EXPERIMENT LIST FOR PROGRAMMING ABILITY AND LOGIC BUILDING – 21

NAME : IKSHU GOEL

BATCH : 2CSE5

ROLL NUMBER : 2410030774

WEEK : 19/1/26 - 25/1/26

LECTURE : 2

EXPERIMENT 1

Given an array **arr**, rotate the array by one position in clockwise direction.

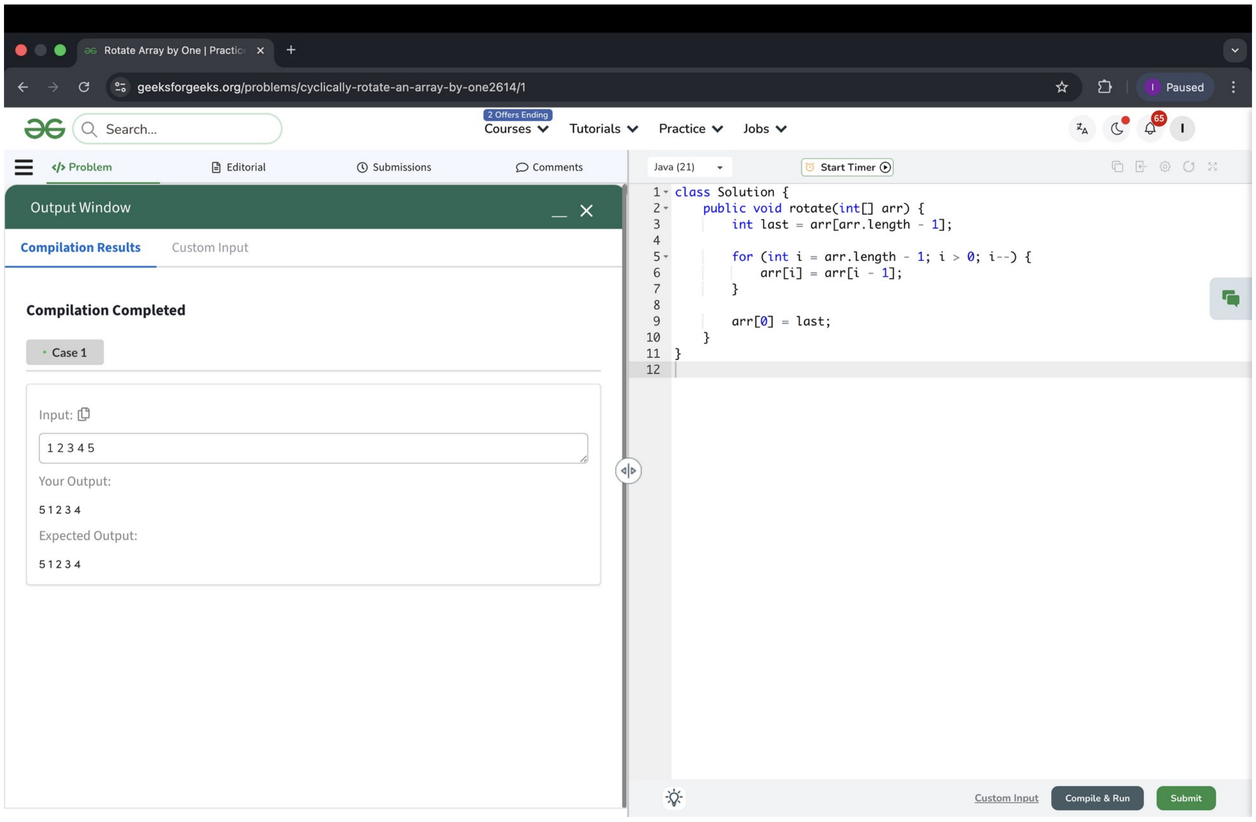
Examples:

Input: arr[] = [1, 2, 3, 4, 5]

Output: [5, 1, 2, 3, 4]

Explanation: If we rotate arr by one position in clockwise 5 come to the front and

remaining those are shifted to the end.



```java
class Solution {
    public void rotate(int[] arr) {
        int last = arr[arr.length - 1];

        for (int i = arr.length - 1; i > 0; i--) {
            arr[i] = arr[i - 1];
        }

        arr[0] = last;
    }
}
```

```
Compilation Completed

Case 1

Input:
1 2 3 4 5
Your Output:
5 1 2 3 4
Expected Output:
5 1 2 3 4
```

EXPERIMENT 2

You are given an integer array **arr[].** You need to find the **maximum** sum of a

subarray (containing at least one element) in the array **arr[]**.

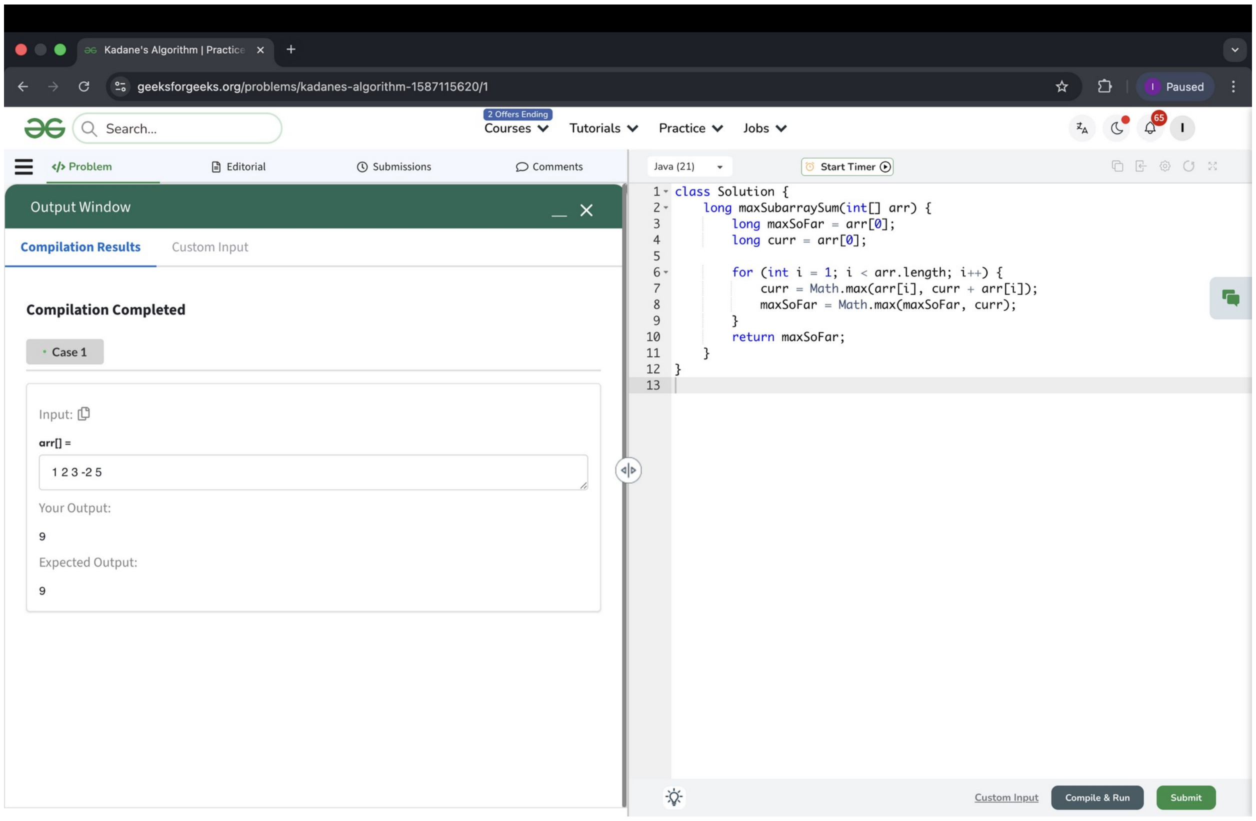
Note : A **subarray** is a continuous part of an array.

Examples:

Input: arr[] = [2, 3, -8, 7, -1, 2, 3]

Output: 11

Explanation: The subarray [7, -1, 2, 3] has the largest sum 11.



Compilation Completed

Case 1

Input:

arr[] =

```
1 2 3 -2 5
```

Your Output:

```
9
```

Expected Output:

```
9
```

```java
class Solution {
    long maxSubarraySum(int[] arr) {
        long maxSoFar = arr[0];
        long curr = arr[0];

        for (int i = 1; i < arr.length; i++) {
            curr = Math.max(arr[i], curr + arr[i]);
            maxSoFar = Math.max(maxSoFar, curr);
        }
        return maxSoFar;
    }
}
```

EXPERIMENT 3

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

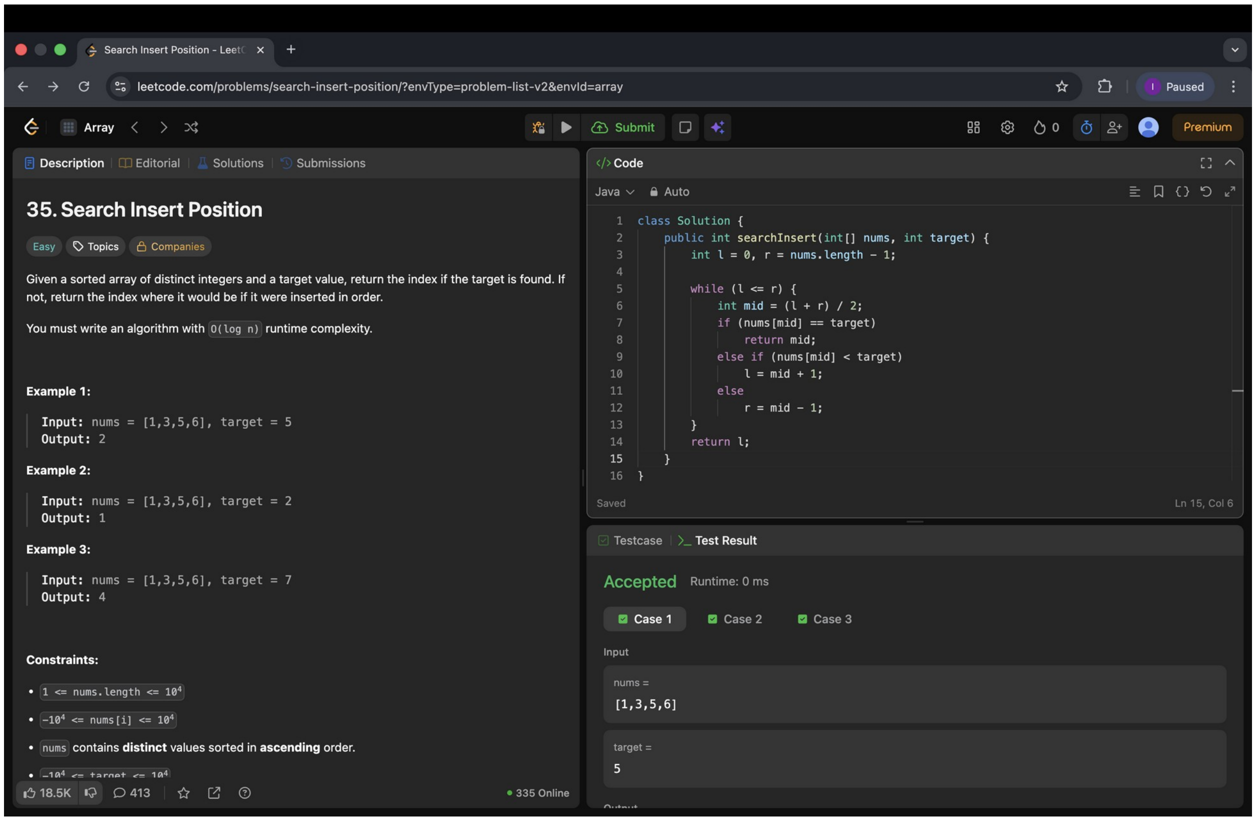
order.

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

Example 1:

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

Output: 2


Search Insert Position - LeetC

leetcode.com/problems/search-insert-position/?envType=problem-list-v2&envId=array

Array

Submit

Premium

Description | Editorial | Solutions | Submissions

35. Search Insert Position

Easy | Topics | Companies

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
```

Constraints:

- `1 <= nums.length <= 10^4`
- `-10^4 <= nums[i] <= 10^4`
- `nums` contains **distinct** values sorted in **ascending** order.
- `-10^4 <= target <= 10^4`

18.5K | 413 | 335 Online

Code

Java | Auto

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

        while (l <= r) {
            int mid = (l + r) / 2;
            if (nums[mid] == target)
                return mid;
            else if (nums[mid] < target)
                l = mid + 1;
            else
                r = mid - 1;
        }
        return l;
    }
}
```

Saved | Ln 15, Col 6

Testcase | Test Result

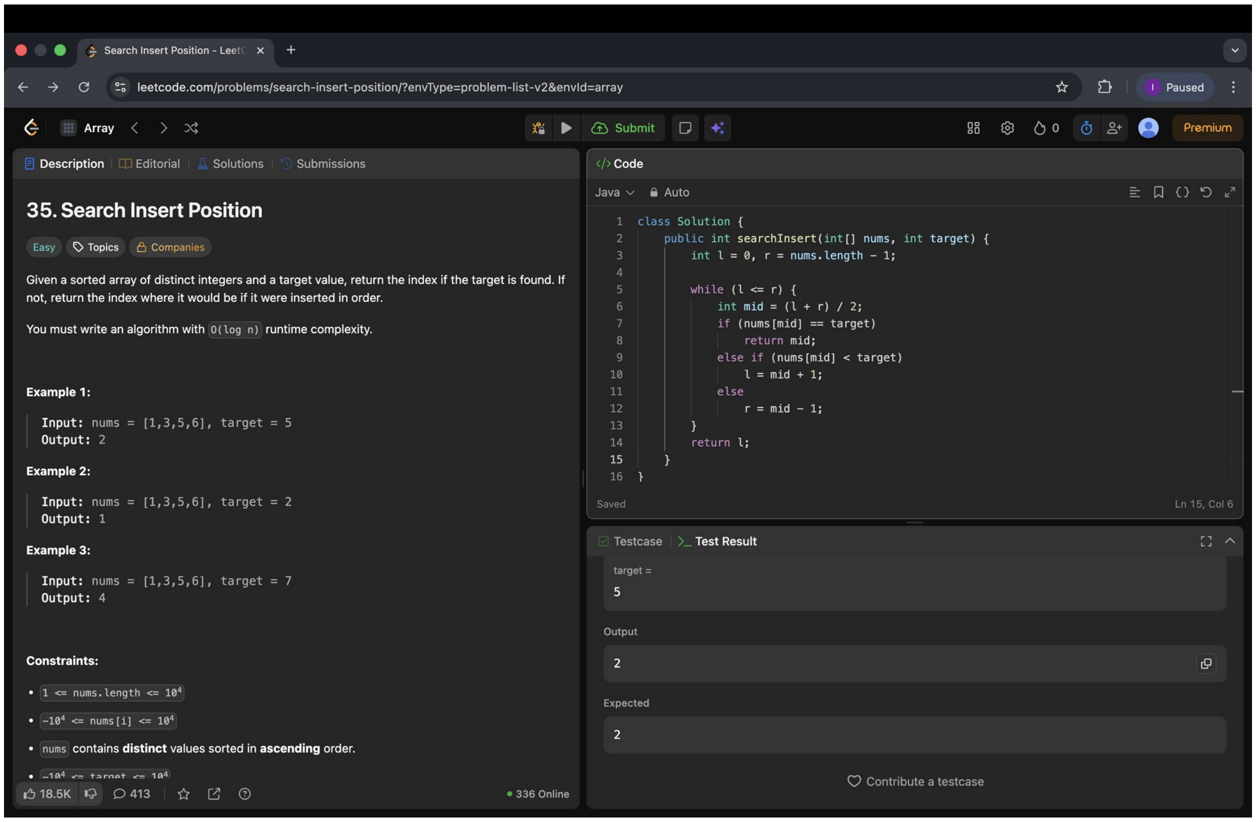
Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

nums =
```
[1,3,5,6]
```

target =
```
5
```


Search Insert Position - LeetC

leetcode.com/problems/search-insert-position/?envType=problem-list-v2&envId=array

Array

Submit

Premium

Description | Editorial | Solutions | Submissions

35. Search Insert Position

Easy | Topics | Companies

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
```

Constraints:

- `1 <= nums.length <= 10^4`
- `-10^4 <= nums[i] <= 10^4`
- `nums` contains **distinct** values sorted in **ascending** order.
- `-10^4 <= target <= 10^4`

18.5K | 413 | 336 Online

Code

Java | Auto

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

        while (l <= r) {
            int mid = (l + r) / 2;
            if (nums[mid] == target)
                return mid;
            else if (nums[mid] < target)
                l = mid + 1;
            else
                r = mid - 1;
        }
        return l;
    }
}
```

Saved | Ln 15, Col 6

Testcase | Test Result

target =
```
5
```

Output
```
2
```

Expected
```
2
```

Contribute a testcase

EXPERIMENT 4

Given an array of integers nums and an integer target, return *indices of the two*

numbers such that they add up to target.

You may assume that each input would have ***exactly*** **one solution**, and you may

not use the *same* element twice.

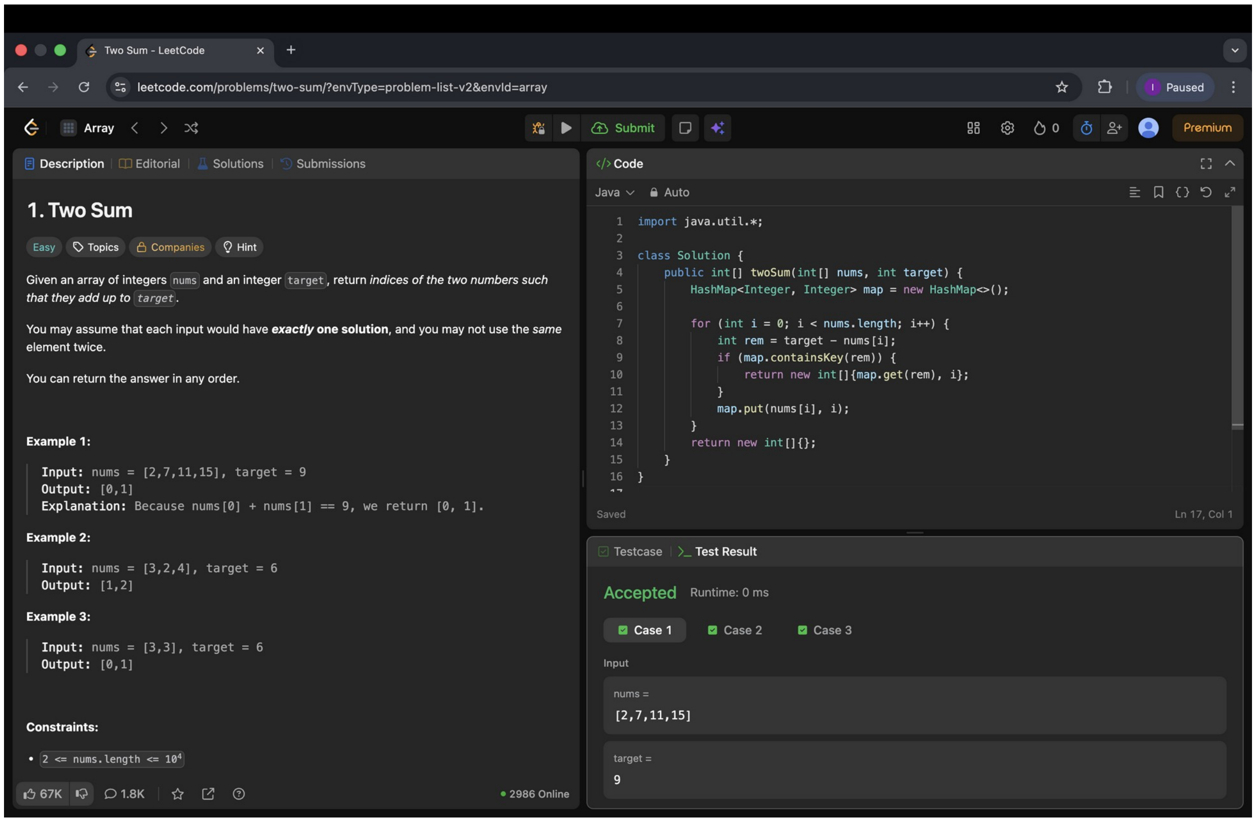
You can return the answer in any order.

Example 1:

Input: nums = [2,7,11,15], target = 9

Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].



1. Two Sum

Easy · Topics · Companies · Hint

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly*** **one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

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

Constraints:

- `2 <= nums.length <= 10`^{4}

67K · 1.8K · 2986 Online

```java
import java.util.*;

class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {
            int rem = target - nums[i];
            if (map.containsKey(rem)) {
                return new int[]{map.get(rem), i};
            }
            map.put(nums[i], i);
        }
        return new int[]{};
    }
}
```

Saved · Ln 17, Col 1

Testcase | Test Result

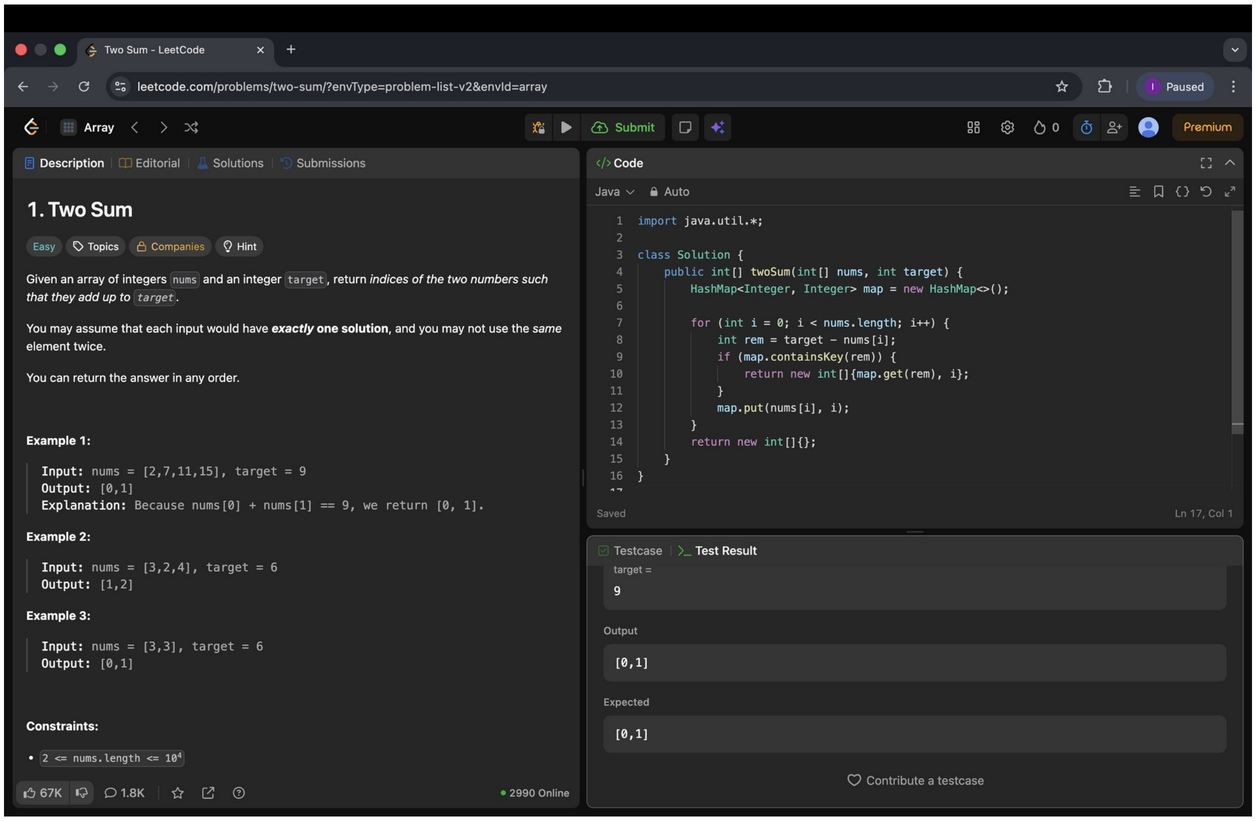
Accepted Runtime: 0 ms

Case 1 · Case 2 · Case 3

Input

```
nums =
[2,7,11,15]
```

```
target =
9
```



Two Sum - LeetCode

leetcode.com/problems/two-sum/?envType=problem-list-v2&envId=array

Array Submit Premium

Description | Editorial | Solutions | Submissions

1. Two Sum

Easy Topics Companies Hint

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to `target`*.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

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

Constraints:

- `2 <= nums.length <= 10`^{4}

67K 1.8K 2990 Online

Code

Java Auto

```java
import java.util.*;

class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {
            int rem = target - nums[i];
            if (map.containsKey(rem)) {
                return new int[]{map.get(rem), i};
            }
            map.put(nums[i], i);
        }
        return new int[]{};
    }
}
```

Saved Ln 17, Col 1

Testcase | Test Result

target =

```
9
```

Output

```
[0,1]
```

Expected

```
[0,1]
```

Contribute a testcase

EXPERIMENT 5

You are given an array **arr[]** of non-negative numbers. Each number tells you

the **maximum number of steps** you can jump forward from that position.

For example:

- If **arr[i] = 3**, you can jump to index **i + 1**, **i + 2**, or **i + 3** from position **i**.

- If **arr[i] = 0**, you **cannot jump forward** from that position.

Your task is to find the **minimum number of jumps** needed to move from

the **first** position in the array to the **last** position.

Note: Return **-1** if you can't reach the end of the array.

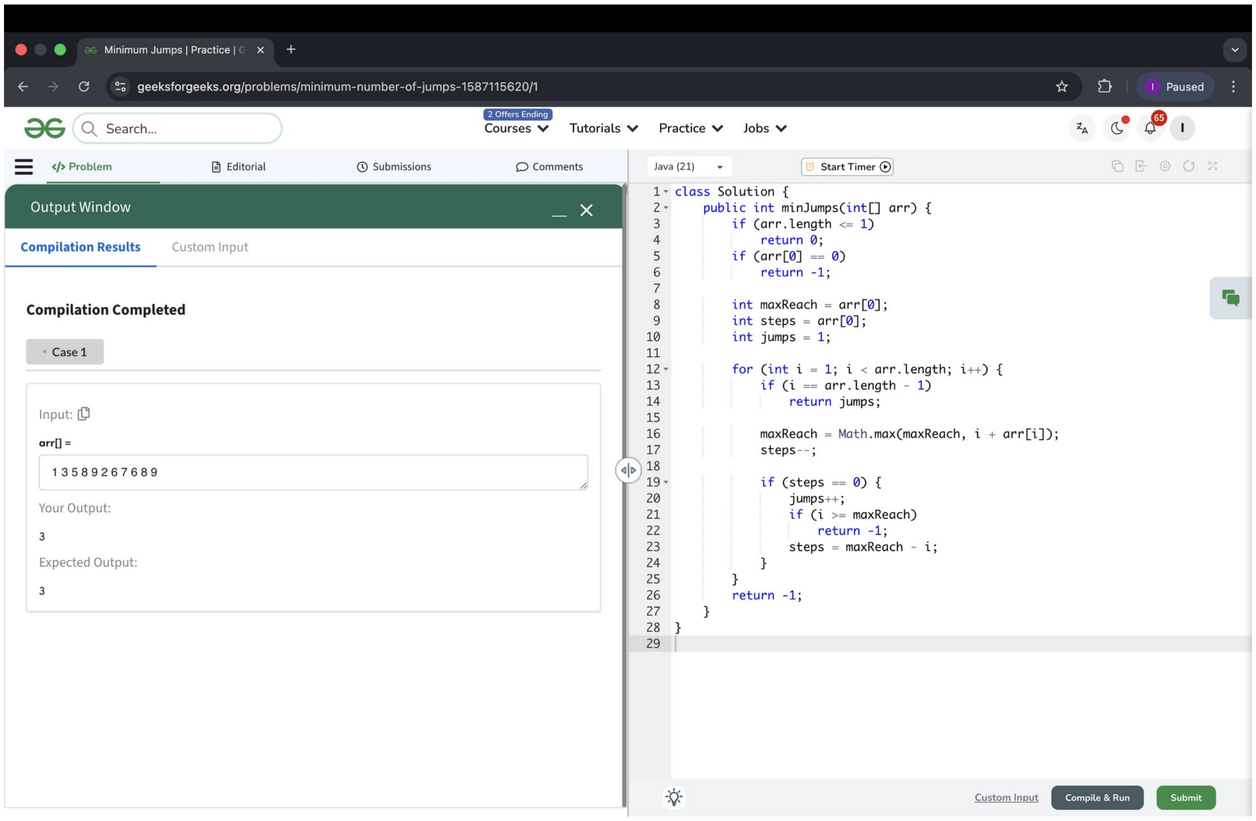
Examples :

Input: arr[] = [1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9]

Output: 3

Explanation: First jump from 1st element to 2nd element with value 3. From here

we jump to 5th element with value 9, and from here we will jump to the last.



Compilation Completed

Input:

arr[] =

```
1 3 5 8 9 2 6 7 6 8 9
```

Your Output:

```
3
```

Expected Output:

```
3
```

```java
class Solution {
    public int minJumps(int[] arr) {
        if (arr.length <= 1)
            return 0;
        if (arr[0] == 0)
            return -1;

        int maxReach = arr[0];
        int steps = arr[0];
        int jumps = 1;

        for (int i = 1; i < arr.length; i++) {
            if (i == arr.length - 1)
                return jumps;

            maxReach = Math.max(maxReach, i + arr[i]);
            steps--;

            if (steps == 0) {
                jumps++;
                if (i >= maxReach)
                    return -1;
                steps = maxReach - i;
            }
        }
        return -1;
    }
}
```