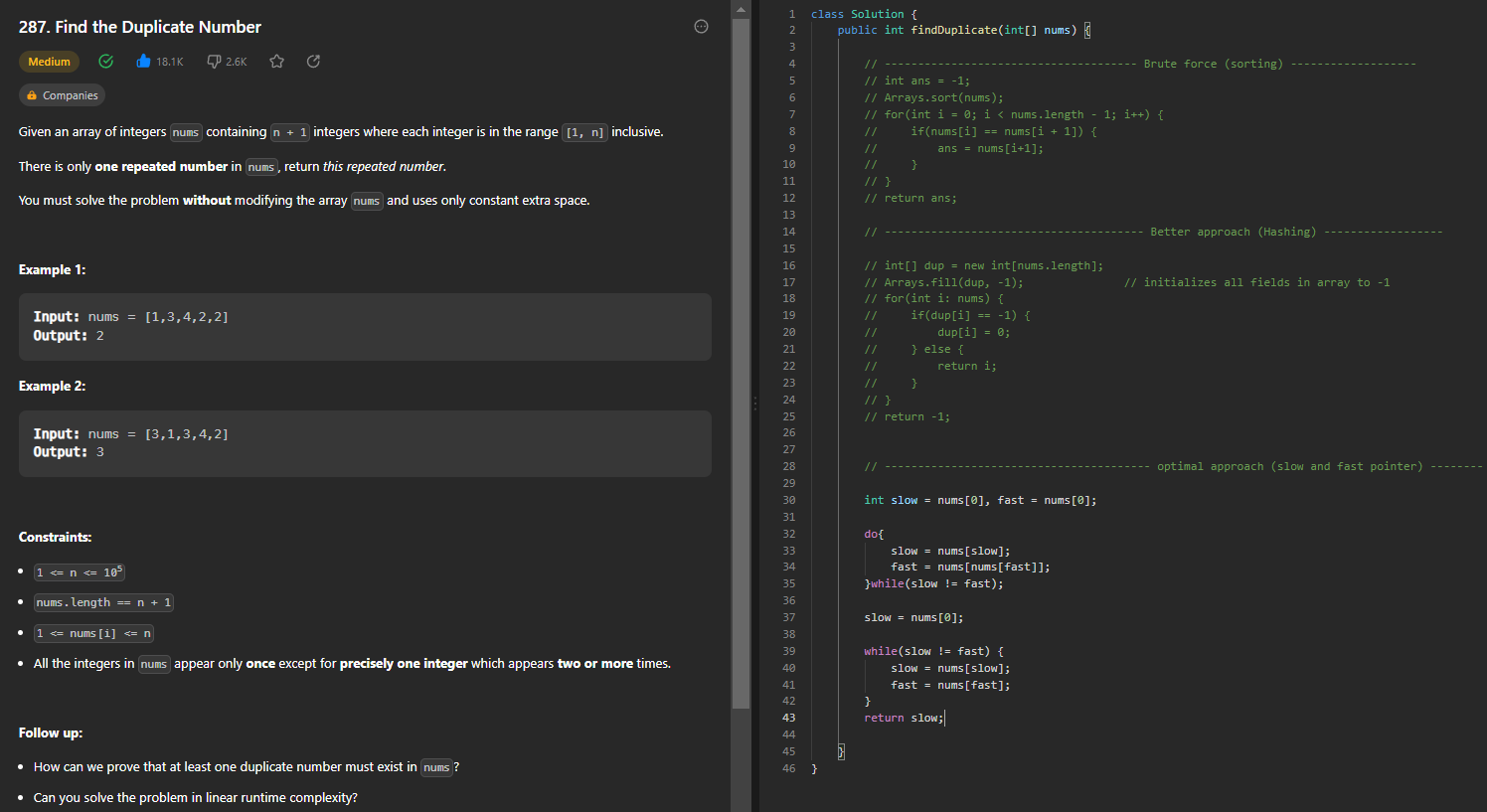
|  |  |
| --- | --- |
| **Arrays** | |
| Find the duplicates number | 1 |
| Sort array of 0 1 and 2’s | 1 |
| Maximum sum subarray | 2 |
| Merge intervals | 2 |
| Pascal’s triangle | 3 |
| Next permutation | 3 |
| Count the number of beautiful subarrays | 4 |
| Rearrange array to maximize prefix score | 4 |
| Kth missing positive integer | 5 |
| Two sum | 5 |

**Find the duplicate number**

Brute force: tc: O(NlogN), sc: O(1)

Better approach: tc: O(N), sc: O(N)

Optimal approach: tc: O(N), sc: O(1)

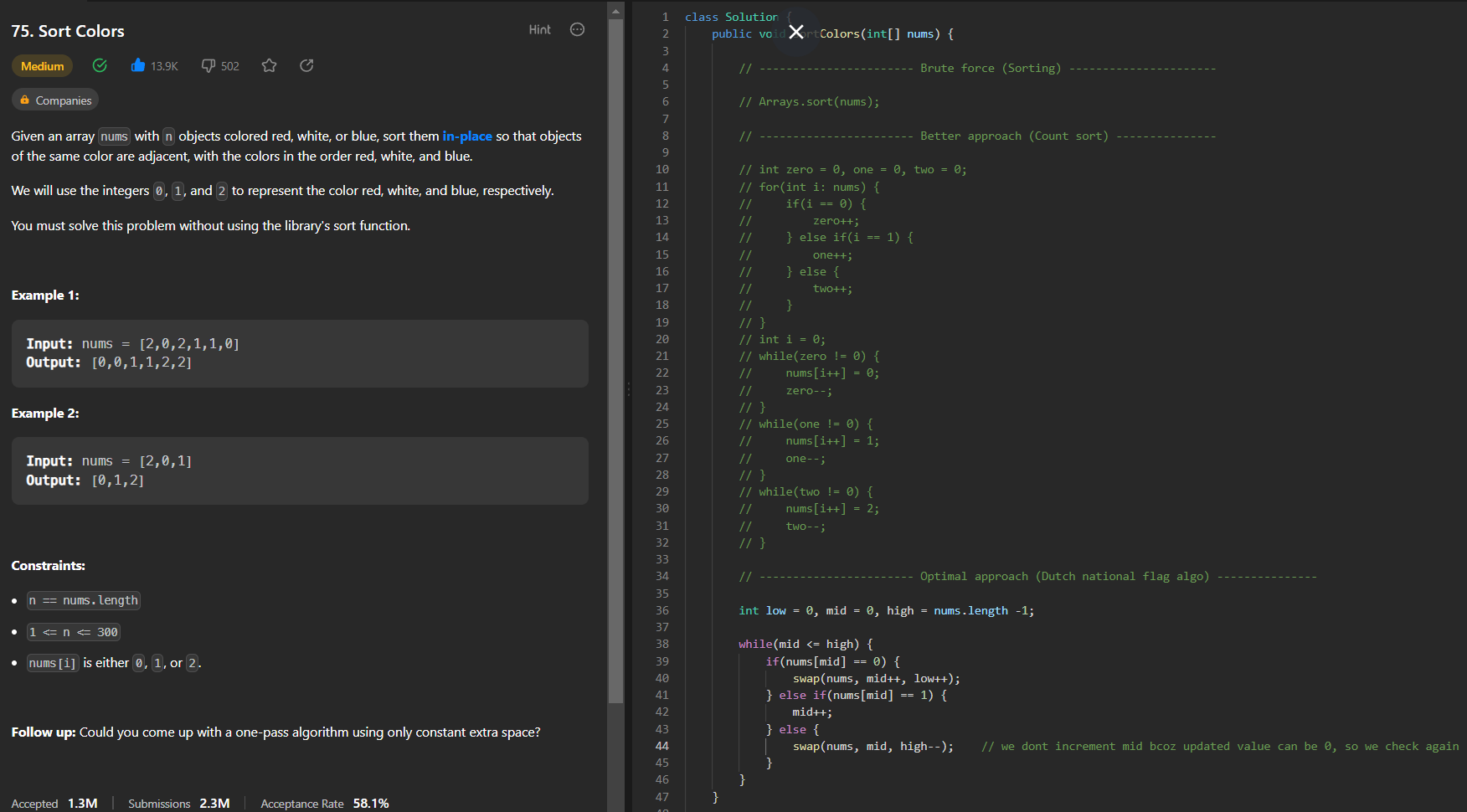


**Sort 0 1 and 2’s**

Brute force: tc: O(NlogN), sc: O(1)

Better approach: tc: O(N + N), sc: O(1)

Optimal approach: tc: O(N), sc: O(1)

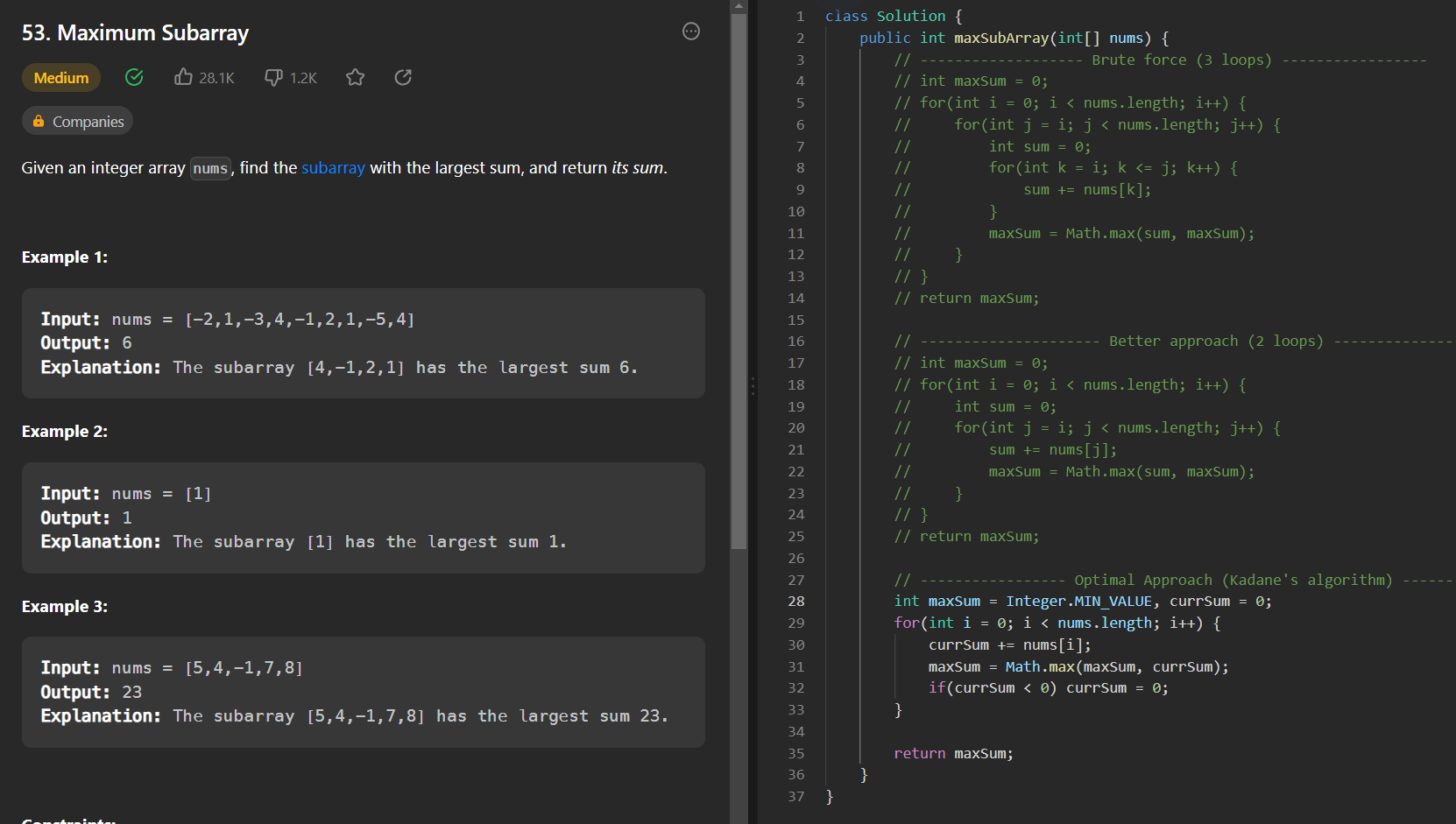


**Maximum sum subarray**

Brute force: tc: O(N^3), sc: O(1)

Better approach: tc: O(N^2), sc: O(1)

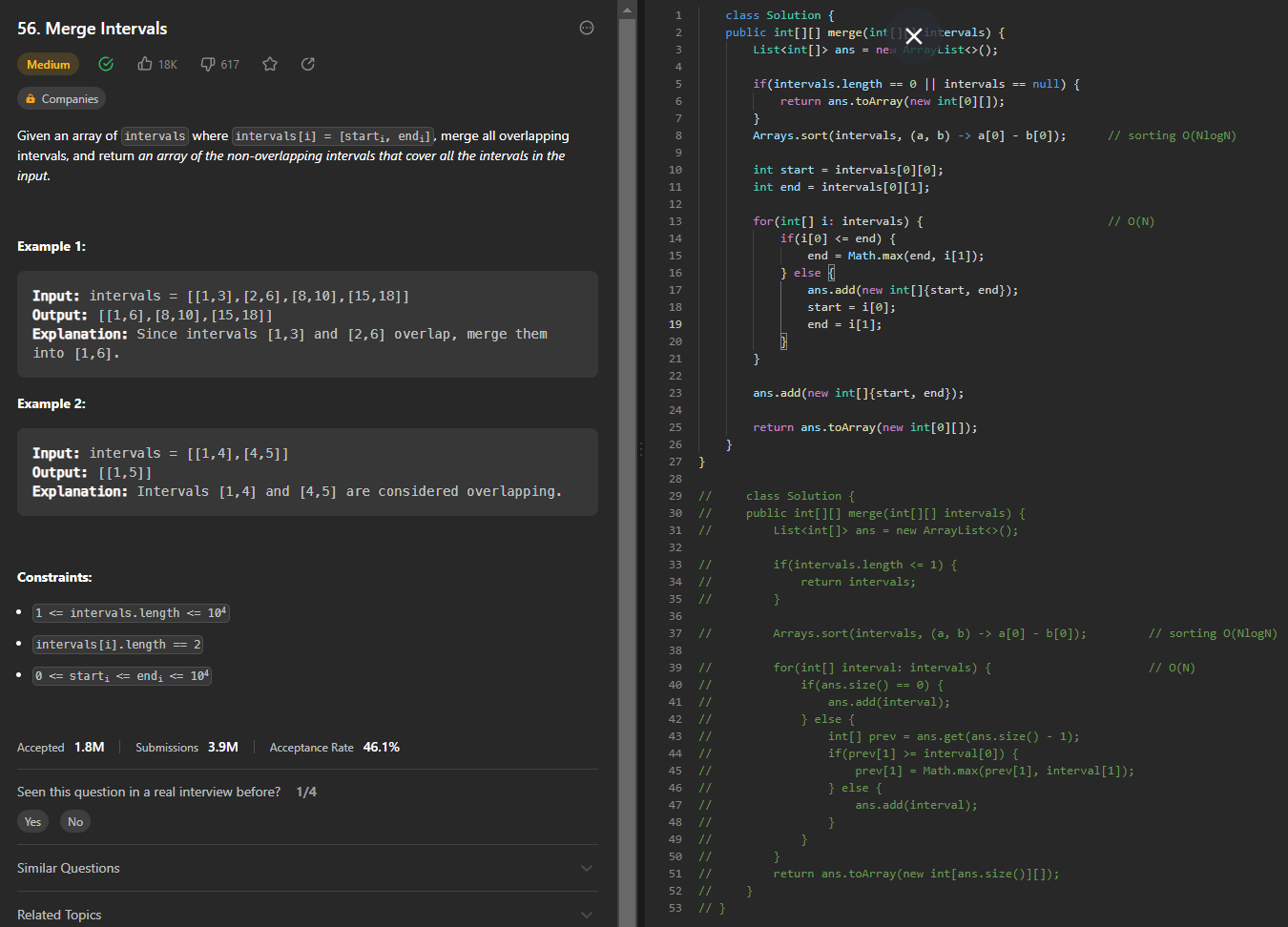
Optimal approach: tc: O(N), sc: O(1)



**Merge Intervals**

Approach 1: tc: O(NlogN), sc: O(1) if we ignore ans array space, if not it is O(N)

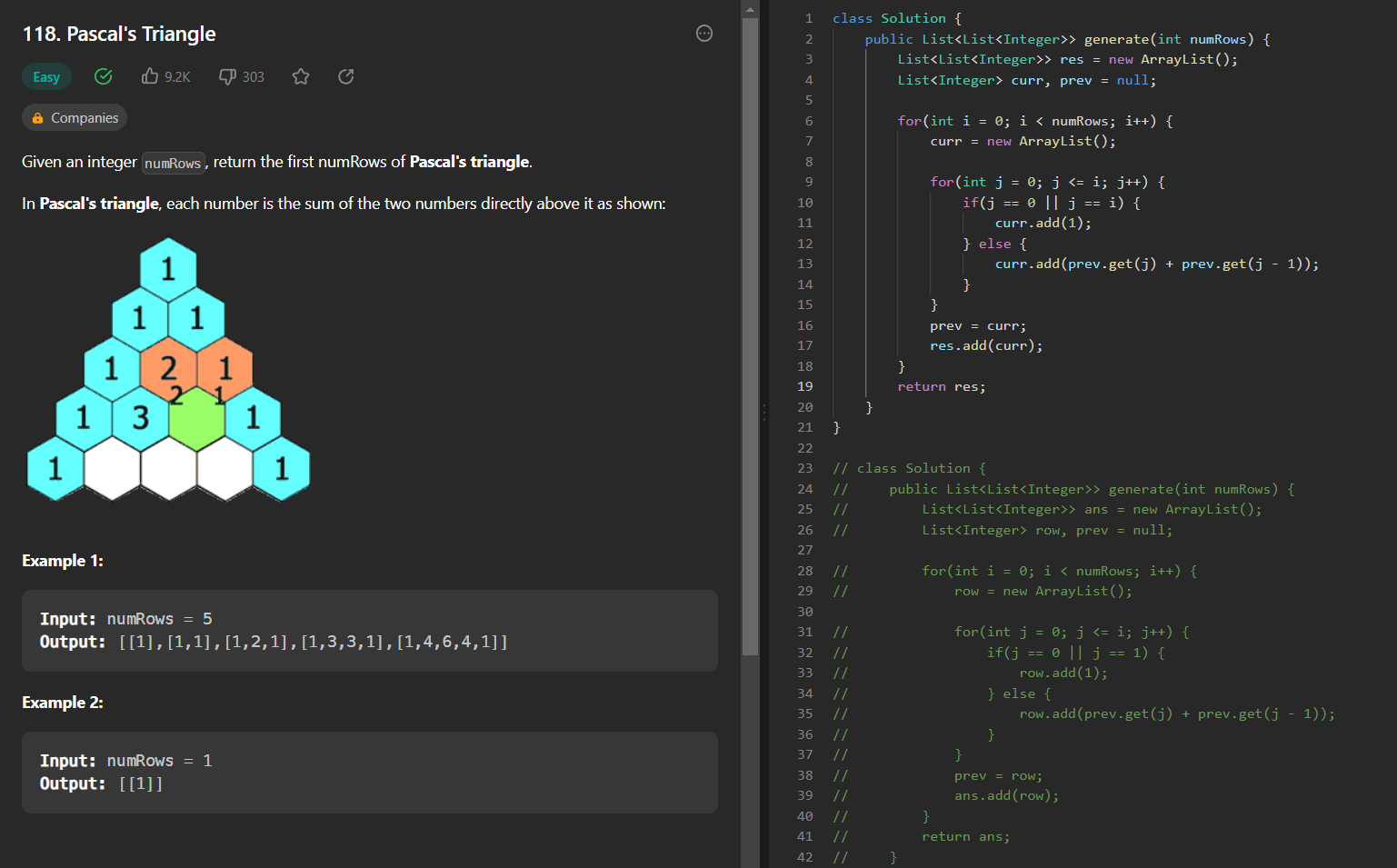
Approach 2: tc: O(NlogN), sc: O(1) if we ignore ans array space, if not it is O(N)



**Pascal’s triangle**

Tc: O(n \* n)

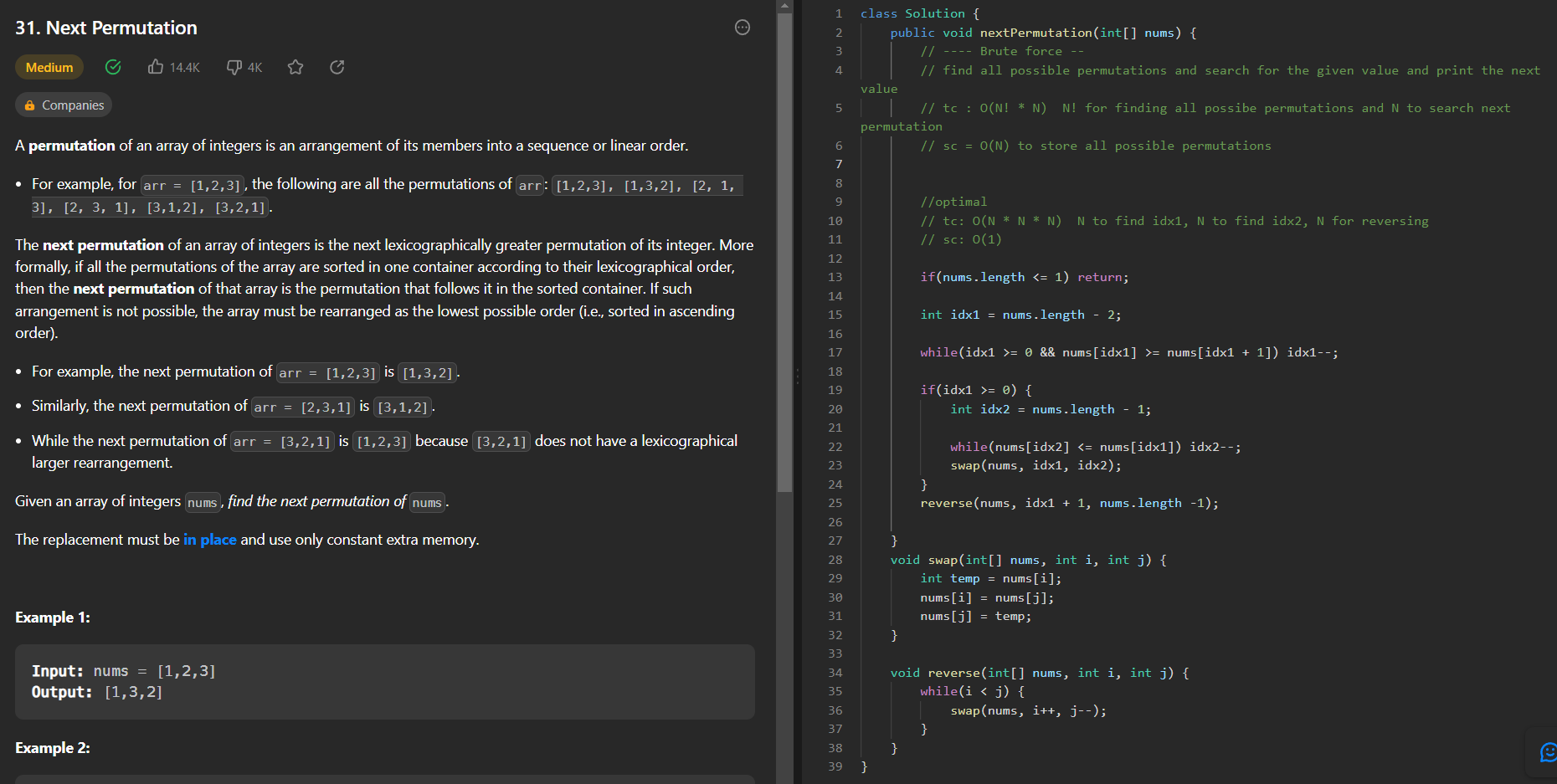
Sc: O(n \* n)



**Next permutation**

Tc: O(n + n + n) => O(3n) => O(n)

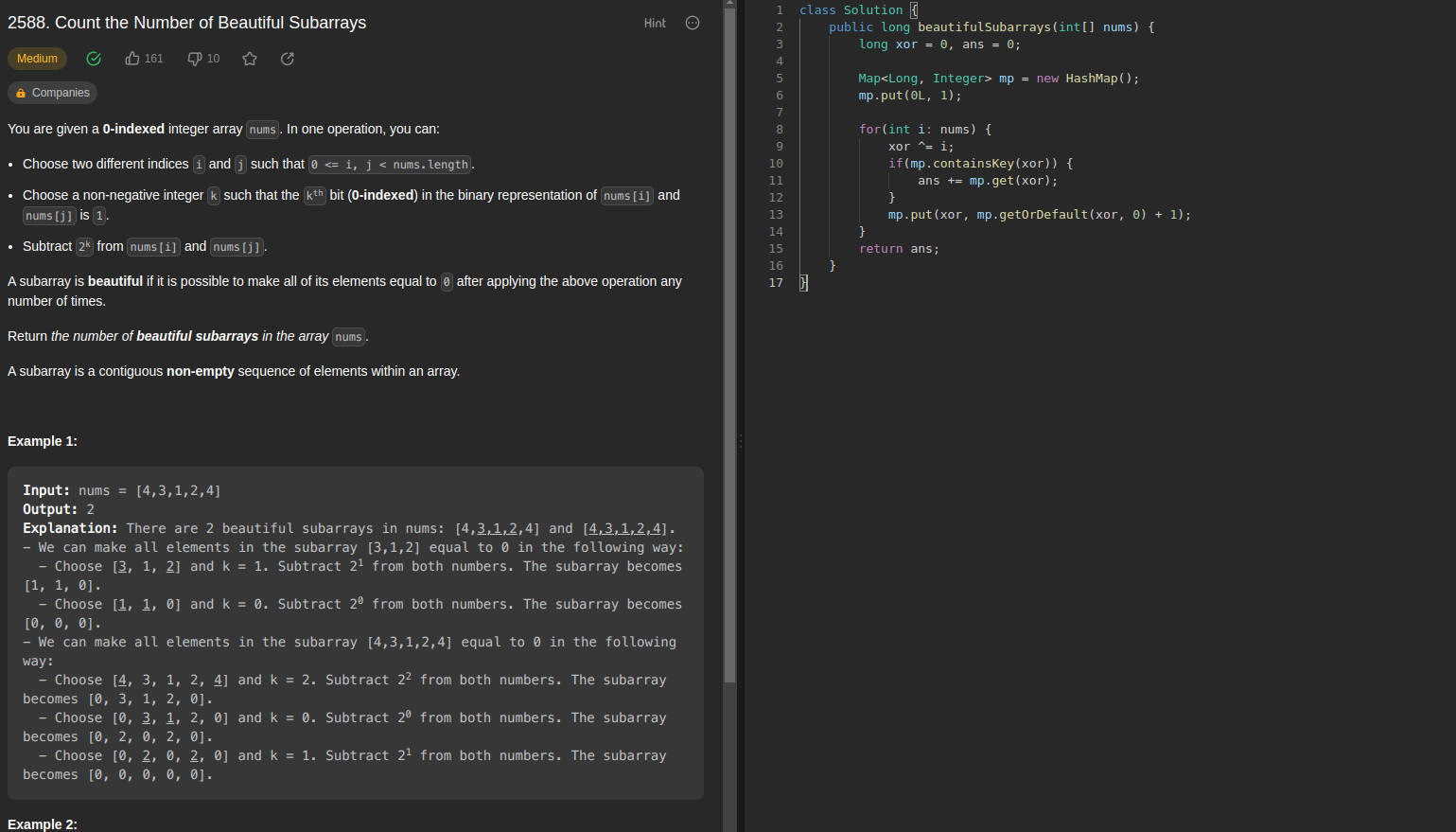
Sc: O(1)



**Count the number of beautiful subarrays**

Tc: O(n)

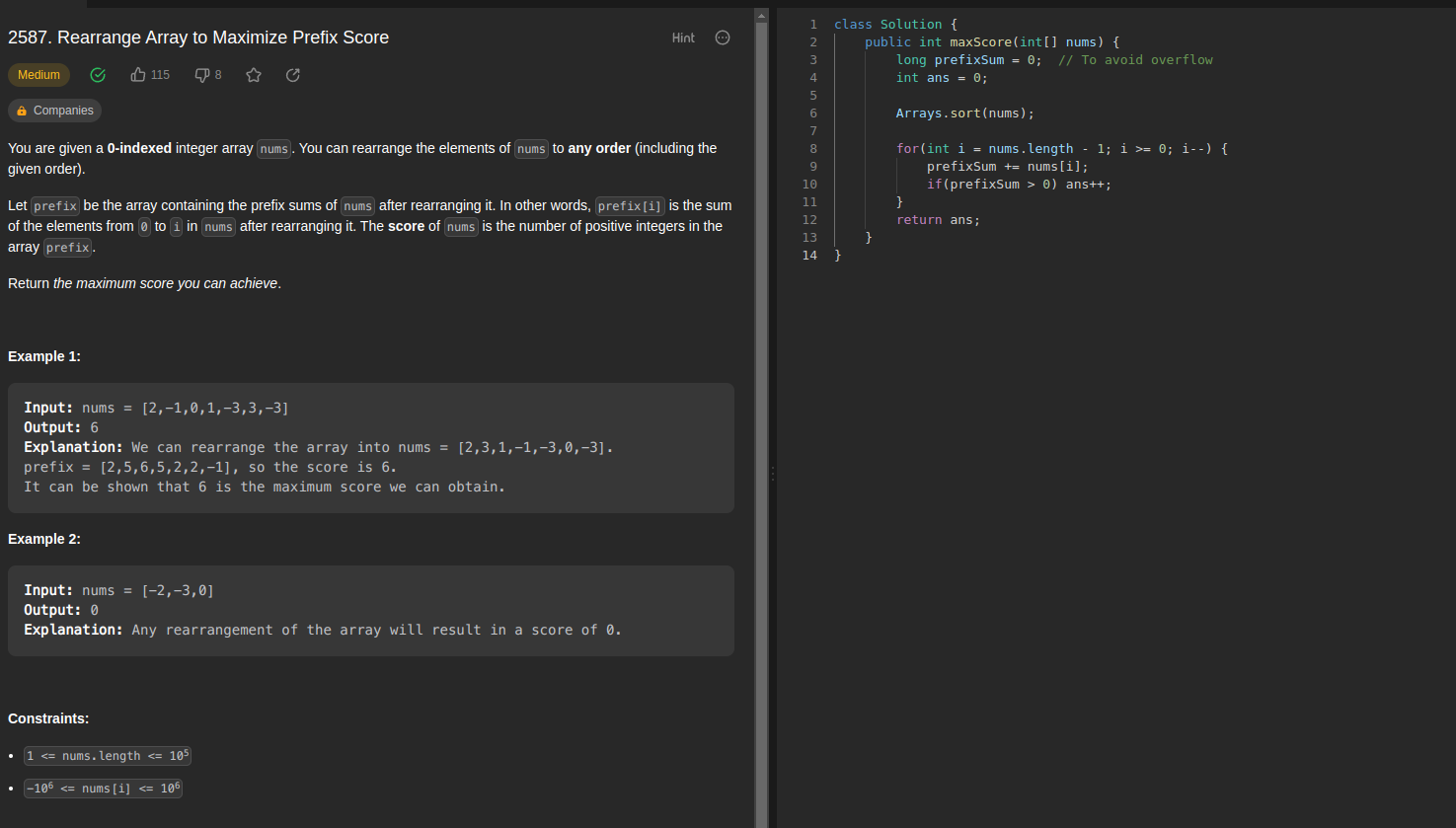
Sc: O(n)



**Rearrange array to maximize prefix score**

Tc: O(NlogN) // sorting

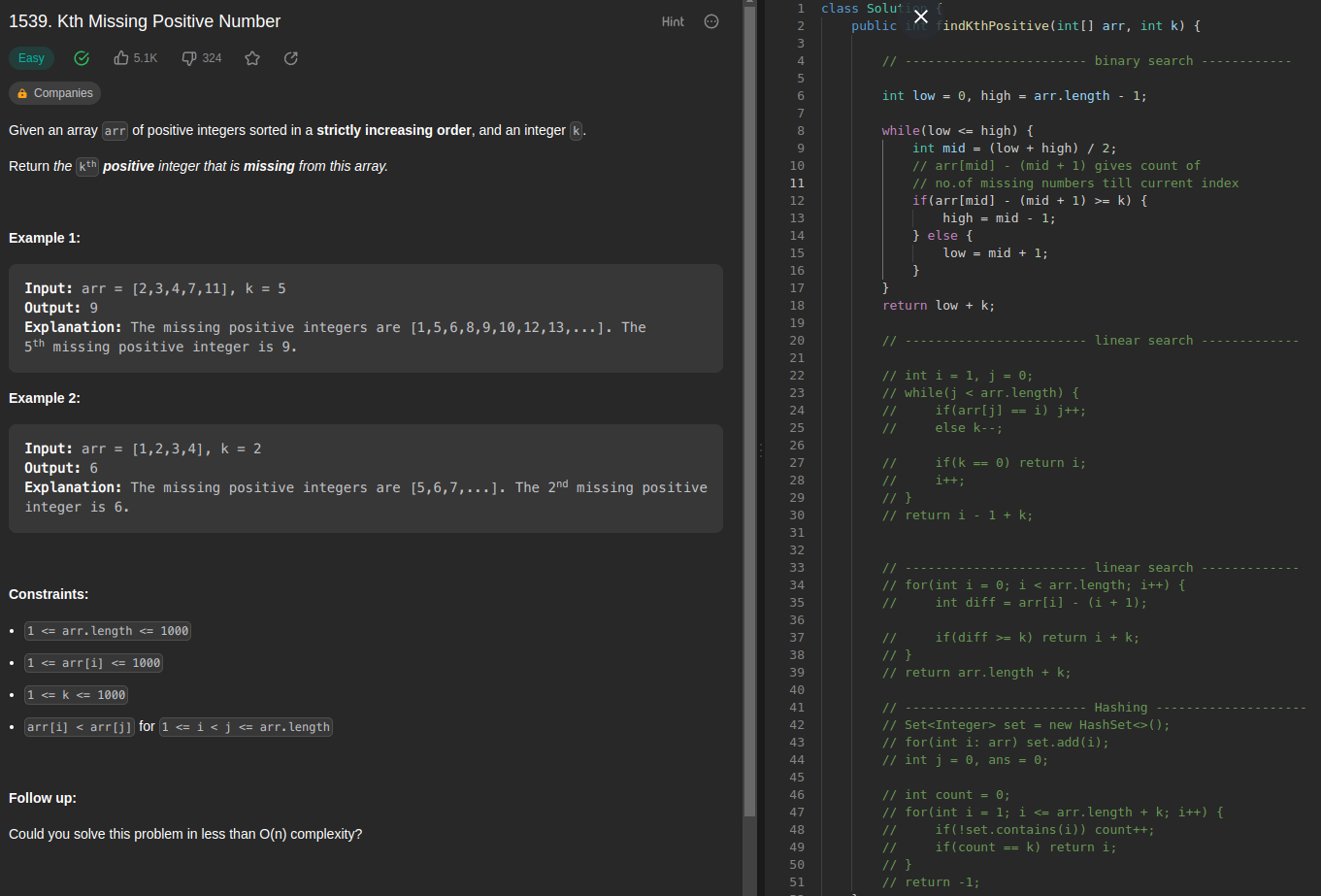
Sc: O(1)

****

**Kth missing positive integer:**

Tc: O(logN)

Sc: O(1)

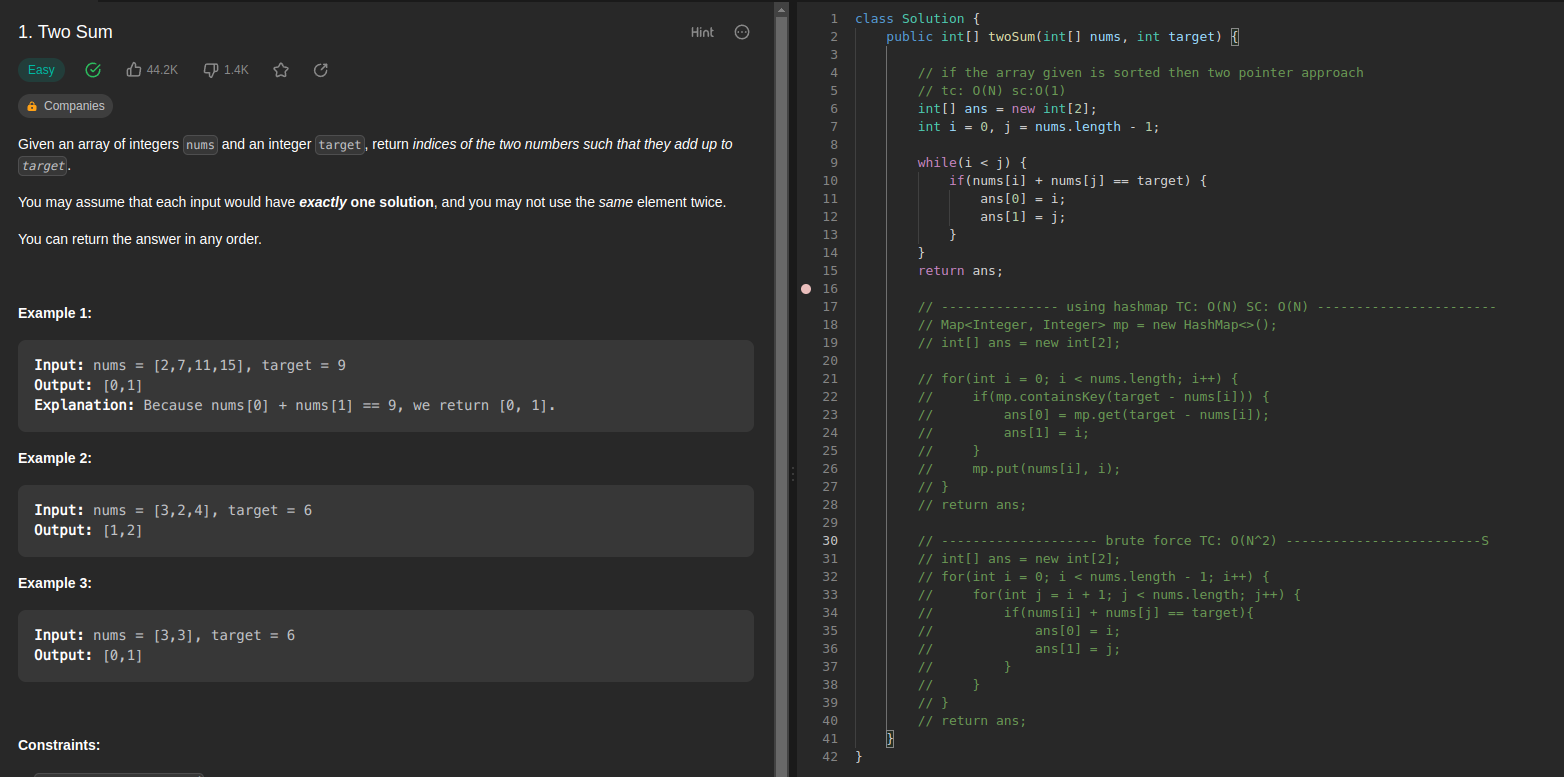
****

**Two sum**

If given array is sorted use two pointer approach else use hashmap

TC: O(N)

SC: O(N) // given array is not sorted

****