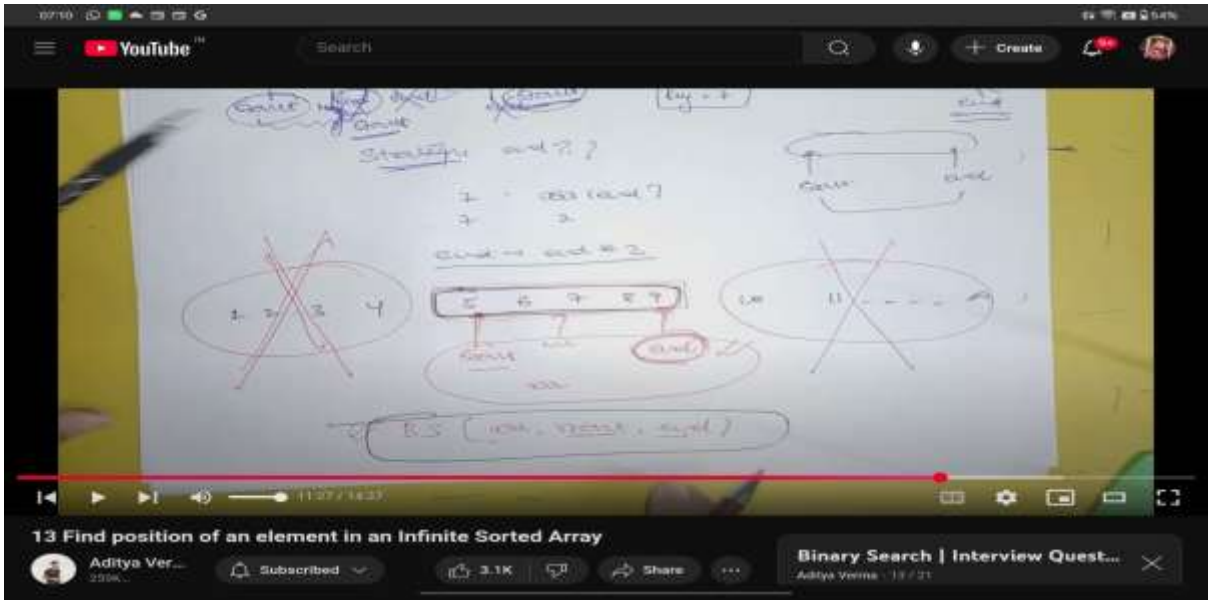

youtube.com/watch?v=X45c37QMdX0&list=PL_z_8CaSLPWeYfhtuKHj-9MpI
YouTube
Search
Create
Next letter
Same
as ceil

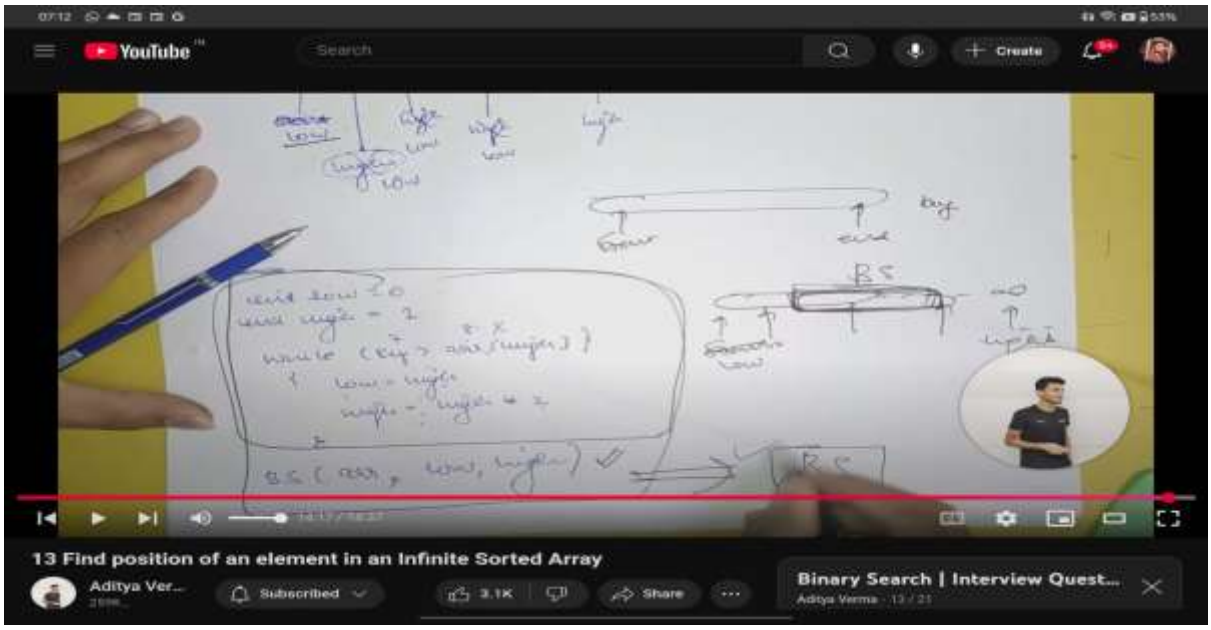

youtube.com/watch?v=X45c37QMdX0&list=PL_z_8CaSLPWeYfhtuKHj-9MpI
YouTube
Search
Create

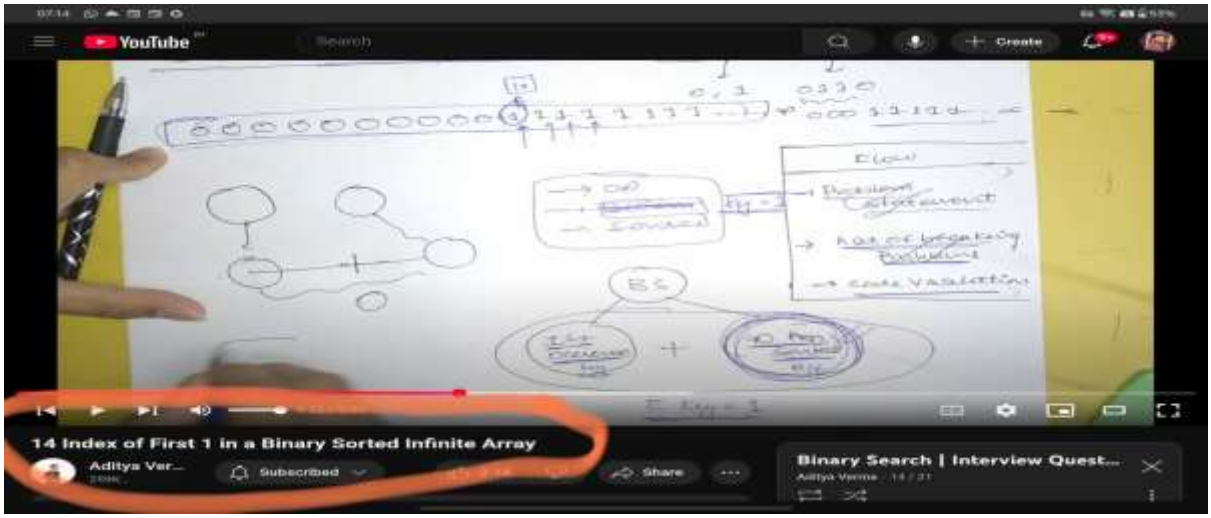

youtube.com/watch?v=X45c37QMdX0&list=PL_z_8CaSLPWeYfhtuKHj-9MpG
YouTube
Search
Create

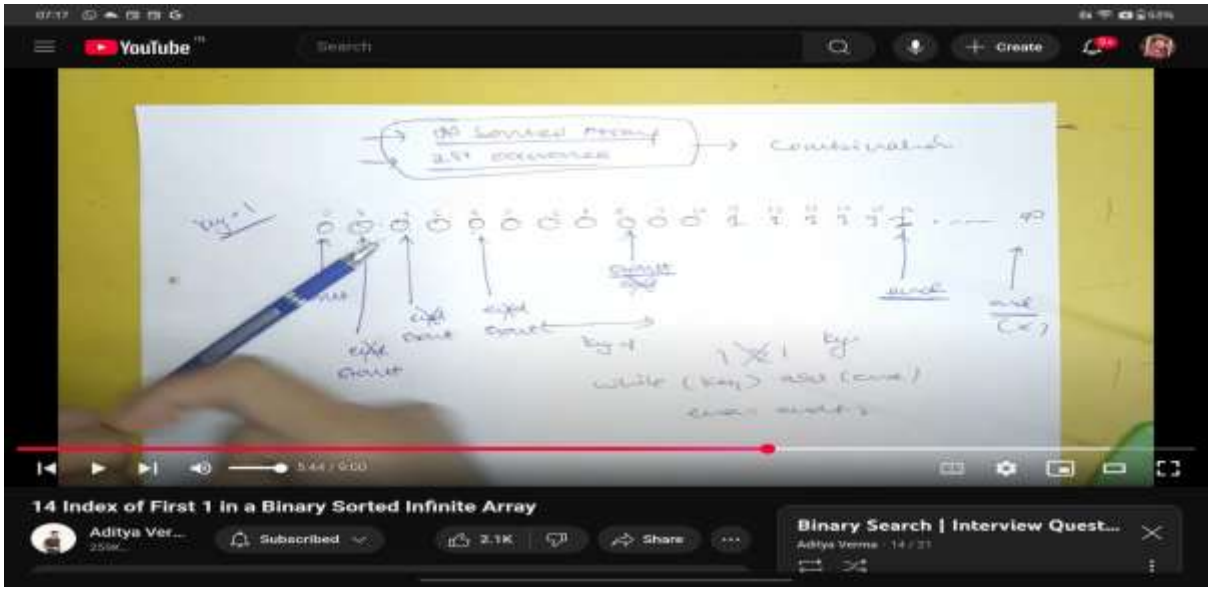

YouTube
Search
Create
Flow
BS
13 Find position of an element in an Infinite Sorted Array
Aditya Ver...
Subscribed
3.1K
Share
Binary Search | Interview Quest...
Aditya Verma

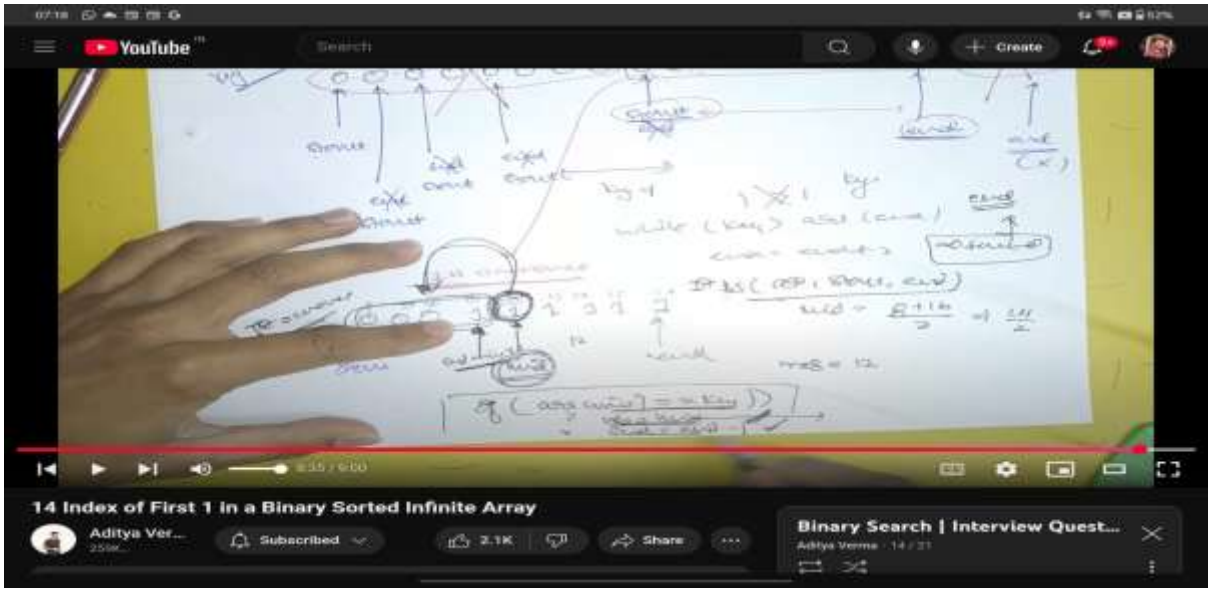

YouTube
Search
Create
13 Find position of an element in an Infinite Sorted Array
Aditya Ver...
Subscribed
3.1K
Share
Binary Search | Interview Quest...
Aditya Verma

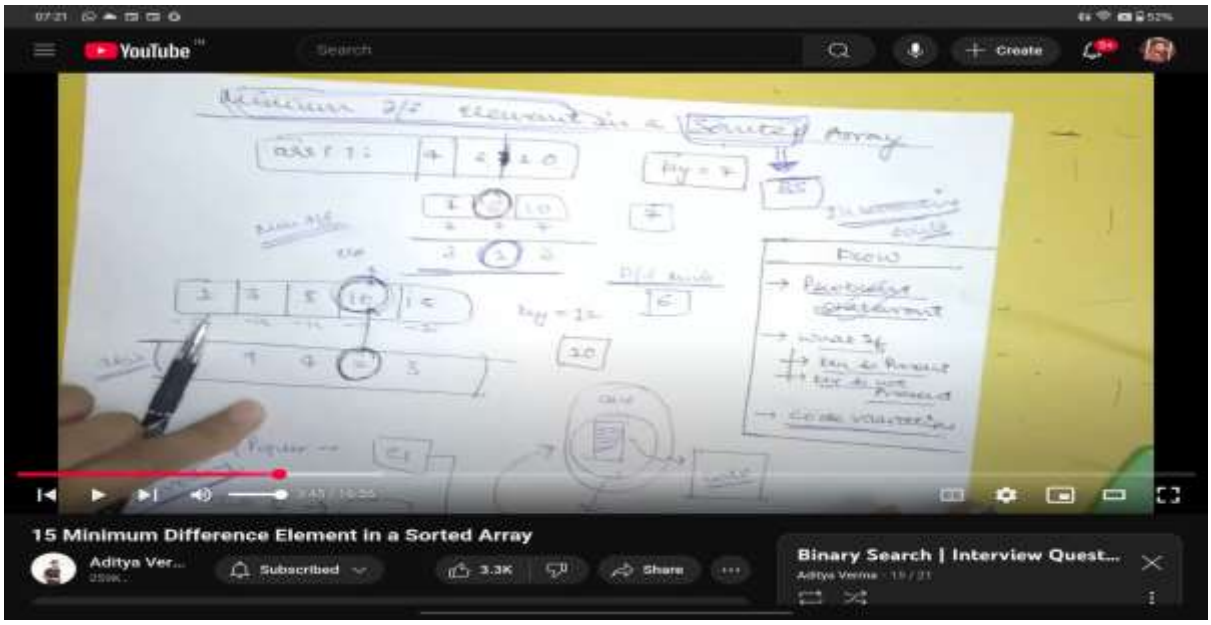

YouTube
Search
Create
13 Find position of an element in an Infinite Sorted Array
Aditya Ver...
Subscribed
3.1K
Share
Binary Search | Interview Quest...
Aditya Verma

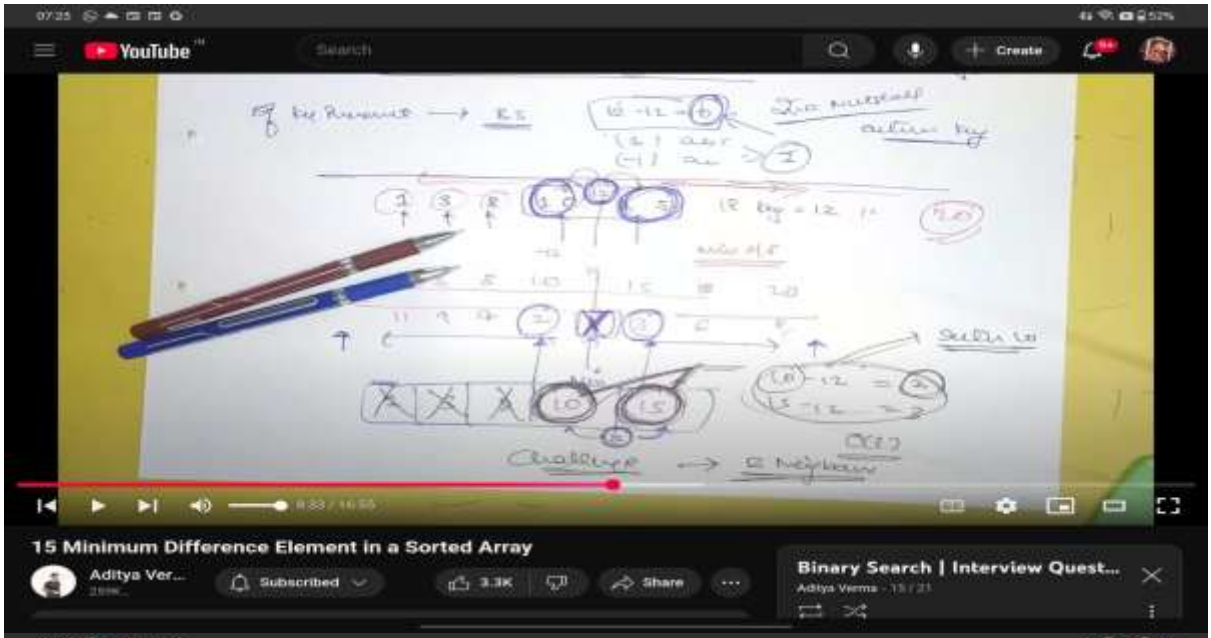

YouTube
Search
Create
13 Find position of an element in an Infinite Sorted Array
Aditya Ver...
Subscribed
3.1K
Share
Binary Search | Interview Quest...
Aditya Verma

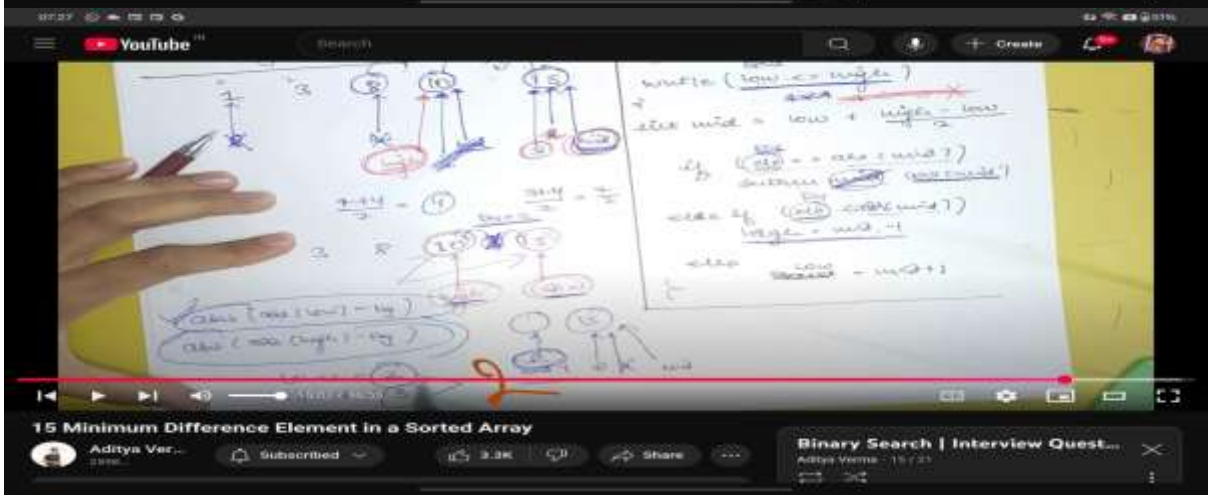

YouTube
Search
Create
14 Index of First 1 in a Binary Sorted Infinite Array
Aditya Ver...
Subscribed
Share
Binary Search | Interview Quest...
Aditya Verma

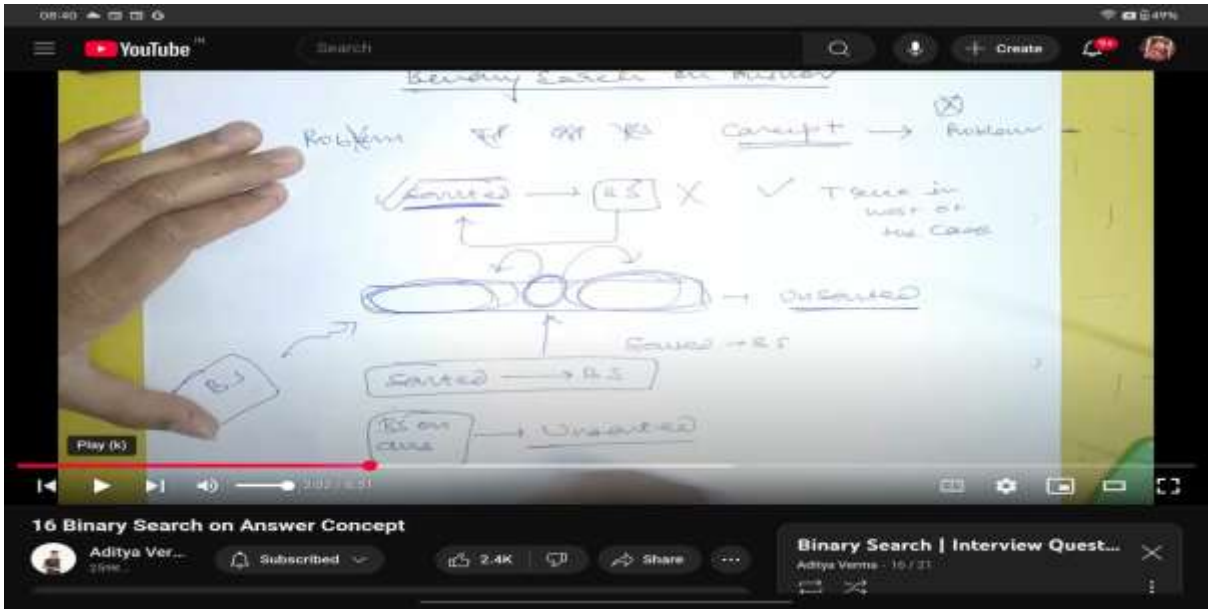

YouTube
Search
Create
14 Index of First 1 in a Binary Sorted Infinite Array
Aditya Ver...
Subscribed
2.1K
Share
Binary Search | Interview Quest...
Aditya Verma · 14 / 21

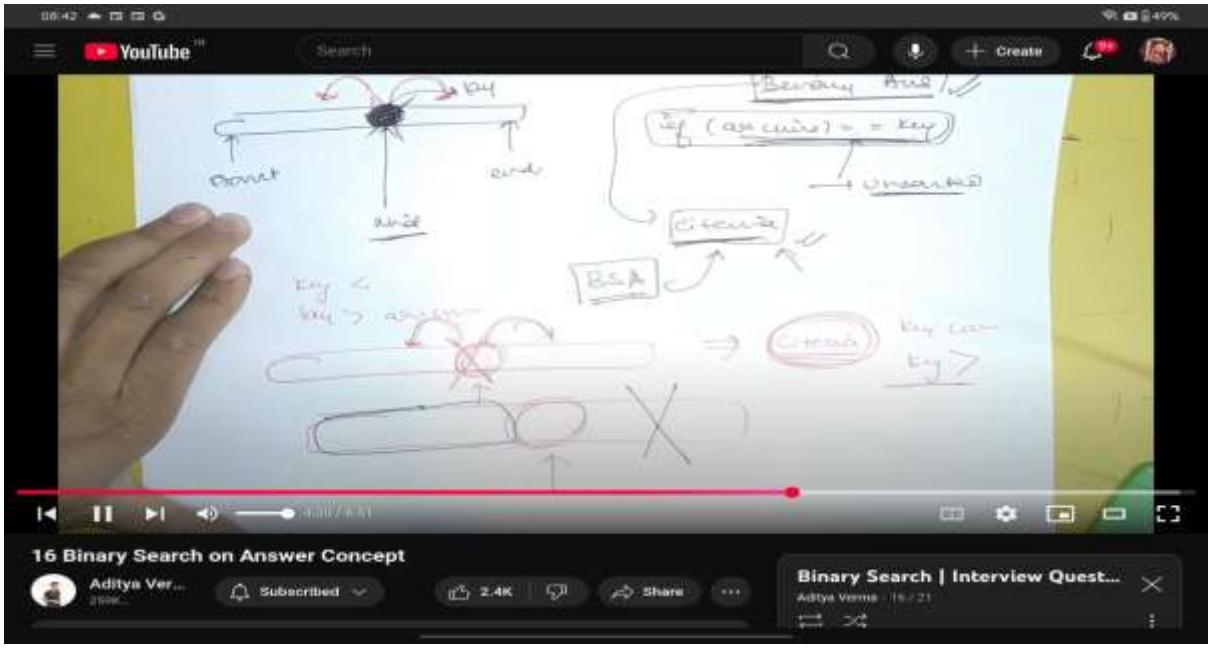

YouTube
Search
Create
14 Index of First 1 in a Binary Sorted Infinite Array
Aditya Ver...
Subscribed
2.1K
Share
Binary Search | Interview Quest...
Aditya Verma · 14 / 21

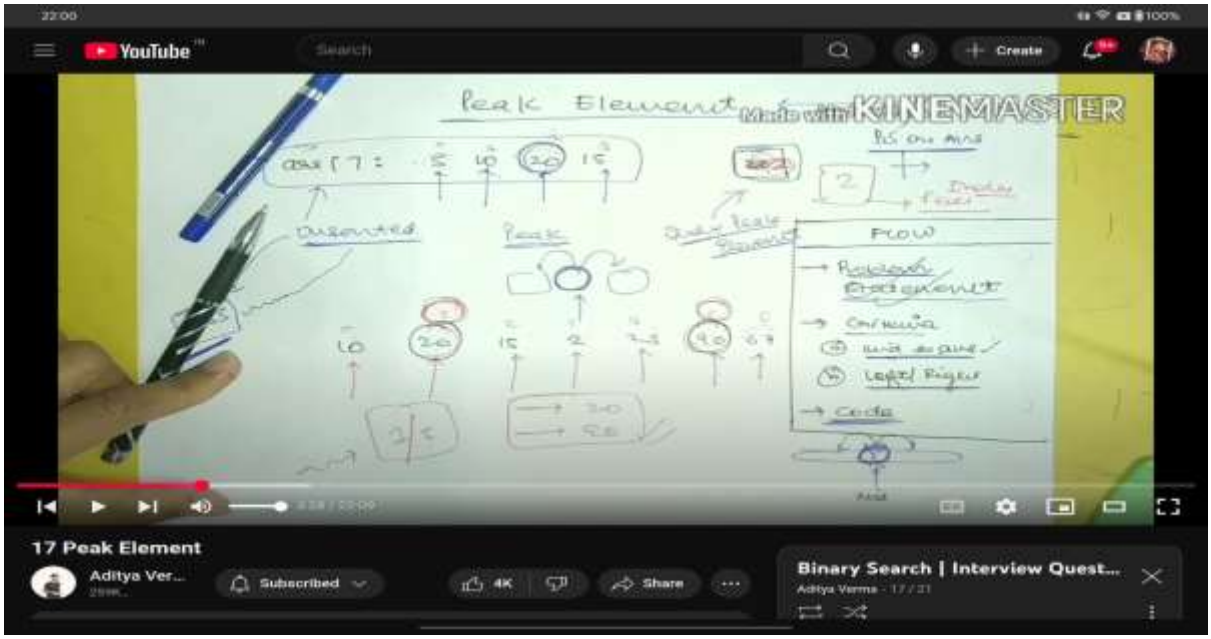

YouTube
Search
Create
15 Minimum Difference Element in a Sorted Array
Aditya Ver...
Subscribed
3.3K
Share
Binary Search | Interview Quest...
Aditya Verma · 15 / 21

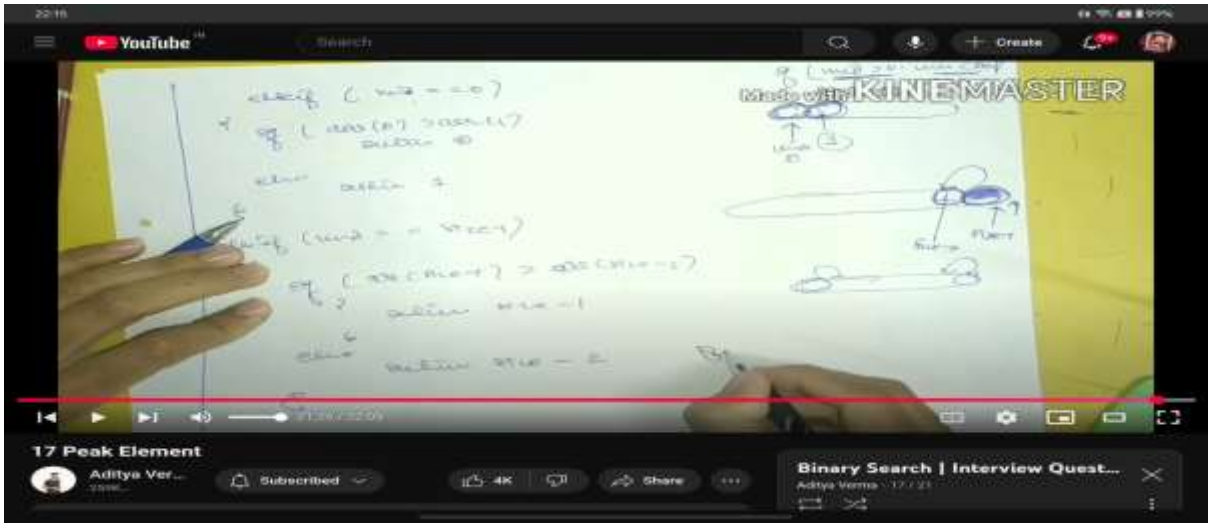

YouTube
Search
Create
15 Minimum Difference Element in a Sorted Array
Aditya Ver...
Subscribed
3.3K
Share
Binary Search | Interview Quest...
Aditya Verma · 15 / 21

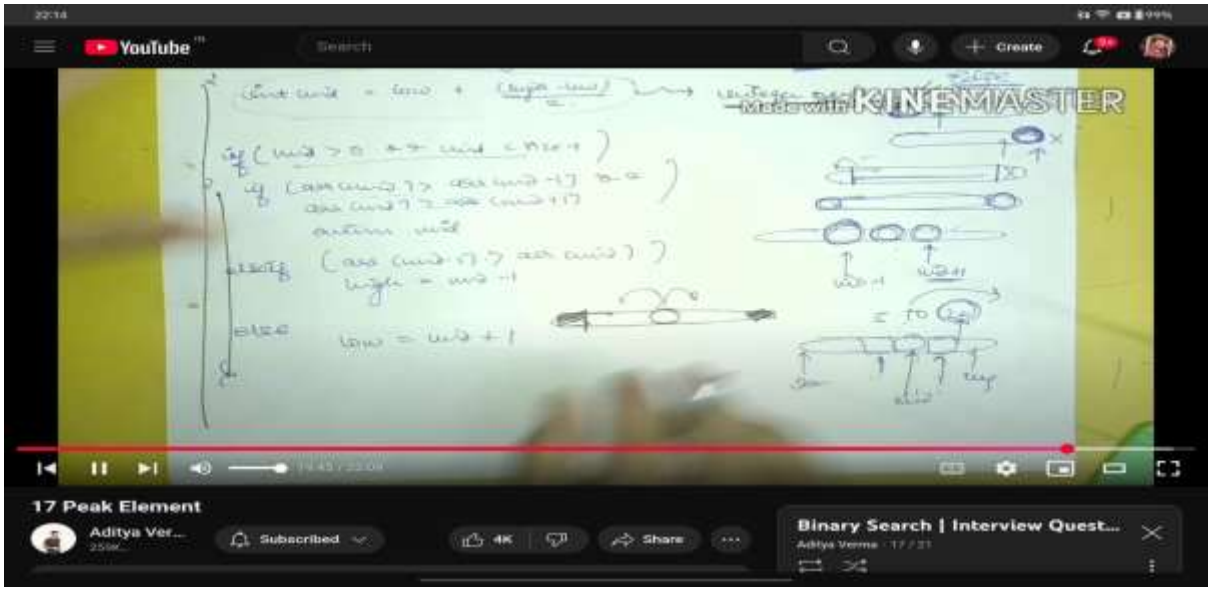

YouTube
Search
Create
15 Minimum Difference Element in a Sorted Array
Aditya Ver...
Subscribed
3.3K
Share
Binary Search | Interview Quest...
Aditya Verma · 15 / 21

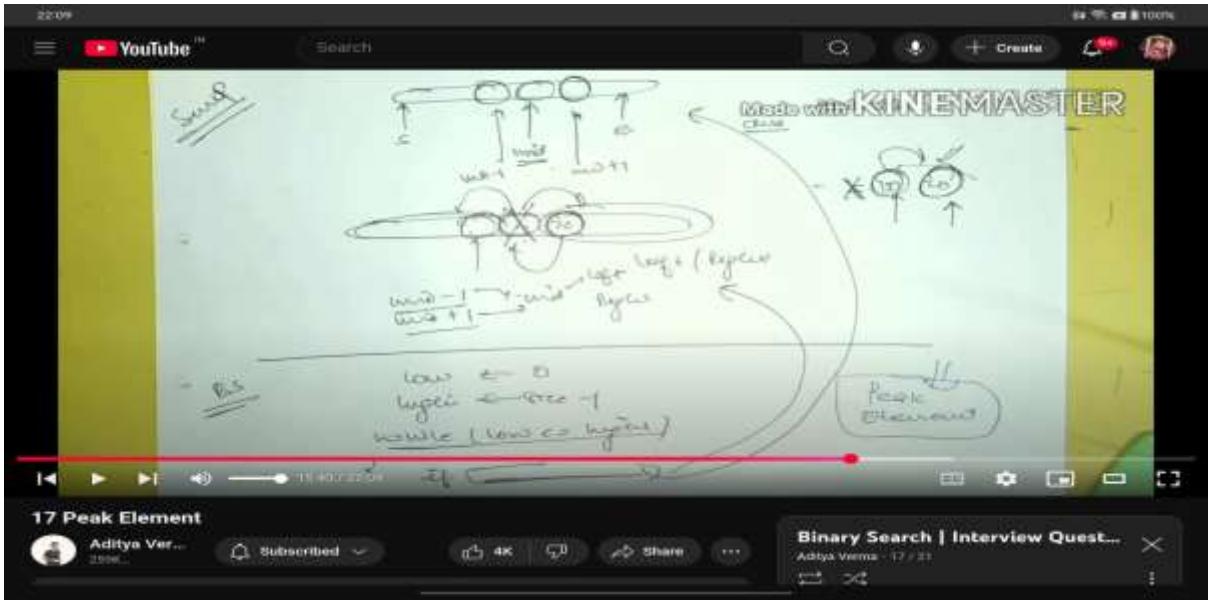

YouTube
Search
Create
Play (k)
16 Binary Search on Answer Concept
Aditya Ver...
Subscribed
2.4K
Share
Binary Search | Interview Quest...
Aditya Verma · 16 / 21

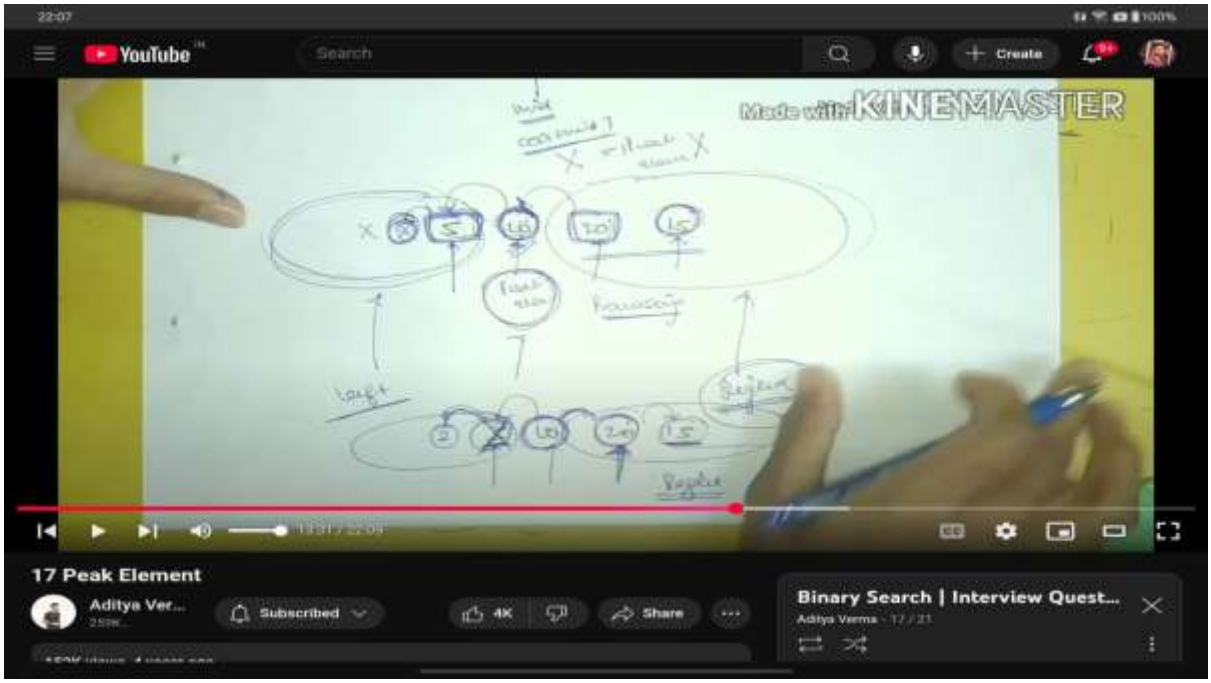

YouTube
Search
Create
16 Binary Search on Answer Concept
Aditya Ver...
Subscribed
2.4K
Share
Binary Search | Interview Quest...
Aditya Verma

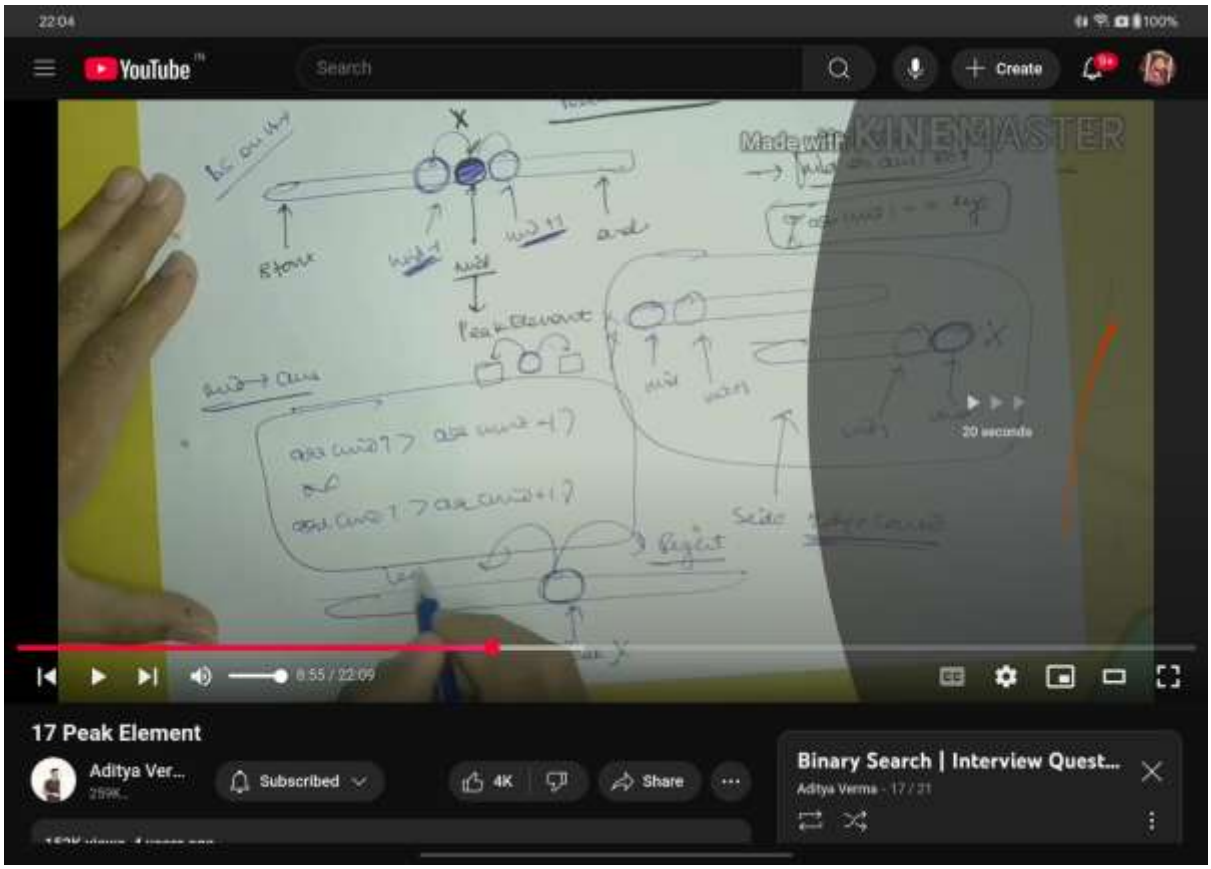

YouTube
Search
Create
Peak Element
Made with KINEMASTER
17 Peak Element
Aditya Ver...
Subscribed
4K
Share
Binary Search | Interview Quest...
Aditya Verma


YouTube
Search
Create
Made with KINEMASTER
17 Peak Element
Aditya Ver...
Subscribed
4K
Share
Binary Search | Interview Quest...
Aditya Verma


YouTube
Search
Create
Made with KINEMASTER
17 Peak Element
Aditya Ver...
Subscribed
4K
Share
Binary Search | Interview Quest...
Aditya Verma · 17 / 21


YouTube
Search
Create
Made with KINEMASTER
Peak Element
17 Peak Element
Aditya Ver...
Subscribed
4K
Share
Binary Search | Interview Quest...
Aditya Verma · 17 / 21


YouTube
Search
Create
Made with KINEMASTER
17 Peak Element
Aditya Ver...
Subscribed
4K
Share
Binary Search | Interview Quest...
Aditya Verma · 17 / 21

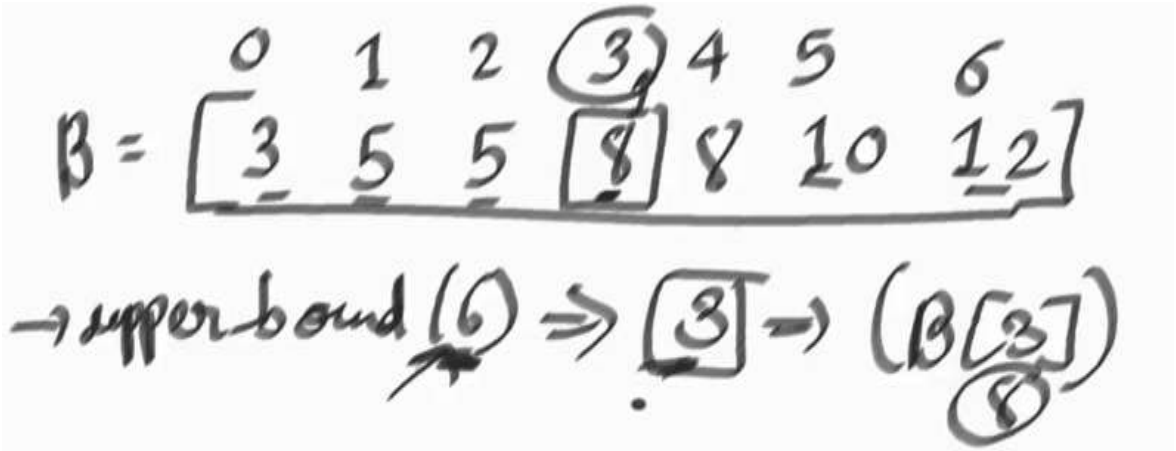

YouTube
Search
Create
Made with KINEMASTER
20 seconds
8:55 / 22:09
17 Peak Element
Aditya Ver...
Subscribed
4K
Share
Binary Search | Interview Quest...
Aditya Verma · 17 / 21

DesiQna

Upper Bound

-> Given a sorted array of size "N"; find the index of the number in the array which is just greater than x and as close as possible to x.

-> Upper Bound (x) = Returns index of the number which is just greater than x and as close as possible to x.




```java
class GFG {
    public static int upperBound(int[] arr, int target) {
        int left = 0;
        int right = arr.length - 1;

        while (left <= right) {
            int mid = (left + right)/2 ;
            if (arr[mid] <= target) {
                left = mid + 1;
            } else {
                if(mid==0){
                    return 0;
                }

                else{
                    if(arr[mid-1]<=target){
                     return mid ;
                    }
                    else{
                     right = mid - 1 ;
                    }

                }


            }
        }

        return left;
    }
```

Search in Rotated Sorted Array

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`*, or* `-1` *if it is not in* `nums`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

> **Input:** nums = [4,5,6,7,0,1,2], target = 0
> **Output:** 4

Example 2:

> **Input:** nums = [4,5,6,7,0,1,2], target = 3
> **Output:** -1

Example 3:

> **Input:** nums = [1], target = 0
> **Output:** -1

Constraints:

- `1 <= nums.length <= 5000`
- `-10^4 <= nums[i] <= 10^4`

```java
public int search(int[] nums, int target) {

    int start=0;
    int end=nums.length-1;

    while(start<=end){
        int mid =(start+end)/2;
        if(nums[mid]==target){
            return mid;

        }else if(nums[start]<=nums[mid]){
            if(target>=nums[start] && target<nums[mid]){
                end=mid-1;
            }else{
                start=mid+1;
            }

        }else{
                if(target>nums[mid] && target<=nums[end]){
                    start=mid+1;
                }else{
                    end=mid-1;
                }
           }

    }
     return -1;
```

540. Single Element in a Sorted Array

Solved

Medium | Topics | Companies

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once.

Return *the single element that appears only once.*

Your solution must run in `O(log n)` time and `O(1)` space.

Example 1:

```
Input: nums = [1,1,2,3,3,4,4,8,8]
Output: 2
```

Example 2:

```
Input: nums = [3,3,7,7,10,11,11]
Output: 10
```

Intuition of this Problem:

Since every element in the sorted array appears exactly twice except for the single element, we know that if we take any element at an even index (0-indexed), the next element should be the same. Similarly, if we take any element at an odd index, the previous element should be the same. Therefore, we can use binary search to compare the middle element with its adjacent elements to determine which side of the array the single element is on.

If the middle element is at an even index, then the single element must be on the right side of the array, since all the elements on the left side should come in pairs. Similarly, if the middle element is at an odd index, then the single element must be on the left side of the array.

We can continue this process by dividing the search range in half each time, until we find the single element.

Another interesting observation you might have made is that this algorithm will still work even if the array isn't fully sorted. As long as pairs are always grouped together in the array (for example, [10, 10, 4, 4, 7, 11, 11, 12, 12, 2, 2]), it doesn't matter what order they're in. Binary search worked for this problem because we knew the subarray with the single number is always odd-lengthed, not because the array was fully sorted numerically

Approach for this Problem:

1. Initialize two pointers, left and right, to the first and last indices of the input array, respectively.
2. While the left pointer is less than the right pointer:
   - a. Compute the index of the middle element by adding left and right and dividing by 2.
   - b. If the index of the middle element is odd, subtract 1 to make it even.
   - c. Compare the middle element with its adjacent element on the right:
     - i. If the middle element is not equal to its right neighbor, the single element must be on the left side of the array, so update the right pointer to be the current middle index.
     - ii. Otherwise, the single element must be on the right side of the array, so update the left pointer to be the middle index plus 2.
3. When the left and right pointers converge to a single element, return that element.

```java
class Solution {
    public int singleNonDuplicate(int[] nums) {


        int left=0;
        int right=nums.length-1;

        while(left<right){
            int mid=(left+right)/2;
            if(mid%2==1){
                mid--;
            }
            if(nums[mid] !=nums[mid+1]){
                right=mid;
            }else{
                left=mid+2;
            }
        }
        return nums[left];
    }
}
```

P0 :- Given a sorted array of 0's and 1's f-> Binary search :- O(logN) low = 0
high = n - 1

```
while(low<=high){
    Mid = (low+high)/2
   if(b[mid]==0){
        low = mid + 1
    }
    else {
          //b[mid]==1
         if(b[mid-1]==1){
              high = mid - 1
          }
         Else{
         Answer = mid
         break
          }
     }
}
```

ind the first occurrence of 1

-> O(N) :- Start traveling the array initially and all zeroes will be there; as soon as you find a "1" coming :-> Break the algorithm and print its index.

-> Binary search :- O(logN) low = 0
high = n - 1

Problem Sets

1.Search in a sorted array

Given an array of integers `nums` which is sorted in ascending order, and an integer `target`, write a function to search `target` in `nums`. If `target` exists, then return its index. Otherwise, return `-1`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

> **Input:** nums = [-1,0,3,5,9,12], target = 9
> **Output:** 4
> **Explanation:** 9 exists in nums and its index is 4

Example 2:

> **Input:** nums = [-1,0,3,5,9,12], target = 2
> **Output:** -1
> **Explanation:** 2 does not exist in nums so return -1

```java
class Solution {
    public int search(int[] nums, int target) {

        int start=0;
        int end=nums.length -1;

        while(start<=end){

            int mid =(start+end)/2;

            if(nums[mid] ==target){
                return mid;
            }else if(target>nums[mid]){
                start=mid+1;
            }else{
                end=mid-1;
            }
        }
        return -1;
    }
}
```

2.Search Insert position

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [1,3,5,6], target = 5
Output: 2
```

Example 2:

```
Input: nums = [1,3,5,6], target = 2
Output: 1
```

Example 3:

```
Input: nums = [1,3,5,6], target = 7
Output: 4
```

```java
class Solution {
    public int searchInsert(int[] nums, int target) {

        int start=0;
        int end=nums.length-1;

        while(start<=end){

            int mid =(start+end)/2;

            if(nums[mid] == target){
                return mid;
            }else if(target>=nums[mid]){
                start =start+1;
            }else{
                end=end-1;
            }
        }
        return start;

    }
}
```

3.Search a 2D Matrix

You are given an `m x n` integer matrix `matrix` with the following two properties:

- Each row is sorted in non-decreasing order.
- The first integer of each row is greater than the last integer of the previous row.

Given an integer `target`, return `true` *if* `target` *is in* `matrix` *or* `false` *otherwise.*

You must write a solution in `O(log(m * n))` time complexity.

Example 1:

1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3
Output: true

```java
public boolean searchMatrix(int[][] matrix, int target) {

    if(matrix==null || matrix.length==0 || matrix[0].length==0)
    {
        return false;

    }
    int m= matrix.length;
    int n= matrix[0].length;
    int start= 0;
    int end= m*n-1;
    while(start<=end)
    {
        int mid=(start+end)/2;
        int midX=mid/n;
        int midY=mid%n;
        if(matrix[midX][midY]==target)
        {
            return true;
        }
        if(matrix[midX][midY] < target)
        {
            start=mid+1;
        }
        else
        {
            end=mid-1;
        }
    }
    return false;
}
```

4.Sqrt(x)

Given a non-negative integer `x`, return *the square root of* `x` *rounded down to the nearest integer*. The returned integer should be **non-negative** as well.

You **must not use** any built-in exponent function or operator.

- For example, do not use `pow(x, 0.5)` in c++ or `x ** 0.5` in python.

Example 1:

> **Input:** x = 4
> **Output:** 2
> **Explanation:** The square root of 4 is 2, so we return 2.

Example 2:

> **Input:** x = 8
> **Output:** 2
> **Explanation:** The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.

```java
class Solution {
    public int mySqrt(int x) {
        if(x<2){
            return x;
        }
        int start=0;
        int end=x;
        int ans=0;

        while(start<=end){
            int mid=start + (end-start)/2;
            if(mid == x/mid){
                ans=mid;
                break;
            }else if(mid < x/mid){
                ans=mid;
                start=mid+1;
            }else{
                end=mid-1;
            }
        }
        return ans;

    }
}
```

5.First Bad Version

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have `n` versions `[1, 2, ..., n]` and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API `bool isBadVersion(version)` which returns whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example 1:

```
Input: n = 5, bad = 4
Output: 4
Explanation:
call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true
Then 4 is the first bad version.
```

Example 2:

```
Input: n = 1, bad = 1
Output: 1
```

```java
public class Solution extends VersionControl {

    public int firstBadVersion(int n) {

        int left=1;
        int right=n;

        while(left < right){
            int mid = left + (right-left)/2;
            if(isBadVersion(mid)){
                right=mid;
            }else{
                left=mid+1;
            }
        }
        return left;

    }
}
```

6.Find Smallest letter greater than target

You are given an array of characters `letters` that is sorted in **non-decreasing order**, and a character `target`. There are **at least two different** characters in `letters`.

Return *the smallest character in* `letters` *that is lexicographically greater than* `target`. If such a character does not exist, return the first character in `letters`.

Example 1:

> **Input:** letters = ["c","f","j"], target = "a"
> **Output:** "c"
> **Explanation:** The smallest character that is lexicographically greater than 'a' in letters is 'c'.

Example 2:

> **Input:** letters = ["c","f","j"], target = "c"
> **Output:** "f"
> **Explanation:** The smallest character that is lexicographically greater than 'c' in letters is 'f'.

Example 3:

> **Input:** letters = ["x","x","y","y"], target = "z"
> **Output:** "x"
> **Explanation:** There are no characters in letters that is lexicographically greater than 'z' so we return letters[0].

```java
class Solution {
    public char nextGreatestLetter(char[] letters, char target) {
        int low = 0, high = letters.length;

        while (low < high) {
            int mid = low + (high - low) / 2;
            if (letters[mid] <= target) {
                low = mid + 1;
            } else {
                high = mid;
            }
        }

        return letters[low % letters.length];
    }
}
```

7.Find Minimum in rotated sorted array

Suppose an array of length `n` sorted in ascending order is **rotated** between `1` and `n` times. For example, the array `nums = [0,1,2,4,5,6,7]` might become:

- `[4,5,6,7,0,1,2]` if it was rotated `4` times.
- `[0,1,2,4,5,6,7]` if it was rotated `7` times.

Notice that **rotating** an array `[a[0], a[1], a[2], ..., a[n-1]]` 1 time results in the array `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]`.

Given the sorted rotated array `nums` of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in `O(log n) time`.

Example 1:

> **Input:** nums = [3,4,5,1,2]
> **Output:** 1
> **Explanation:** The original array was [1,2,3,4,5] rotated 3 times.

Example 2:

> **Input:** nums = [4,5,6,7,0,1,2]
> **Output:** 0
> **Explanation:** The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.

Example 3:

> **Input:** nums = [11,13,15,17]
> **Output:** 11
> **Explanation:** The original array was [11,13,15,17] and it was rotated 4 times.

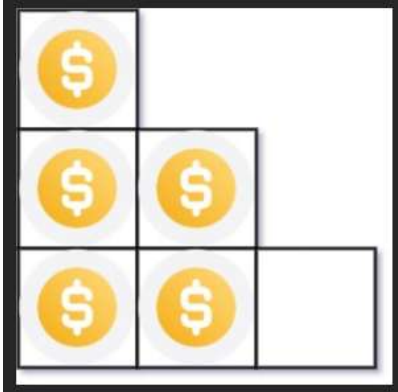
```java
class Solution {
    public int findMin(int[] nums) {
        int n= nums.length;
        int mini=Integer.MAX_VALUE;
        for(int i=0;i<n;i++){
            mini=Math.min(mini,nums[i]);

        }
        return mini;

    }
}
```

8.Arranging Coins

You have `n` coins and you want to build a staircase with these coins. The staircase consists of `k` rows where the `i^th` row has exactly `i` coins. The last row of the staircase **may be** incomplete.

Given the integer `n`, return *the number of **complete rows** of the staircase you will build.*

Example 1:



Input: n = 5
Output: 2
Explanation: Because the 3rd row is incomplete, we return 2.

```java
class Solution {
    public int arrangeCoins(int n) {

        int left = 0, right = n;
        while (left <= right) {
            long mid = left + (right - left) / 2;
            long curr = mid * (mid + 1) / 2;
            if (curr == n) {
                return (int) mid;
            }
            if (curr < n) {
                left = (int) mid + 1;
            } else {
                right = (int) mid - 1;
            }
        }
        return right;
    }
}
```

9.Find First and last position of element in sorted array

Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given `target` value.

If `target` is not found in the array, return `[-1, -1]`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

> **Input:** nums = [5,7,7,8,8,10], target = 8
> **Output:** [3,4]

Example 2:

> **Input:** nums = [5,7,7,8,8,10], target = 6
> **Output:** [-1,-1]

Example 3:

> **Input:** nums = [], target = 0
> **Output:** [-1,-1]

```java
class Solution {
    public int[] searchRange(int[] nums, int target) {

        int[] ans = {-1, -1};
        if(nums.length==0){
            return ans ;
        }
        ans[0] = firstOccurence(nums,target);
        ans[1] = lastOccurence(nums,target);
        return ans ;

    }


    public int firstOccurence(int[] nums,int target){
        int start=0;
        int end=nums.length-1;
        int ans=-1;


        while(start<=end){
            int mid=(start+end)/2;

            if(nums[mid]==target){
                ans=mid;
                end=mid-1;

            }else if(target>nums[mid]){
                start=mid+1;
            }else{
                end=mid-1;
```

```
            if(nums[mid]==target){
                ans=mid;
                end=mid-1;

            }else if(target>nums[mid]){
                start=mid+1;
            }else{
                end=mid-1;
            }
        }
        return ans;
    }

    public int lastOccurence(int nums[],int target){
        int start = 0 ;
        int end = nums.length-1;
        int ans = -1 ;

        while(start<=end){
            int mid = (start + end )/2;
            if(nums[mid]==target){
                ans = mid ;
               start = mid+1 ;
            }else if ( target > nums[mid]){
                start = mid+1 ;
            }else{
                end = mid-1 ;
            }
        }
        return ans ;
    }
```

10.peak Index in a Mountain array

You are given an integer **mountain** array `arr` of length `n` where the values increase to a **peak element** and then decrease.

Return the index of the peak element.

Your task is to solve it in `O(log(n))` time complexity.

Example 1:

> **Input:** `arr = [0,1,0]`
>
> **Output:** 1

Example 2:

> **Input:** `arr = [0,2,1,0]`
>
> **Output:** 1

Example 3:

> **Input:** `arr = [0,10,5,2]`
>
> **Output:** 1

```java
class Solution {
    public int peakIndexInMountainArray(int[] arr) {
        int n= arr.length;
        int start=0;
        int end=n-1;

        while(start<=end){
            int mid=start+(end-start)/2;
            if(mid!=0 && mid!=n-1 && arr[mid]>arr[mid-1] && arr[mid]>arr[mid+1]){
                return mid;
            }else if(mid!=n-1 && arr[mid]<arr[mid+1]){
                start=mid+1;
            }else{
                end=mid-1;
            }

        }
        return -1;
    }
}
```

11.Find peak Element

A peak element is an element that is strictly greater than its neighbors.

Given a **0-indexed** integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in `O(log n)` time.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.
```

Example 2:

```
Input: nums = [1,2,1,3,5,6,4]
Output: 5
Explanation: Your function can return either index number 1 where the peak element is 2, or
index number 5 where the peak element is 6.
```

```java
class Solution {
    public int findPeakElement(int[] nums) {
        if(nums.length==1){
            return 0;
        }else if ( nums[0]>nums[1]){
            return 0;
        }else if ( nums[nums.length-1]>nums[nums.length-2]){
            return nums.length-1;
        }else {

            int start = 1 ;
            int end = nums.length-2;

             while(start<=end){
                int mid = ( start + end)/2;

                if(nums[mid]>nums[mid-1] && nums[mid]>nums[mid+1]){
                    return mid ;
                }else if ( nums[mid]<nums[mid+1]){
                    start = mid+1 ;
                }else{
                    end = mid-1 ;
                }
             }
           return -1 ;
        }
    }
}
```

12.Minimum Speed to arrive on time

https://leetcode.com/problems/minimum-speed-to-arrive-on-time/description/

13.Kth Missing Positive number

1539. Kth Missing Positive Number

Solved

Easy | Topics | Companies | Hint

Given an array `arr` of positive integers sorted in a **strictly increasing order**, and an integer `k`.

Return *the* `k^th` ***positive*** *integer that is* ***missing*** *from this array.*

Example 1:

> **Input:** arr = [2,3,4,7,11], k = 5
> **Output:** 9
> **Explanation:** The missing positive integers are [1,5,6,8,9,10,12,13,...]. The 5^{th} missing positive integer is 9.

Example 2:

> **Input:** arr = [1,2,3,4], k = 2
> **Output:** 6
> **Explanation:** The missing positive integers are [5,6,7,...]. The 2^{nd} missing positive integer is 6.

```java
class Solution {
    public int findKthPositive(int[] arr, int k) {
        int n=arr.length;
        int left=0;
        int right=n-1;
        while(left<=right){
            int mid=(left+right)/2;
            int missing=arr[mid]-(mid+1);

            if(missing<k){
                left=mid+1;
            }else{
                right=mid-1;
            }
        }
        return left+k;
    }
}
```

Saved

14.Search In Rotated sorted array

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` (`1 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`, *or* `-1` *if it is not in* `nums`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

Example 2:

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

Example 3:

```
Input: nums = [1], target = 0
Output: -1
```

```java
public int search(int[] nums, int target) {

    int start=0;
    int end=nums.length-1;

    while(start<=end){
        int mid =(start+end)/2;
        if(nums[mid]==target){
            return mid;

        }else if(nums[start]<=nums[mid]){
            if(target>=nums[start] && target<nums[mid]){
                end=mid-1;
            }else{
                start=mid+1;
            }

        }else{
                if(target>nums[mid] && target<=nums[end]){
                    start=mid+1;
                }else{
                    end=mid-1;
                }
           }

    }
     return -1;
}
```

15.Nth Digit

Given an integer `n`, return the n^{th} digit of the infinite integer sequence `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...]`.

Example 1:

> **Input:** n = 3
> **Output:** 3

Example 2:

> **Input:** n = 11
> **Output:** 0
> **Explanation:** The 11^{th} digit of the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ... is a 0, which is part of the number 10.

```java
class Solution {
    public int findNthDigit(int n) {
        long digitInNum = 1, start = 1, end = 9;

        while (n > digitInNum * end) {
            n -= digitInNum * end;
            digitInNum++;
            start *= 10;
            end *= 10;
        }

        long num = start + (n - 1) / digitInNum;
        String numStr = Long.toString(num);
        return numStr.charAt((int) ((n - 1) % digitInNum)) - '0';
    }
}
```

16.Find Minimum in Rotated sorted array ii

Suppose an array of length `n` sorted in ascending order is **rotated** between `1` and `n` times. For example, the array `nums = [0,1,4,4,5,6,7]` might become:

- `[4,5,6,7,0,1,4]` if it was rotated `4` times.
- `[0,1,4,4,5,6,7]` if it was rotated `7` times.

Notice that **rotating** an array `[a[0], a[1], a[2], ..., a[n-1]]` 1 time results in the array `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]`.

Given the sorted rotated array `nums` that may contain **duplicates**, return *the minimum element of this array*.

You must decrease the overall operation steps as much as possible.

Example 1:

> **Input:** nums = [1,3,5]
> **Output:** 1

Example 2:

> **Input:** nums = [2,2,2,0,1]
> **Output:** 0

```java
class Solution {
    public int findMin(int[] nums) {
        int n= nums.length;
        int mini=Integer.MAX_VALUE;
        for(int i=0;i<n;i++){
            mini=Math.min(mini,nums[i]);

        }
        return mini;
    }
}
```

17.Koko Eating banana

Koko loves to eat bananas. There are `n` piles of bananas, the `i^th` pile has `piles[i]` bananas. The guards have gone and will come back in `h` hours.

Koko can decide her bananas-per-hour eating speed of `k`. Each hour, she chooses some pile of bananas and eats `k` bananas from that pile. If the pile has less than `k` bananas, she eats all of them instead and will not eat any more bananas during this hour.

Koko likes to eat slowly but still wants to finish eating all the bananas before the guards return.

Return *the minimum integer* `k` *such that she can eat all the bananas within* `h` *hours.*

Example 1:

```
Input: piles = [3,6,7,11], h = 8
Output: 4
```

Example 2:

```
Input: piles = [30,11,23,4,20], h = 5
Output: 30
```

Example 3:

```
Input: piles = [30,11,23,4,20], h = 6
Output: 23
```

```java
class Solution {
    public int calculate(int piles[],int h){
        int hrs=0;
        for(int i=0;i<piles.length;i++){
            hrs +=Math.ceil((double)piles[i]/(double)h);
        }
        return hrs;
    }
    public int minEatingSpeed(int[] piles, int h) {
        int n=piles.length;
        int start=1;
        int end=Integer.MIN_VALUE;
        for(int i=0;i<n;i++){
            if(piles[i]>end){
                end=piles[i];
            }
        }
        while(start<=end){
            int mid=start+(end-start)/2;
            int time=calculate(piles,mid);
            if(time<=h){
                end=mid-1;
            }else{
                start=mid+1;
            }
        }
        return start;
    }
}
```

18.Find in Mountain array

You may recall that an array `arr` is a **mountain array** if and only if:

- `arr.length >= 3`
- There exists some `i` with `0 < i < arr.length - 1` such that:
  - `arr[0] < arr[1] < ... < arr[i - 1] < arr[i]`
  - `arr[i] > arr[i + 1] > ... > arr[arr.length - 1]`

Given a mountain array `mountainArr`, return the **minimum** `index` such that `mountainArr.get(index) == target`. If such an `index` does not exist, return `-1`.

You cannot access the mountain array directly. You may only access the array using a `MountainArray` interface:

- `MountainArray.get(k)` returns the element of the array at index `k` (0-indexed).
- `MountainArray.length()` returns the length of the array.

Submissions making more than `100` calls to `MountainArray.get` will be judged *Wrong Answer*. Also, any solutions that attempt to circumvent the judge will result in disqualification.

Example 1:

> **Input:** mountainArr = [1,2,3,4,5,3,1], target = 3
> **Output:** 2
> **Explanation:** 3 exists in the array, at index=2 and index=5. Return the minimum index, which is 2.

Example 2:

> **Input:** mountainArr = [0,1,2,4,2,1], target = 3
> **Output:** -1
> **Explanation:** 3 does not exist in `the array,` so we return -1.

```java
class Solution {
    public int findInMountainArray(int target, MountainArray mountainArr) {
        int length = mountainArr.length();

        // Find the index of the peak element in the mountain array.
        int peakIndex = findPeakIndex(1, length - 2, mountainArr);

        // Binary search for the target in the increasing part of the mountain array.
        int increasingIndex = binarySearch(0, peakIndex, target, mountainArr, false);
        if (mountainArr.get(increasingIndex) == target)
            return increasingIndex; // Target found in the increasing part.

        // Binary search for the target in the decreasing part of the mountain array.
        int decreasingIndex = binarySearch(peakIndex + 1, length - 1, target, mountainArr, true);
        if (mountainArr.get(decreasingIndex) == target)
            return decreasingIndex; // Target found in the decreasing part.

        return -1;  // Target not found in the mountain array.
    }

    private int findPeakIndex(int low, int high, MountainArray mountainArr) {
```

```java
private int findPeakIndex(int low, int high, MountainArray mountainArr) {
    while (low != high) {
        int mid = low + (high - low) / 2;
        if (mountainArr.get(mid) < mountainArr.get(mid + 1)) {
            low = mid + 1;
        } else {
            high = mid;
        }
    }
    return low;
}

private int binarySearch(int low, int high, int target, MountainArray mountainArr, boolean reversed)
```

```java
    private int binarySearch(int low, int high, int target, MountainArray mountainArr, boolean reversed) {
        while (low != high) {
            int mid = low + (high - low) / 2;
            if (reversed) {
                if (mountainArr.get(mid) > target)
                    low = mid + 1;
                else
                    high = mid;
            } else {
                if (mountainArr.get(mid) < target)
                    low = mid + 1;
                else
                    high = mid;
            }
        }
        return low;
    }
}
```

19.Find a Peak Element II

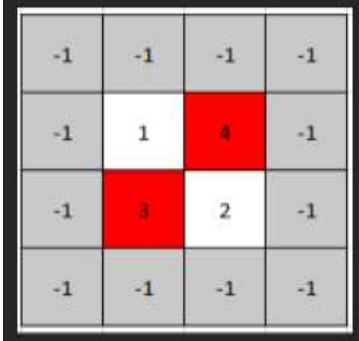
A **peak** element in a 2D grid is an element that is **strictly greater** than all of its **adjacent** neighbors to the left, right, top, and bottom.

Given a **0-indexed** `m x n` matrix `mat` where **no two adjacent cells are equal**, find **any** peak element `mat[i][j]` and return *the length 2 array* `[i,j]`.

You may assume that the entire matrix is surrounded by an **outer perimeter** with the value `-1` in each cell.

You must write an algorithm that runs in `O(m log(n))` or `O(n log(m))` time.

Example 1:




```
Input: mat = [[1,4],[3,2]]
Output: [0,1]
Explanation: Both 3 and 4 are peak elements so [1,0] and [0,1] are both acceptable answers.
```

Example 2:


```
Input: mat = [[10,20,15],[21,30,14],[7,16,32]]
Output: [1,1]
Explanation: Both 30 and 32 are peak elements so [1,1] and [2,2] are both acceptable answers.
```

```java
class Solution {
    public int maxEl(int[][] mat, int n, int m, int col) {
        int maxi = -1;
        int ind = -1;
        for (int i = 0; i < n; i++) {
            if (mat[i][col] > maxi) {
                maxi = mat[i][col];
                ind = i;
            }
        }
        return ind;
    }

    public int[] findPeakGrid(int[][] mat) {
        int n = mat.length;
        int m = mat[0].length;
        int low = 0;
        int high = m - 1;

        while (low <= high) {
            int mid = (low + high) / 2;
            int row = maxEl(mat, n, m, mid);
```

```java
    public int[] findPeakGrid(int[][] mat) {
        int n = mat.length;
        int m = mat[0].length;
        int low = 0;
        int high = m - 1;

        while (low <= high) {
            int mid = (low + high) / 2;
            int row = maxEl(mat, n, m, mid);
            int left = mid - 1 >= 0 ? mat[row][mid - 1] : -1;
            int right = mid + 1 < m ? mat[row][mid + 1] : -1;

            if (mat[row][mid] > left && mat[row][mid] > right)
                return new int[]{row, mid};
            else if (mat[row][mid] < left)
                high = mid - 1;
            else
                low = mid + 1;
        }
        return new int[]{-1, -1};
    }
}
```

20.Search in Rotated sorted array II

There is an integer array `nums` sorted in non-decreasing order (not necessarily with **distinct** values).

Before being passed to your function, `nums` is **rotated** at an unknown pivot index `k` (`0 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,4,4,5,6,6,7]` might be rotated at pivot index `5` and become `[4,5,6,6,7,0,1,2,4,4]`.

Given the array `nums` **after** the rotation and an integer `target`, return `true` *if* `target` *is in* `nums`, *or* `false` *if it is not in* `nums`.

You must decrease the overall operation steps as much as possible.

Example 1:

```
Input: nums = [2,5,6,0,0,1,2], target = 0
Output: true
```

Example 2:

```
Input: nums = [2,5,6,0,0,1,2], target = 3
Output: false
```

```java
public boolean search(int[] nums, int target) {
    int l = 0;
    int r = nums.length - 1;

    while (l <= r) {
        int mid = l + (r - l) / 2;
        if (nums[mid] == target) {
            return true;
        }

        // Handle duplicates
        if (nums[l] == nums[mid] && nums[r] == nums[mid]) {
            l++;
            r--;
        }
        // First half is in order
        else if (nums[l] <= nums[mid]) {
            // Target is in the first half
            if (nums[l] <= target && nums[mid] > target) {
                r = mid - 1;
            } else {
                l = mid + 1;
            }
        }
        // Second half is in order
        else {
            // Target is in the second half
            if (nums[mid] < target && nums[r] >= target) {
                l = mid + 1;
            } else {
                r = mid - 1;
            }
        }
    }
    return false;
}
```

21.Median of Two Sorted array

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be `O(log (m+n))`.

Example 1:

```
Input: nums1 = [1,3], nums2 = [2]
Output: 2.00000
Explanation: merged array = [1,2,3] and median is 2.
```

Example 2:

```
Input: nums1 = [1,2], nums2 = [3,4]
Output: 2.50000
Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.
```

```java
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int ans[] =merge(nums1,nums2);

        if(ans.length%2 ==0){
            double ans2=(double)(ans[ans.length/2] + ans[ans.length/2-1])/2;
            return ans2;
        }else{
            double ans2 =(double)(ans[ans.length/2]);

            return ans2;
        }
    }

    public int[] merge(int[] arr1,int[] arr2){
        int ans[] =new int[arr1.length+arr2.length];

        int p1=0;
        int p2=0;
        int p3=0;

        while(p1<arr1.length || p2<arr2.length){
            int val1 =p1<arr1.length ? arr1[p1]:Integer.MAX_VALUE;
            int val2=p2<arr2.length ? arr2[p2]:Integer.MAX_VALUE;

            if(val1<val2){
                ans[p3]=val1;
                p1++;
            }else{
                ans[p3] =val2;
                p2++;
            }
            p3++;
        }
        return ans;
    }
```

22.Find the row with maximum number of 1’s

Given a **binary** 2D array, where each row is **sorted**. Find the row with the maximum number of 1s.

Examples:

Input matrix : *0 1 1 1*
0 0 1 1
1 1 1 1
0 0 0 0
Output: *2*
Explanation: *Row = 2 has maximum number of 1s, that is 4.*

Input matrix : *0 0 1 1*
0 1 1 1
0 0 1 1
0 0 0 0
Output: *1*
Explanation: *Row = 1 has maximum number of 1s, that is 3.*

```
static int first(int arr[], int low, int high)
{
    int idx = -1;
    while (low <= high) {
        // Get the middle index
        int mid = low + (high - low) / 2;

        // If the element at mid is 1, then update mid
        // as starting index of 1s and search in the
        // left half
        if (arr[mid] == 1) {
            idx = mid;
            high = mid - 1;
        }
        // If the element at mid is 0, then search in
        // the right half
        else {
            low = mid + 1;
        }
    }
    return idx;
}

// Function that returns index of row
// with maximum number of 1s.
static int rowWithMax1s(int mat[][])
```

```
}

// Function that returns index of row
// with maximum number of 1s.
static int rowWithMax1s(int mat[][])
{
    // Initialize max values
    int max_row_index = 0, max = -1;

    // Traverse for each row and count number of
    // 1s by finding the index of first 1
    int i, index;
    for (i = 0; i < R; i++) {
        index = first(mat[i], 0, C - 1);
        if (index != -1 && C - index > max) {
            max = C - index;
            max_row_index = i;
        }
    }

    return max_row_index;
}
// Driver Code
```

23.Find the row with minimum number of 1's

Given a 2D **binary matrix**(1-based indexed) **mat** of dimensions **nxm** , determine the **row** that contains the **minimum number** of **1's**.
Note: The matrix contains only **1's** and **0's**. Also, if two or more rows contain the **minimum number** of **1's**, the answer is the **lowest** of those **indices**.

Examples :

Input: mat = [[1, 1, 1, 1], [1, 1, 0, 0], [0, 0, 1, 1], [1, 1, 1, 1]]
Output: 2
Explanation: Rows 2 and 3 contain the minimum number of 1's (2 each). Since, row 2 is less than row 3. Thus, the answer is 2.

Input: mat = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
Output: 1
Explanation: All the rows contain the same number of 1's (0 each). Among them, index 1 is the smallest, so the answer is 1.

```java
class Solution {
    int minRow(int mat[][]) {
        // code here
            // code here
        int min=Integer.MAX_VALUE;
        int ind=1;
        for(int i=0;i<mat.length;i++)
        {
            Arrays.sort(mat[i]);
            int lb=lowerBound(mat[i]);
            int noOfOnes=mat[i].length-lb;
            if(noOfOnes<min)
            {
                min=noOfOnes;
                ind=i+1;
            }
        }
        return ind;
    }
    int lowerBound(int[] arr)
    {
        int left=0;
        int right=arr.length-1;
        while(left<=right)
        {
            int mid=left+(right-left)/2;
            if(arr[mid]==1)
            {
                right=mid-1;
            }
            else
            {
                left=mid+1;
            }
        }
        return left;
    }
};
```