LeetCode Explore Problems Interview Contest Discuss Store September LeetCoding Challenge 2021

Description Solution Discuss (999+) Submissions C++

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:

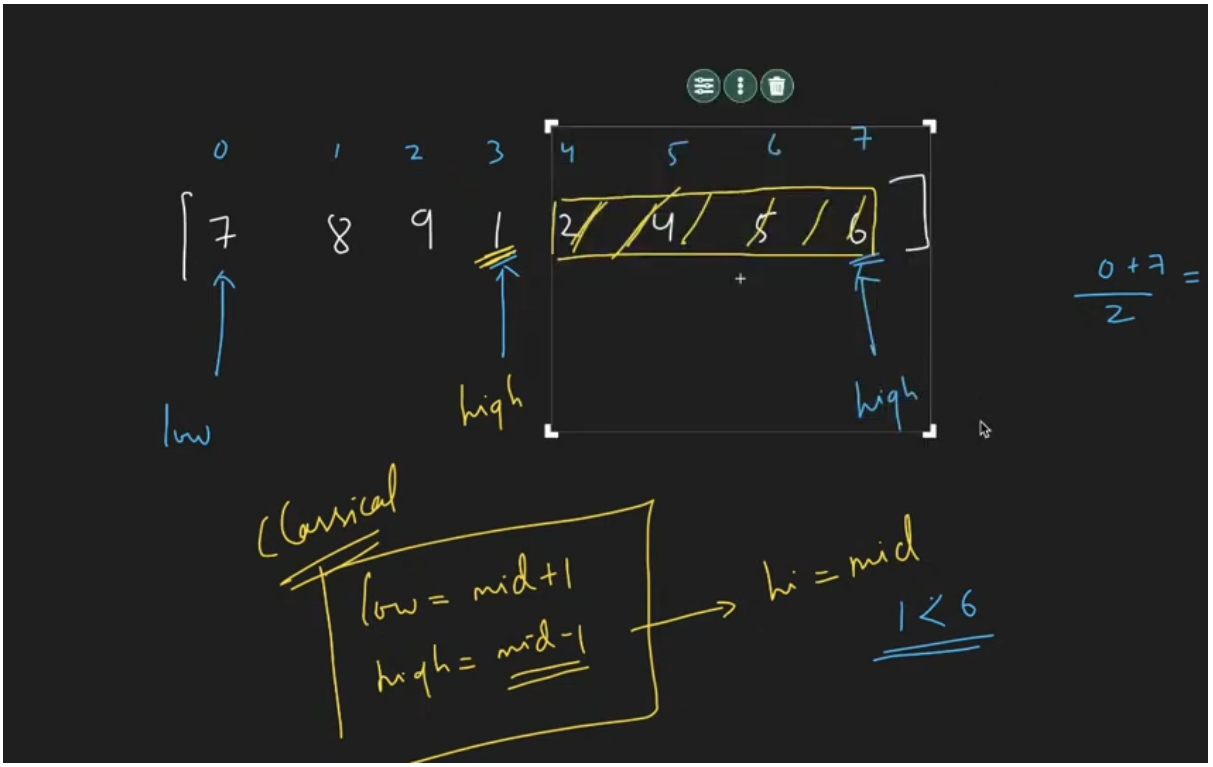
```
Input: nums = [11,13,15,17]
Output: 11
Explanation: The original array was [11,13,15,17] and it was rotated 4 times.
```

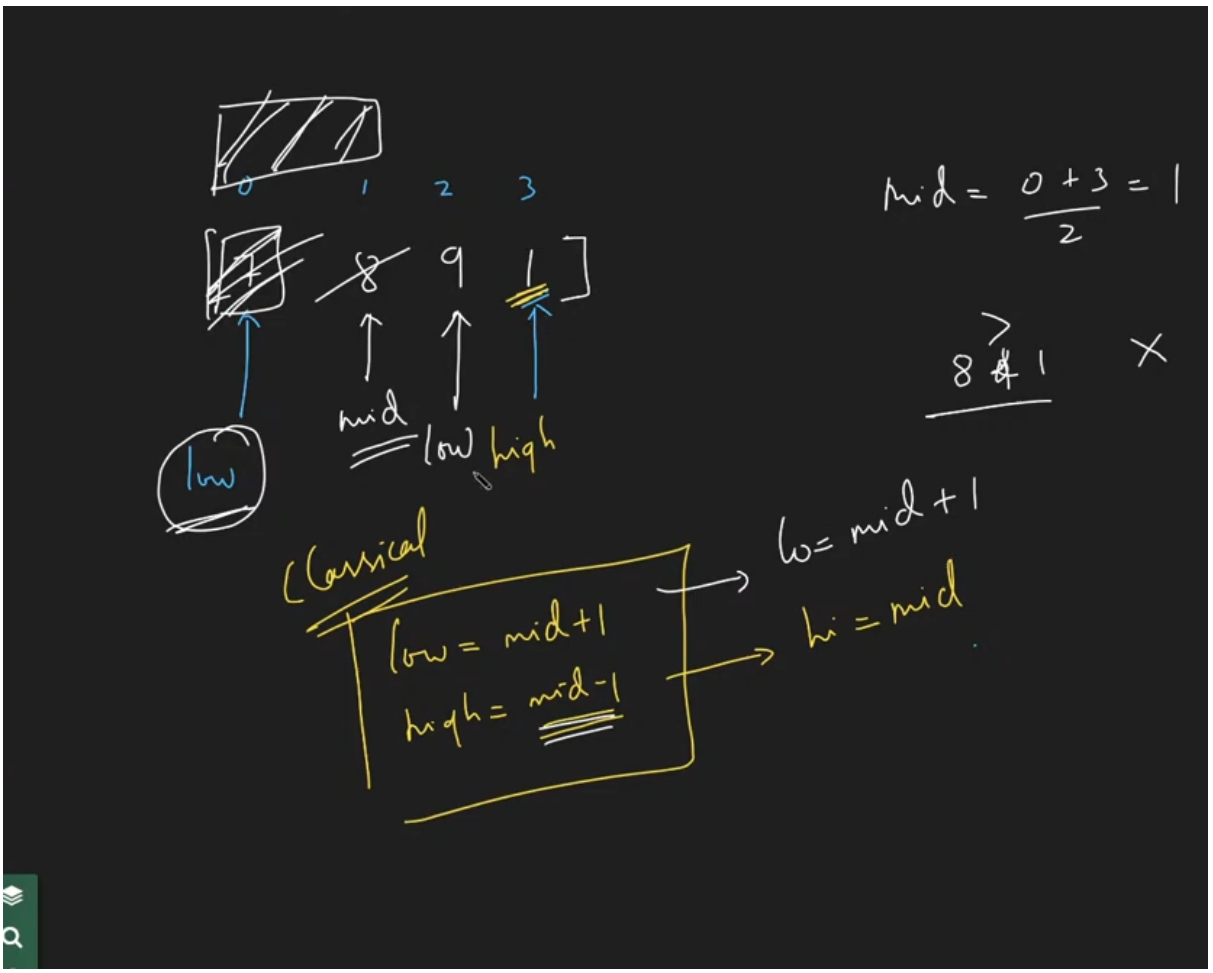
```
class
public
    in
nums)

nums.s

lo)/2;

nums[h

    }
};
```

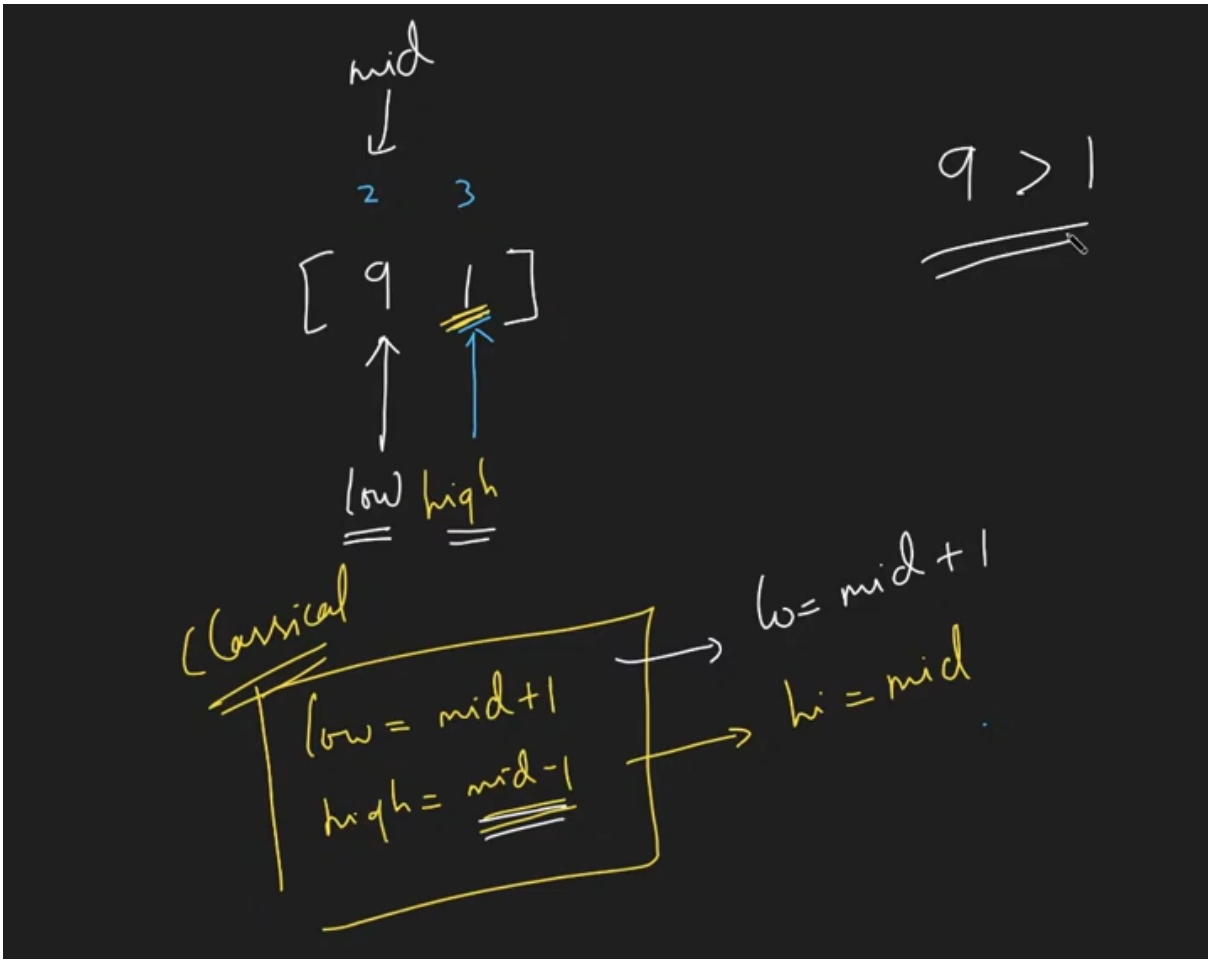
Your previous co

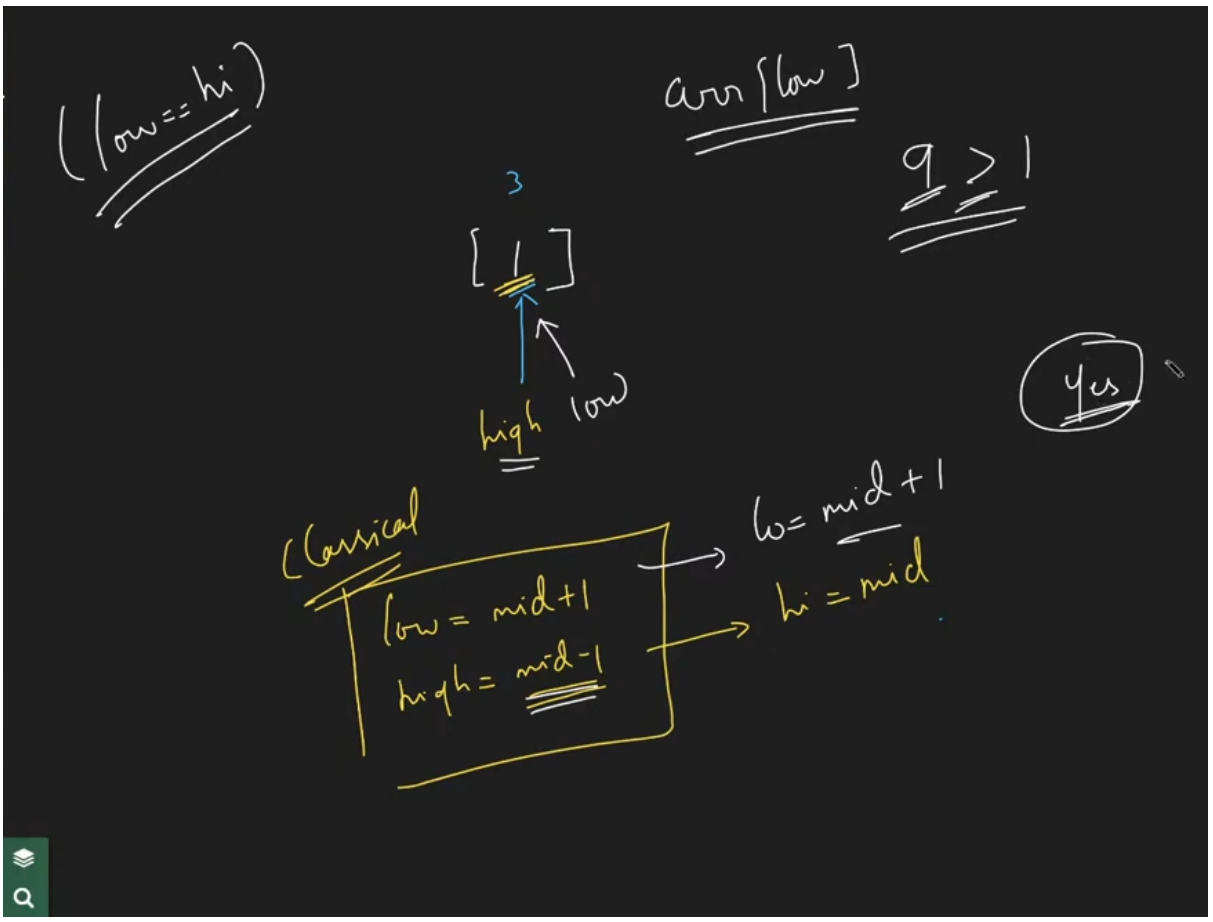
Console

Problems Pick One Prev 153/2004 Next Run Code




0 1 2 3
8 9 1
mid
low high
low
mid = 0+3/2 = 1
8 > 1 X
Classical
low = mid+1
high = mid-1
lo = mid+1
hi = mid


mid
2
3
[9 1]
low
high
9 > 1
Classical
low = mid+1
high = mid-1
lo = mid+1
hi = mid


(low == hi)
arr[low]
q >= 1
3
[1]
high
low
yes
Classical
low = mid+1
high = mid-1
lo = mid+1
hi = mid

```cpp
int findMin(int arr[], int n) {
    int low = 0, high = n - 1;
    while (low < high) {
        int mid = (low + high) / 2;
        if (arr[mid] < arr[high]) {
            high = mid;
        }
        else {
            low = mid + 1;
        }
    }
    return arr[low];
}
signed main() {
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
    int n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cout << findMin(arr, n) << endl;

}
```

154. Find Minimum in Rotated Sorted Array II

Hard 1909 297 Add to List Share

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]]`.

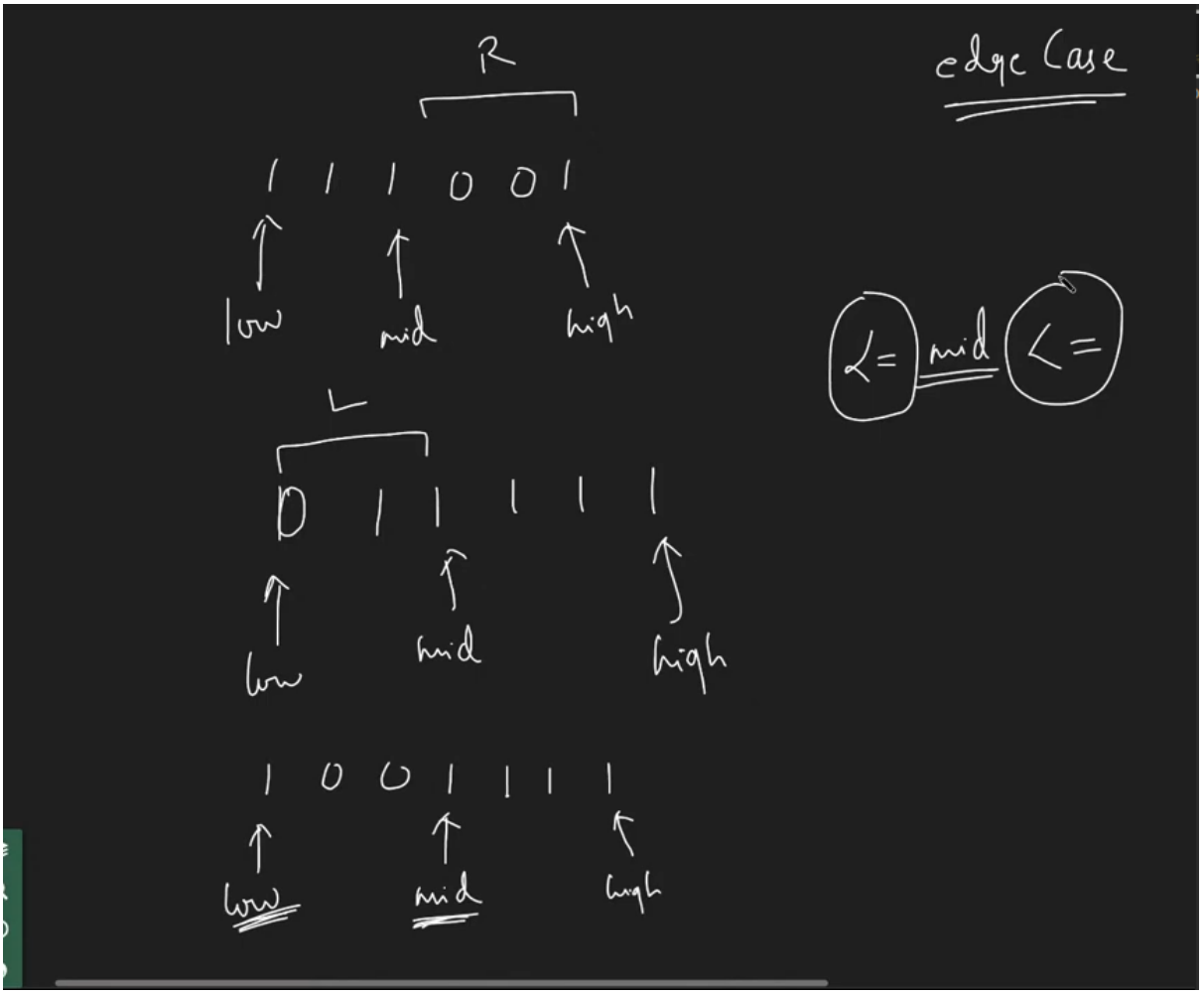
Given the sorted rotated array `nums` that may contain **duplicates**, return *the minimum element of this array*.

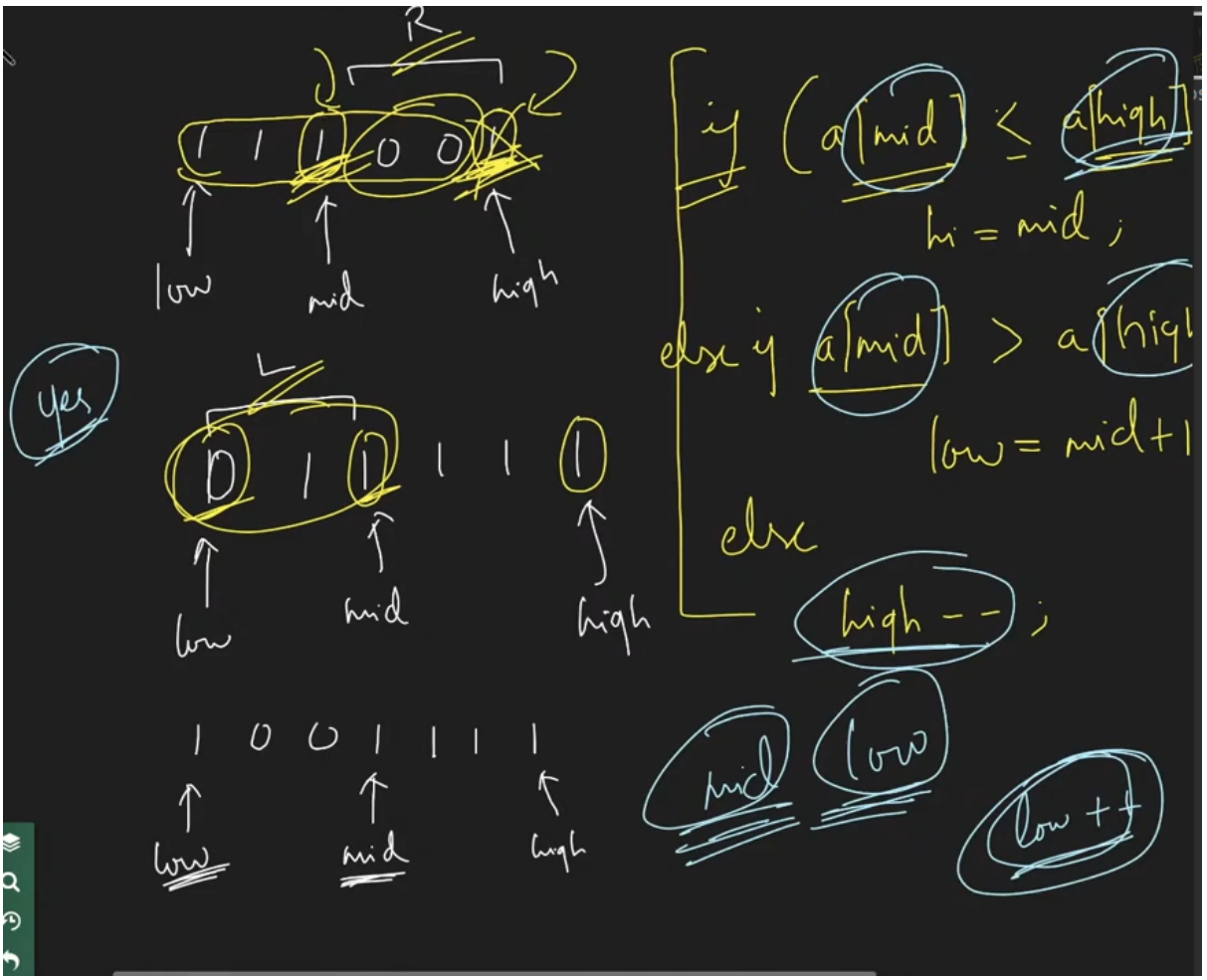
You must decrease the overall operation steps as much as possible.

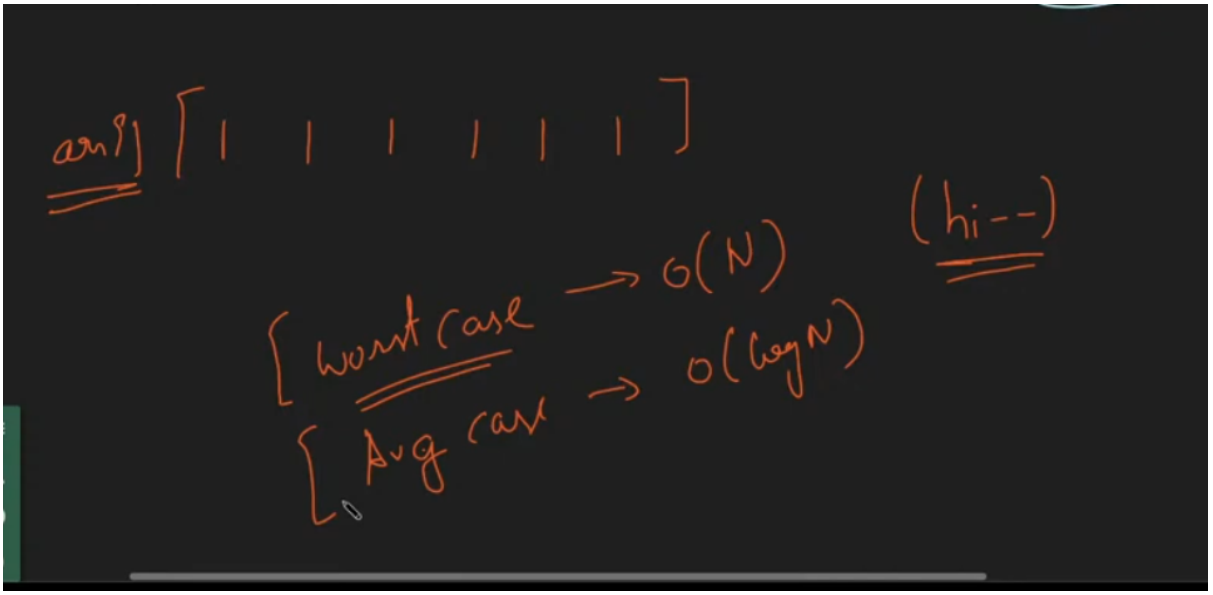
Example 1:

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

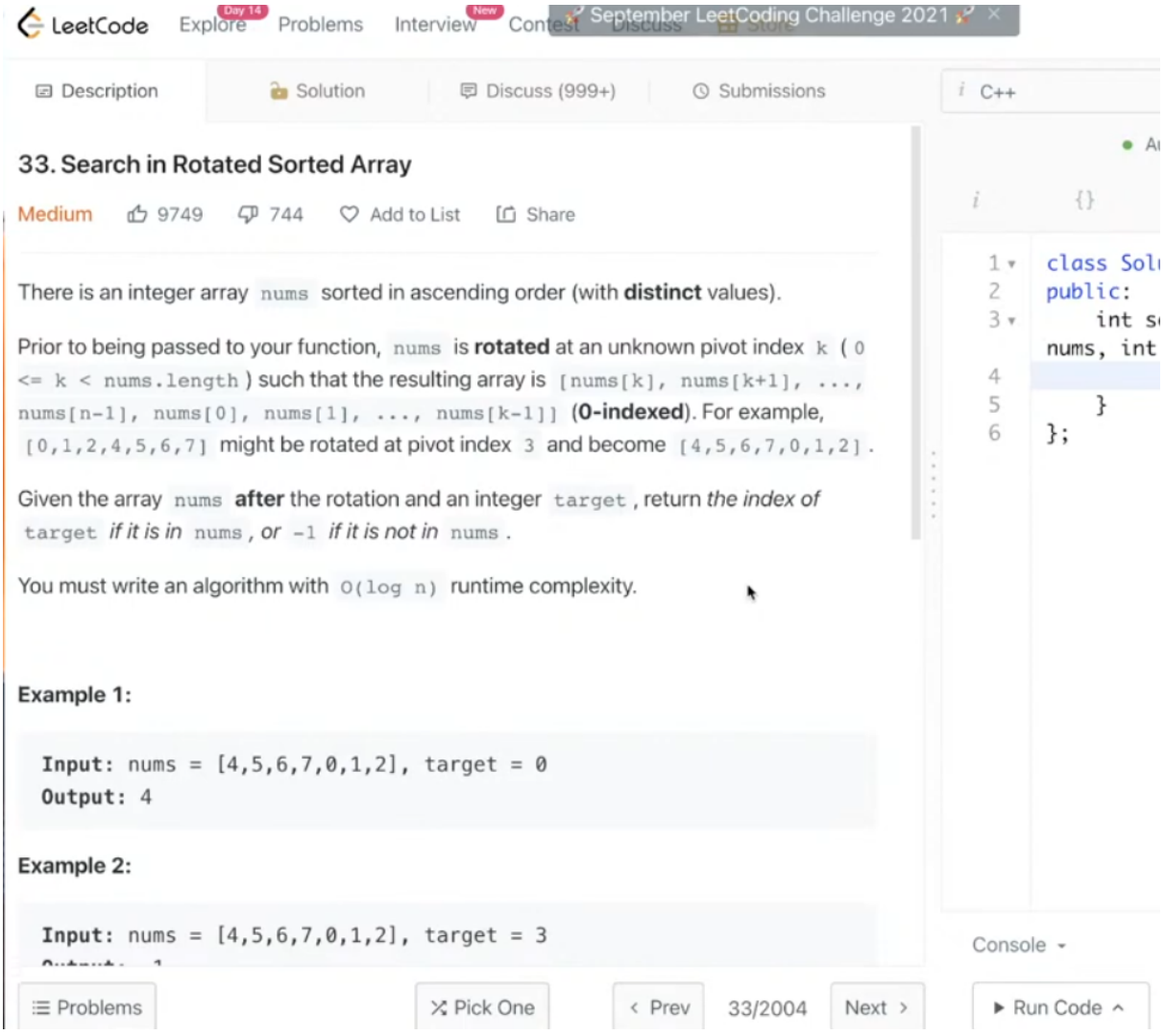
Example 2:

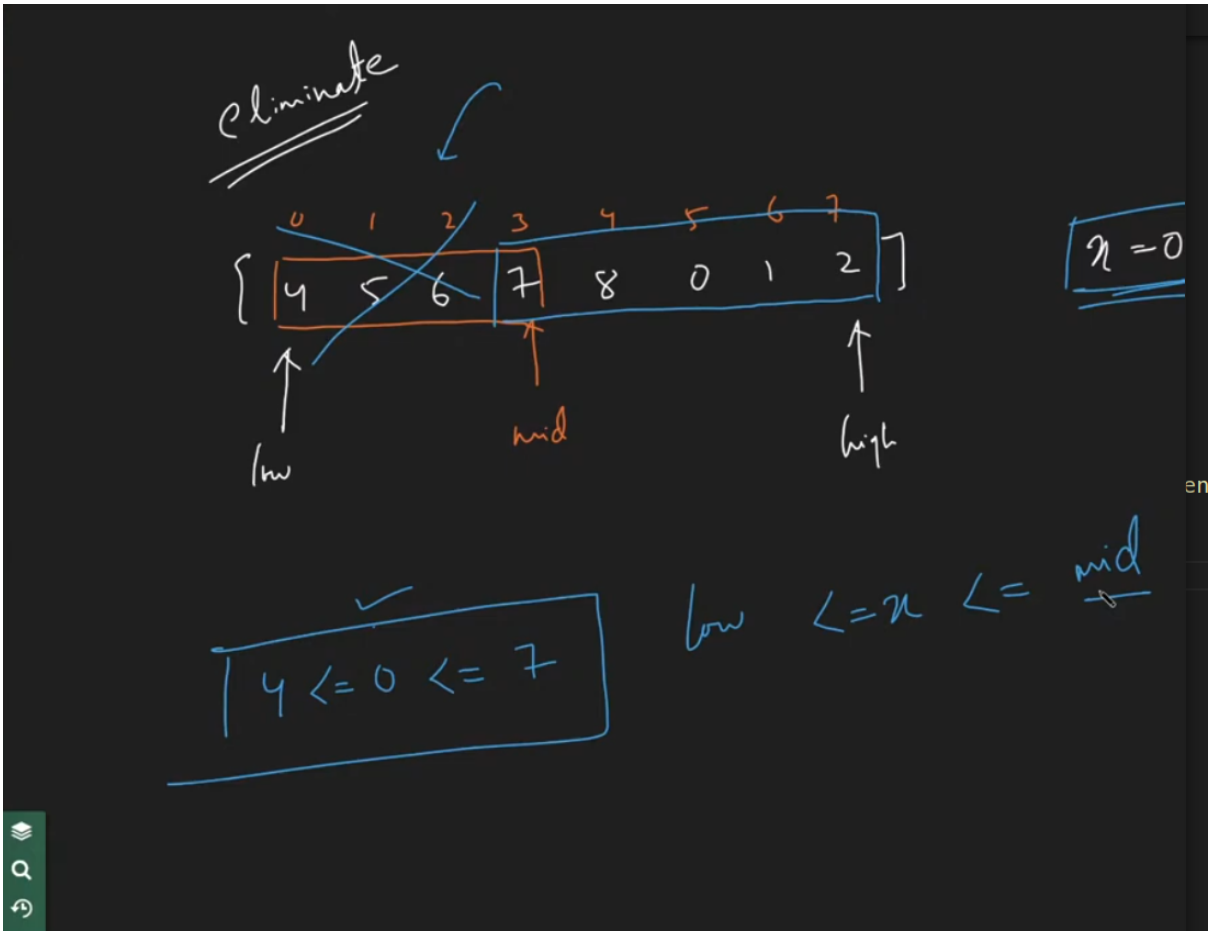

R
edge Case
1 1 1 0 0 1
low
mid
high
L
0 1 1 1 1 1
low
mid
high
1 0 0 1 1 1 1
low
mid
high
<=
mid
<=

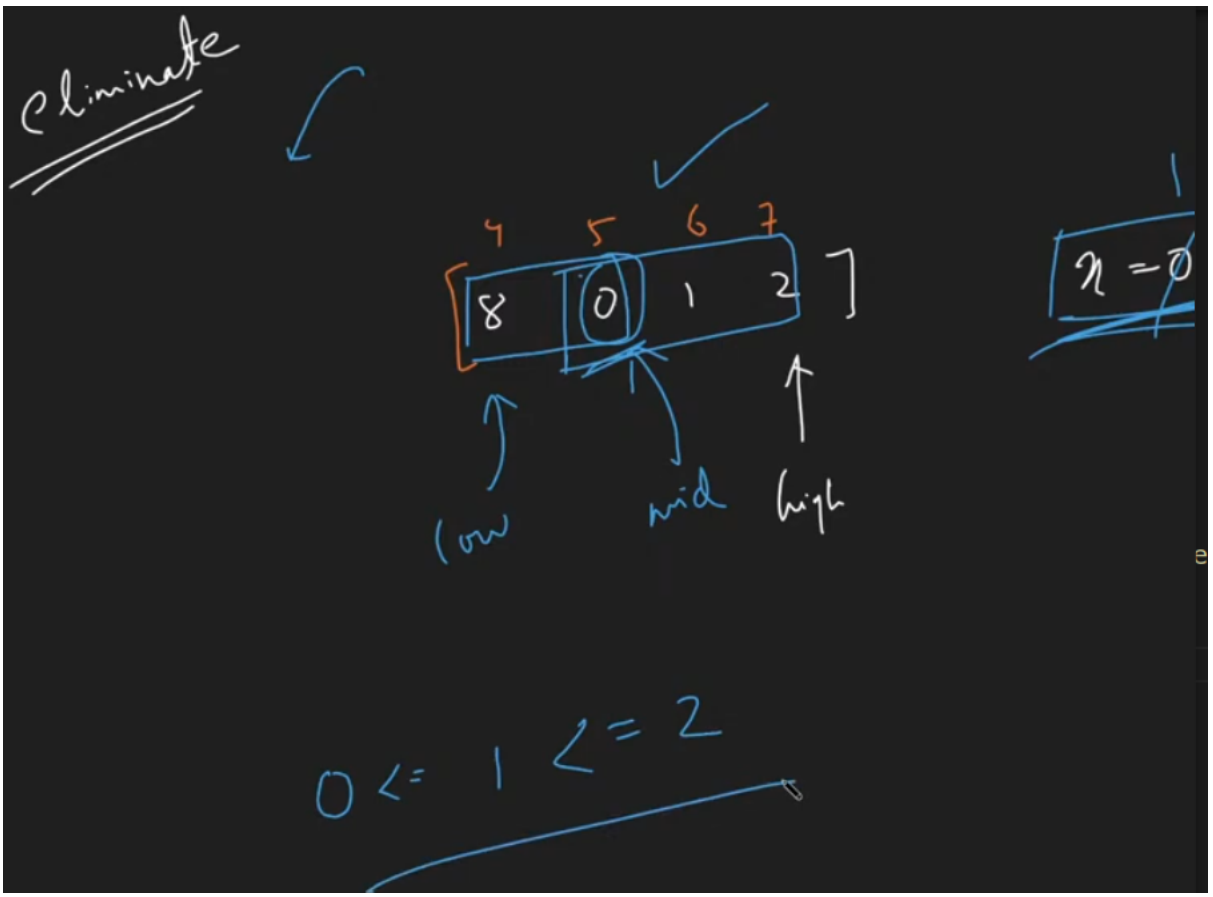

R
low
mid
high
yes
L
low
mid
high
low
mid
high
if (a[mid] ≤ a[high])
hi = mid ;
else if a[mid] > a[high]
low = mid+1
else
high --;
mid
low
low ++


arr[] [| | | | | |]
[worst case → O(N)
[Avg case → O(logN)
(hi--)

```java
// o(n) for the worst case ...
int findMin2(int arr[], int n) {
    int low = 0, high = n - 1;
    while (low < high) {
        int mid = (low + high) / 2;
        if (arr[mid] < arr[high]) {
            high = mid;
        }
        else if (arr[mid] > arr[high]) {
            low = mid + 1;
        }
        else {
            high--;
        }
    }
    return arr[high];
}
```




eliminate
0 1 2 3 4 5 6 7
4 5 6 7 8 0 1 2
low
mid
high
x = 0
4 <= 0 <= 7
low <= x <= mid


eliminate
4 5 6 7
8 0 1 2
low
mid
high
x = 0
0 <= 1 <= 2

```
int findEl3(int arr[], int n, int x) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == x) return mid;

        // if the left part is sorted
        if (arr[low] <= arr[mid]) {
            // either it lies on the left part or not
            if (x >= arr[low] && x <= arr[mid]) {
                high = mid - 1;
            }
            else {
                low = mid + 1;
            }
        }
        // or the right part is sorted
        else {
            // either it lies on the right part or not
            if (x >= arr[mid] && x <= arr[high]) {
                low = mid + 1;
            }
            else {
                high = mid - 1;
            }
        }
    }
    return -1;
}
```

162. Find Peak Element

Medium 3979 3032 Add to List Share

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

Given an 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] = -∞`.

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
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Consc

LeetCode Explore Day 14 Problems Interview New Contest Discuss Store

September LeetCoding Challenge 2021

Description | Solution | Discuss (999+) | Submissions

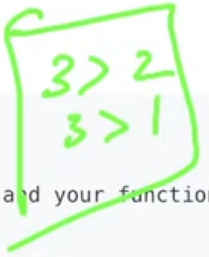
C++

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

Given an 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] = -∞`.

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



Example 1:

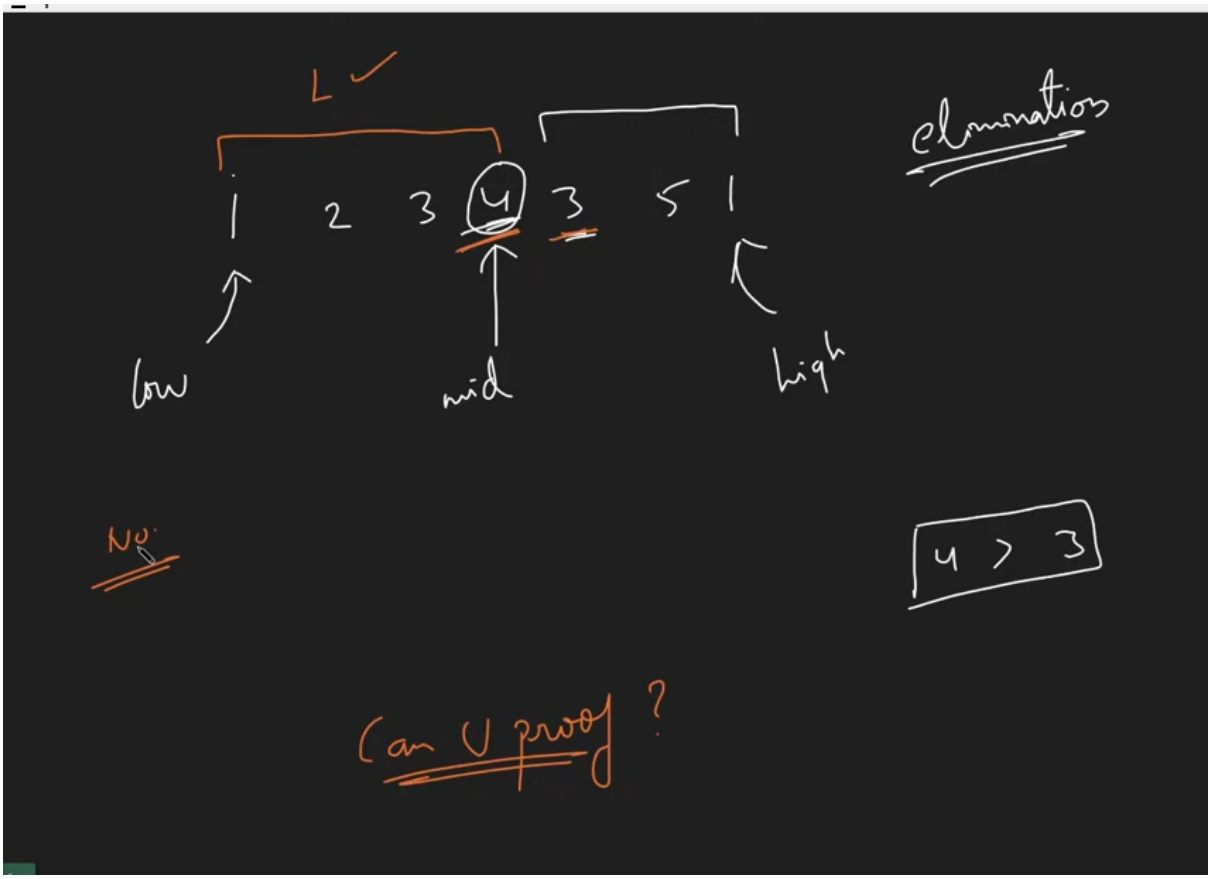
```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return
the index number 2.
```

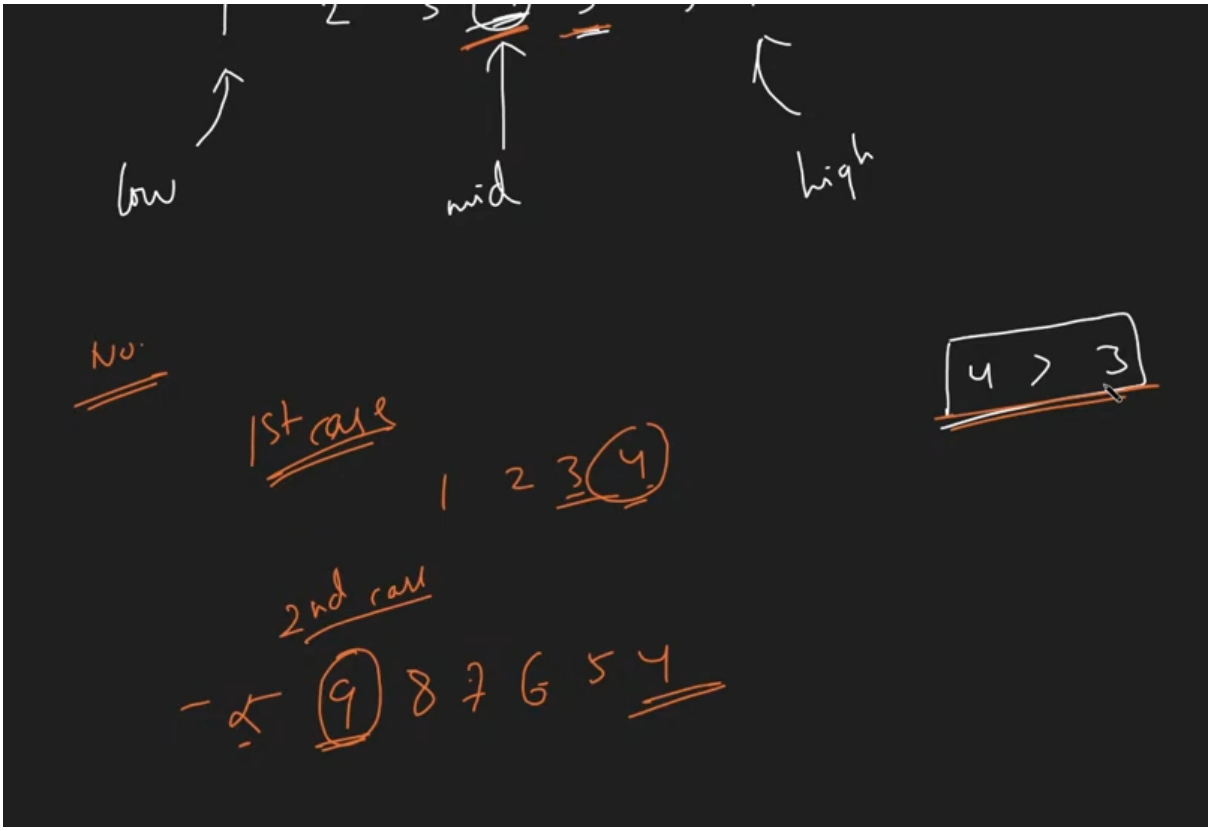
Example 2:

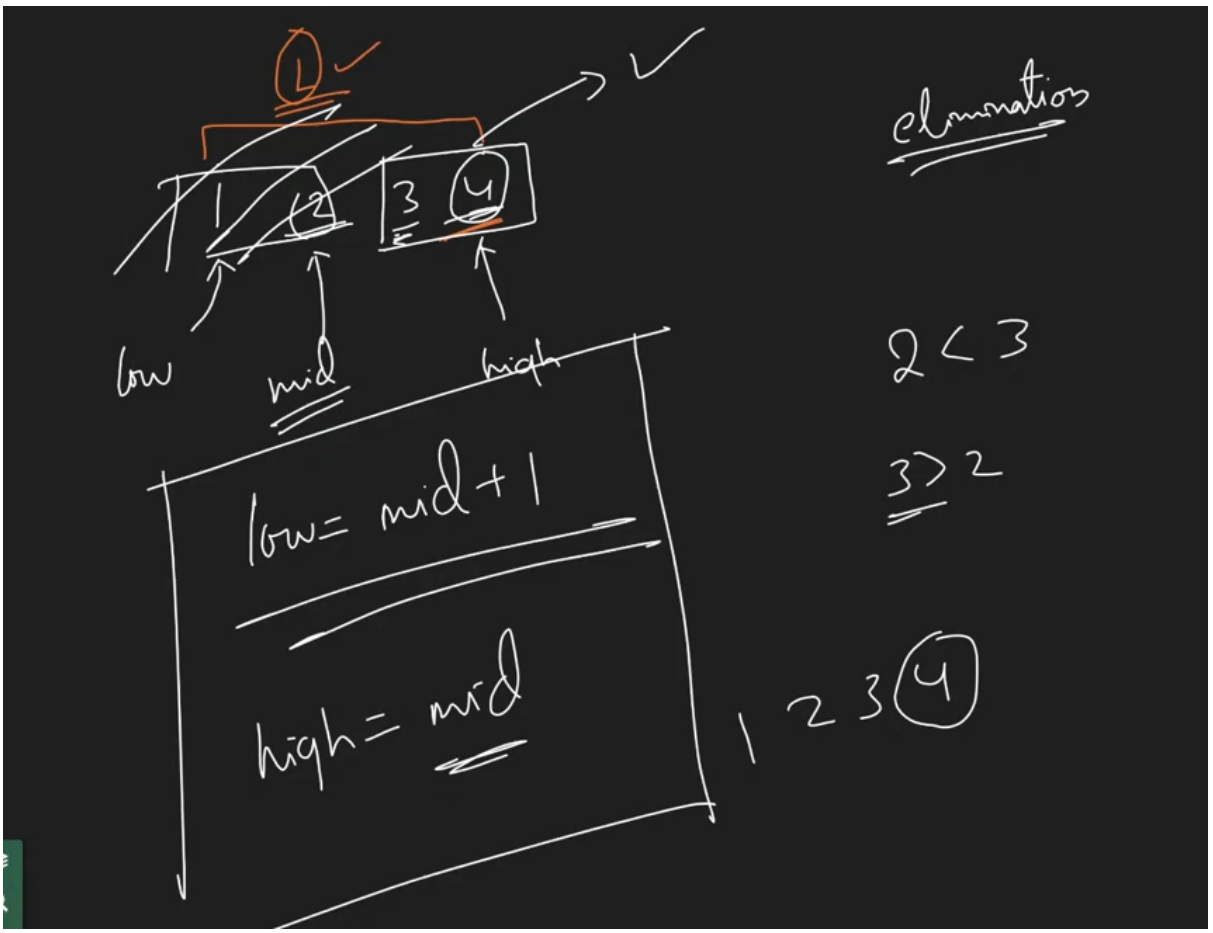

```
Input: nums = [1,2,1,3,5,6,4]
Output: 5
Explanation: Your function can return either index number 1 where
the peak element is 2, or index number 5 where the peak element
is 6.
```

```
1  class Solut
2  public:
3      int
   findPeakEle
   nums) {
4          int
   nums.size()
5          wh
6          {
7
   high) >> 1;
8
   nums[mid+1]
9
10
11
   1;
12         }
13         ret
14     }
15 };
```

Console


L
1 2 3 4 3 5 1
low
mid
high
elimination
No
4 > 3
Can U proof ?


low
mid
high
No
4 > 3
1st call
1 2 3 4
2nd call
9 8 7 6 5 4




```java
int findPeak(int arr[], int n) {
    int low = 0, high = n - 1;
    while (low < high) {
        int mid = (low + high) / 2;
        if (arr[mid] > arr[mid + 1]) {
            high = mid;
        }
        else {
            low = mid + 1;
        }
    }
    return arr[low];
}
```

Description | Solution | Discuss (999+) | Submissions

209. Minimum Size Subarray Sum

Medium 4630 154 Add to List Share

Given an array of positive integers `nums` and a positive integer `target`, return the minimal length of a **contiguous subarray** $[nums_l, nums_{l+1}, \ldots, nums_{r-1}, nums_r]$ of which the sum is greater than or equal to `target`. If there is no such subarray, return `0` instead.

Example 1:

```
Input: target = 7, nums = [2,3,1,2,4,3]
Output: 2
Explanation: The subarray [4,3] has the minimal length under the problem constraint.
```

Example 2:

```
Input: target = 4, nums = [1,4,4]
Output: 1
```

Example 3:

```
Input: target = 11, nums = [1,1,1,1,1,1,1,1]
Output: 0
```

Constraints:

- $1 <= target <= 10^9$
- $1 <= nums.length <= 10^5$
- $1 <= nums[i] <= 10^5$

Follow up: If you have figured out the $O(n)$ solution, try coding another solution of which the time complexity is $O(n \log(n))$.

Accepted 403,572 Submissions 982,107

Seen this question in a real interview before? Yes No

Companies

Binary search on answers

Sum of subarray >= t

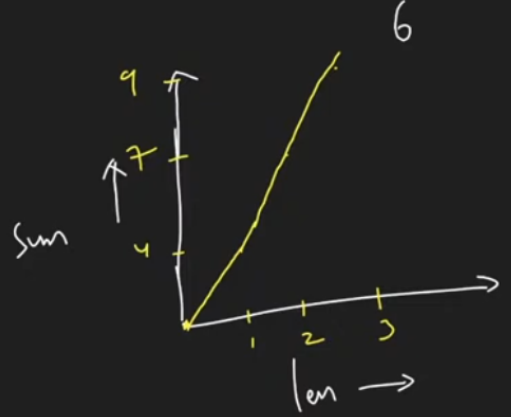
Set Blend Mode

n=6 [2 3 1 2 4 3] , target = 7

Min length

Min length

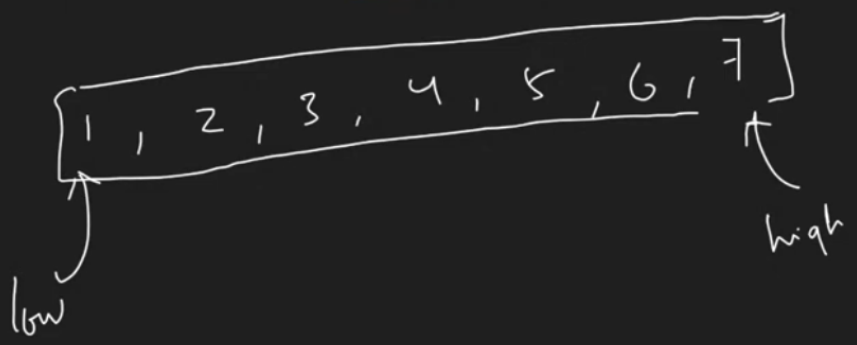
:



6

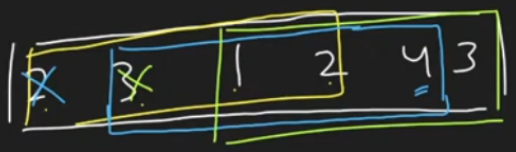
Monotonic increasing Fn

n=6 [2 3 1 2 4 3] , target = 7



Max subarray sum of size k

Q/ Max Subarray Sum of Size K



k = 4

Sliding Window

Sum = 8 + 4 − 2

= 10 + 3 − 3

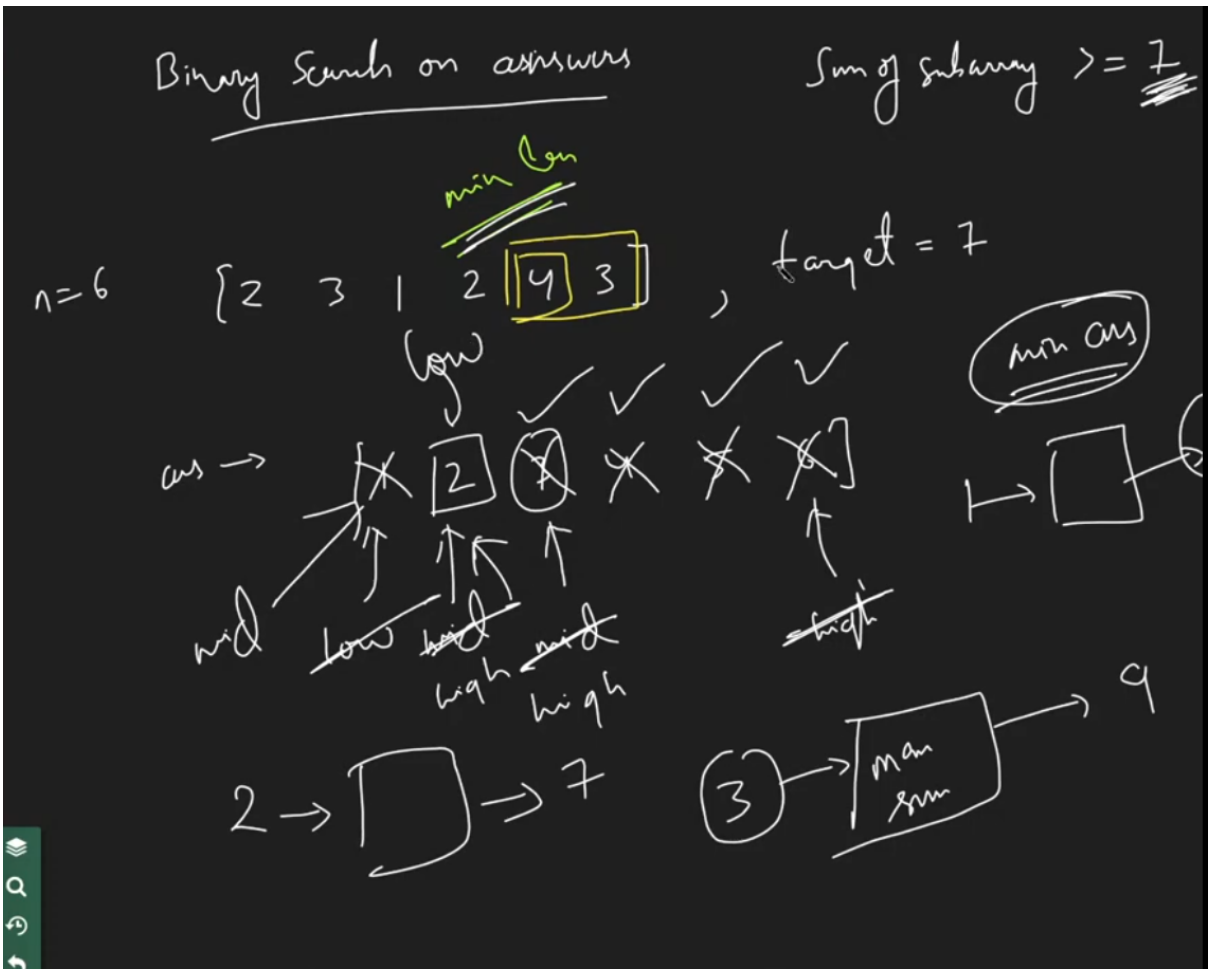
= 10

10

len = 4

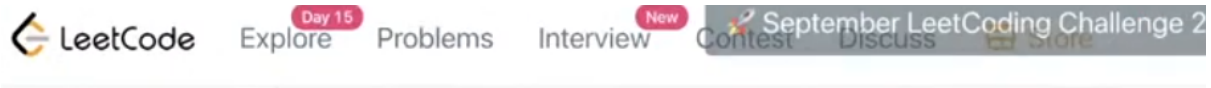
Sm = 10

target = 7




```cpp
// check if there exists a subarray of size k
// which has a sum >= target
bool blackBox(int arr[], int n, int target, int k) {
    // first find the first K size subarray sum;
    int sum = 0;
    for(int i = 0;i<k;i++) sum += arr[i];
    int maxi = sum;
    int l = 0, r = k - 1;
    // move the slider
    while( r != n-1 ) {
        sum -= arr[l];
        l++;

        sum += arr[r];
        r++;

        maxi = max(maxi, sum);
    }
    return maxi >= target;
}
```

```cpp
// find the min lenght of subarray such that sum >= target
int findMinLength(int arr[], int n, int target) {
    int low = 1, high = n;
    bool ansPossible = false;
    while(low < high) {
        int mid = (low + high) / 2;
        if(blackBox(arr, n, target, mid) == true) {
            ansPossible = true
            hi = mid;
        }
        else {
            low = mid + 1;
        }
    }
    if(ansPossible == true) return low;
    return 0;
}
```



Description | Solution | Discuss (420) | Submissions

1283. Find the Smallest Divisor Given a Threshold

Medium 949 137 Add to List Share

Given an array of integers `nums` and an integer `threshold`, we will choose a positive integer `divisor`, divide all the array by it, and sum the division's result. Find the **smallest** `divisor` such that the result mentioned above is less than or equal to `threshold`.

Each result of the division is rounded to the nearest integer greater than or equal to that element. (For example: `7/3 = 3` and `10/2 = 5`).

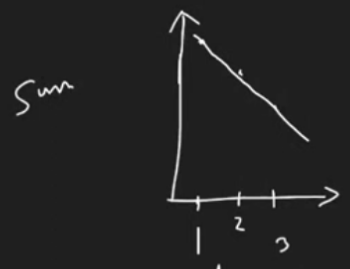
It is guaranteed that there will be an answer.

Example 1:

```
Input: nums = [1,2,5,9], threshold = 6
Output: 5
Explanation: We can get a sum to 17 (1+2+5+9) if the divisor
is 1.
```

(Binary Search on Answer)

divisor → ??



divisor ↑↑ Sum ↓↓

Sum <= threshold

[1 max(A[i])]

divisor

min Sum = n

divisor

max Sum = Σ a[i]

1/9 2/9 5/9 9/9

⇓ ⇓ ⇓ ⇓

1 1 1+1 = 4 = n

1/10 2/10 5/10 9/10

1 1 1 1

= 4

(Binary Seach on Answer)

[1 2 5 9] | Threshold = 6

[1 5 9]

low mid high

→ [5] → Sum =

1 + 1 + 1 + 2

(Binary Seach on Answer) Min Divisor

ans = 5

[1 2 5 9] | Threshold = 6

mid

[1 2 4] [3 4]

low high

→ [5] → Sum =

→ [2] → Sum = 10

1 + 1 + 3 + 5

1 + 1 + 1 + 2

```
int findSumAfterDiv(int arr[], int n, int div) {
    int sum = 0;
    for(int i = 0;i<n;i++) {
        sum += (arr[i]/div);
        // for ceiling add one more
        // 5 / 2 , we will add 2 on line 270, but wee
        // need to add 3,. hence add 1 more
        if(arr[i] % div != 0) {
            sum += 1;
        }
    }
    return sum;
}
```

```
int findMinDivisor(int arr[], int n, int thres) {
    int low = 1, high = *max_element(arr, arr+n);
    int ans = high;
    while(low <= high) {
        int mid = (low + high) / 2;
        // mid is giving <= thres
        // but am looking for even smaller, hence do
        // a search on the left
        if(findSumAfterDiv(arr, n, mid) <= thres) {
            ans = mid;
            high = mid - 1;
        }
        else {
            low = mid + 1;
        }
    }
    return ans;
}
```

410. Split Array Largest Sum

Hard 3359 107 Add to List Share

Given an array `nums` which consists of non-negative integers and an integer `m`, you can split the array into `m` non-empty continuous subarrays.

Write an algorithm to minimize the largest sum among these `m` subarrays.

Example 1:

```
Input: nums = [7,2,5,10,8], m = 2
Output: 18
Explanation:
There are four ways to split nums into two subarrays.
The best way is to split it into [7,2,5] and [10,8],
where the largest sum among the two subarrays is only 18.
```

Example 2:

You have one chocolate bar that consists of some chunks. Each chunk has its own sweetness given by the array `sweetness`.

You want to share the chocolate with your `k` friends so you start cutting the chocolate bar into `k + 1` pieces using `k` cuts, each piece consists of some **consecutive** chunks.

Being generous, you will eat the piece with the **minimum total sweetness** and give the other pieces to your friends.

Find the **maximum total sweetness** of the piece you can get by cutting the chocolate bar optimally.

Example 1:

```
Input: sweetness = [1,2,3,4,5,6,7,8,9], k = 5
Output: 6
Explanation: You can divide the chocolate to [1,2,3], [4,5], [6], [7], [8], [9]
```

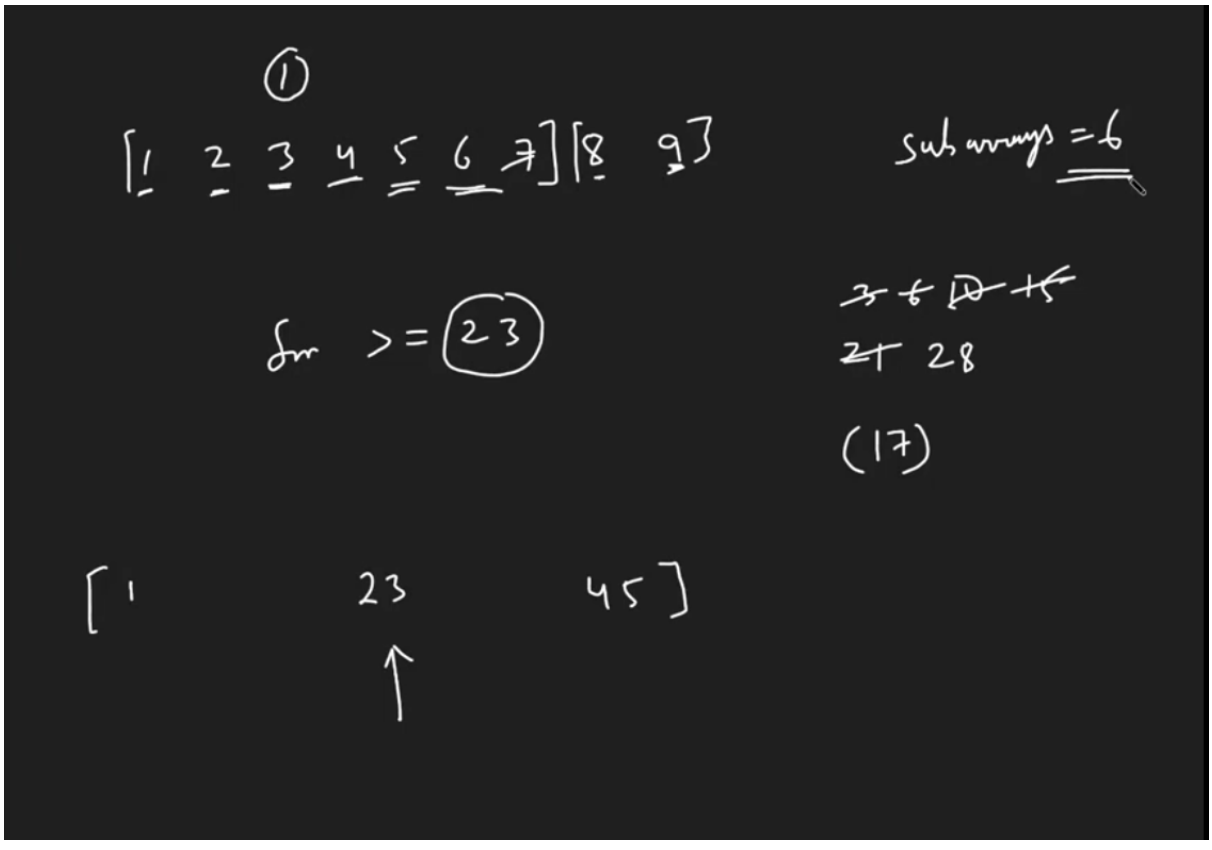
k = 5

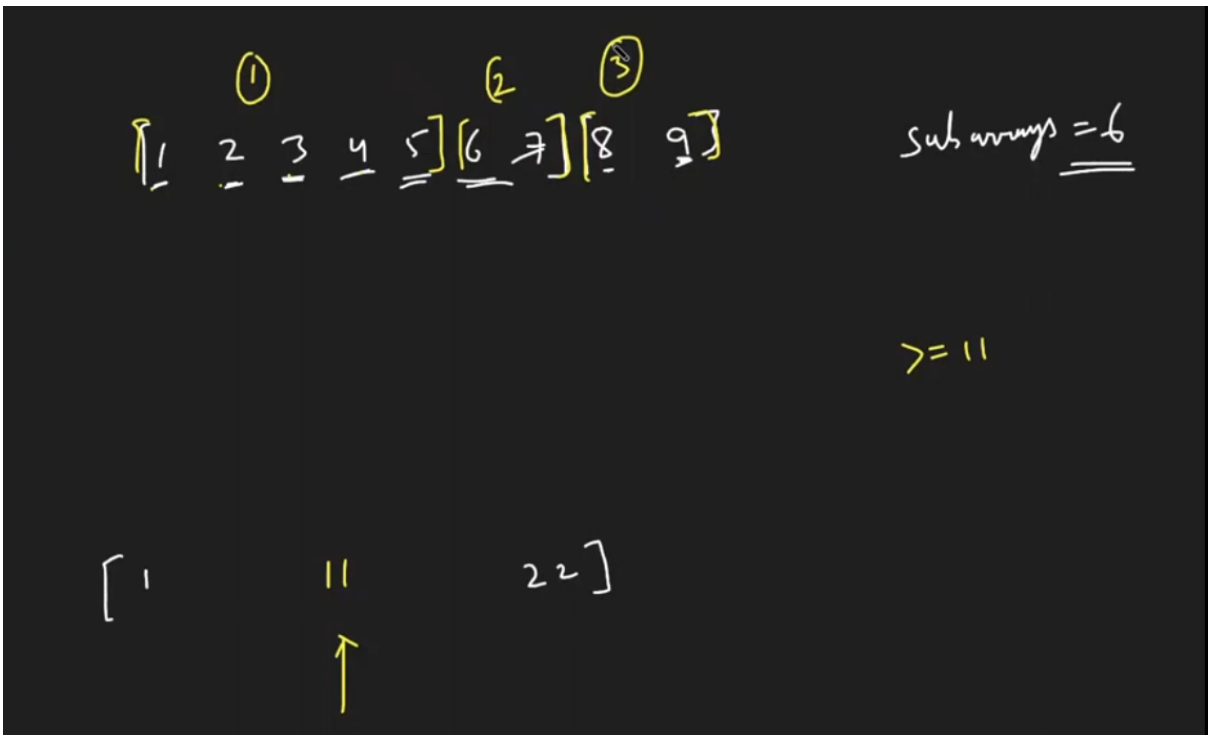
⇓

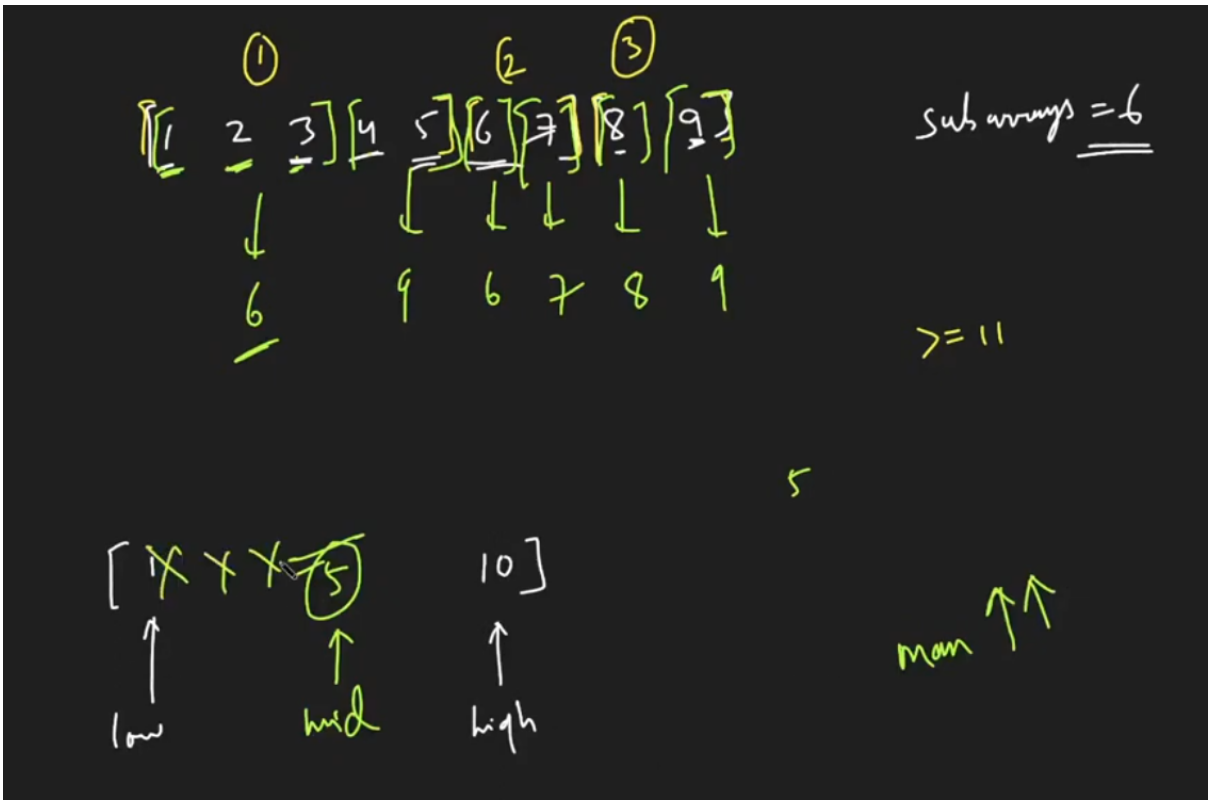
6 subarrays

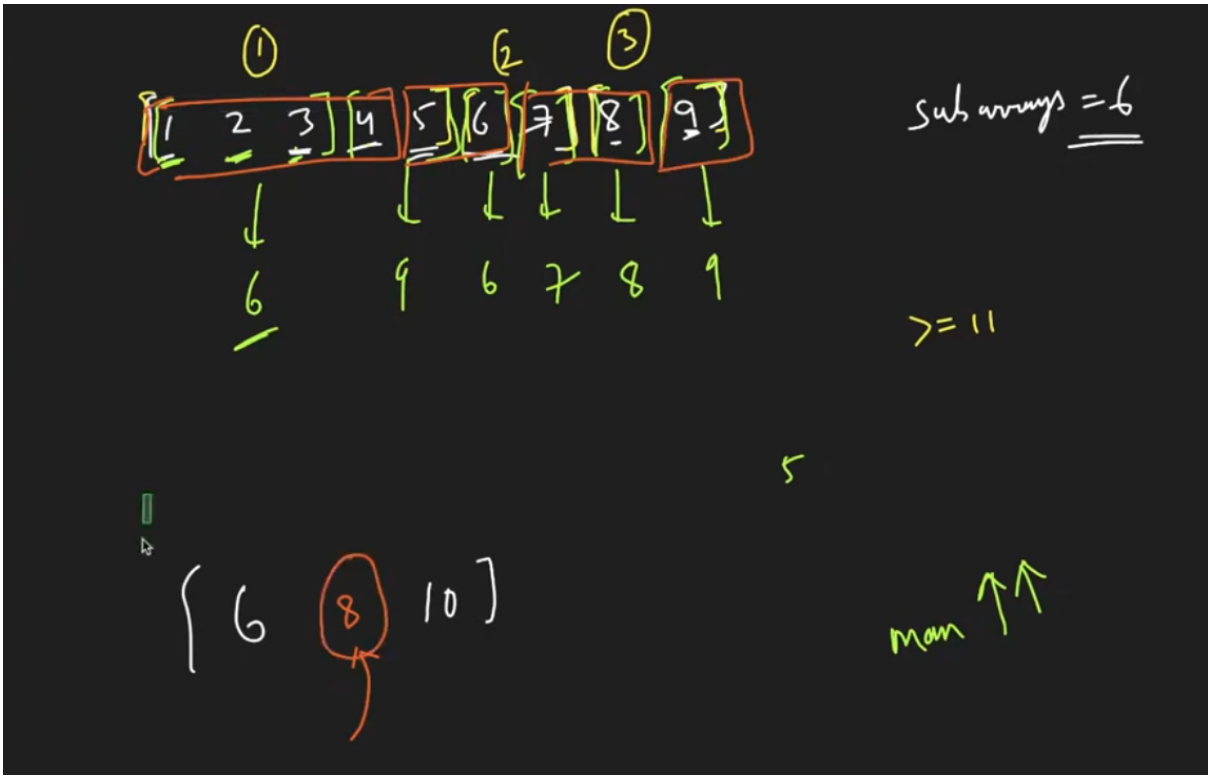
max(sum)

ans → [1 , sum]




1
2
3
[1 2 3 4 5] [6 7] [8 9]
sub arrays = 6
>= 11
[1 11 22]


1
2
3
[1 2 3] [4 5] [6] [7] [8] [9]
sub arrays = 6
6
9 6 7 8 9
>= 11
5
[X X X 5 10]
low
mid
high
mom


1
2
3
1 2 3 4 5 6 7 8 9
6 9 6 7 8 9
sub arrays = 6
>= 11
5
[6 8 10]
max

```
bool canGetMoreThanKSubarrays(int arr[], int n, int limit, int k) {
    int cnt = 0;
    int sum = 0;
    for(int i = 0;i<n;i++) {
        sum += arr[i];

        if(sum  >= limit) {
            cnt++;
            sum = 0;
        }
    }

    return cnt >= k;
}

int findMaxChocolates(int arr[], int n, int k) {
    int low = 1, high = 0;
    for(int i = 0;i<n;i++) {
        high += arr[i];
    }
    int ans = 1;

    while(low <= high) {
        int mid = (low + high) / 2;
        if(canGetMoreThanKSubarrays(arr, n, mid, k) == true) {
            ans = mid;
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }
    return ans;
}
```

Farmer John has built a new long barn, with N (2 <= N <= 100,000) stalls. The stalls are located along a straight line at positions x1,...,xN (0 <= xi <= 1,000,000,000).

His C (2 <= C <= N) cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, FJ wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

Input

t – the number of test cases, then *t* test cases follows.
* Line 1: Two space-separated integers: N and C
* Lines 2..N+1: Line i+1 contains an integer stall location, xi

Output

For each test case output one integer: the largest minimum distance.

Example

Input:

```
1
5 3
1
2
8
4
9
```

Output:

```
3
```