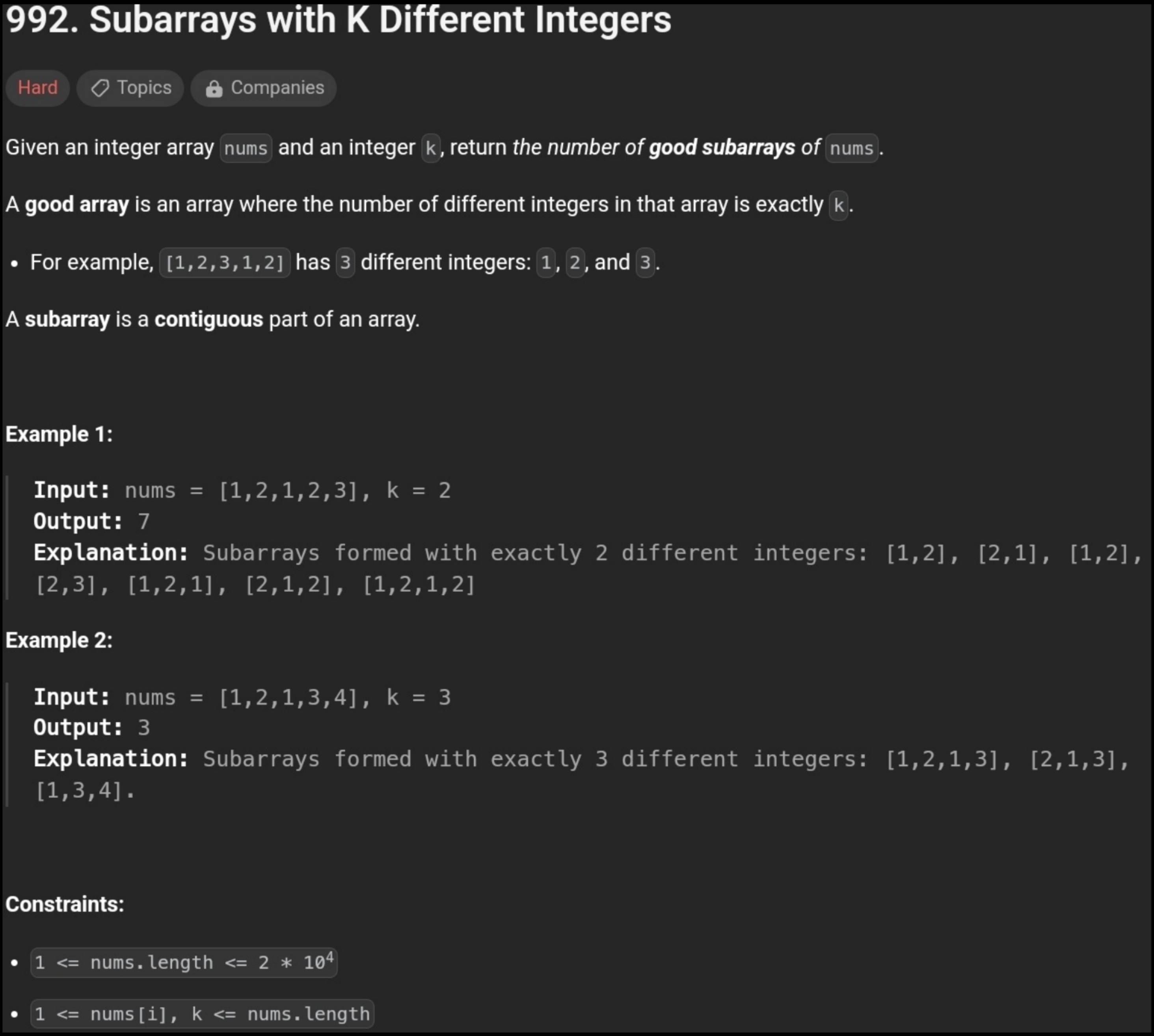


992. Subarrays with K Different Integers

Hard · Topics · Companies

Given an integer array `nums` and an integer `k`, return *the number of* ***good subarrays*** *of* `nums`.

A **good array** is an array where the number of different integers in that array is exactly `k`.

- For example, `[1,2,3,1,2]` has `3` different integers: `1`, `2`, and `3`.

A **subarray** is a **contiguous** part of an array.

Example 1:

```
Input: nums = [1,2,1,2,3], k = 2
Output: 7
Explanation: Subarrays formed with exactly 2 different integers: [1,2], [2,1], [1,2],
[2,3], [1,2,1], [2,1,2], [1,2,1,2]
```

Example 2:

```
Input: nums = [1,2,1,3,4], k = 3
Output: 3
Explanation: Subarrays formed with exactly 3 different integers: [1,2,1,3], [2,1,3],
[1,3,4].
```

Constraints:

- `1 <= nums.length <= 2 * 10`^{4}
- `1 <= nums[i], k <= nums.length`

Approach 1: Brute force

$T(n) : O(n^2)$
$S(n) : O(1)$

Approach 2: Sliding window

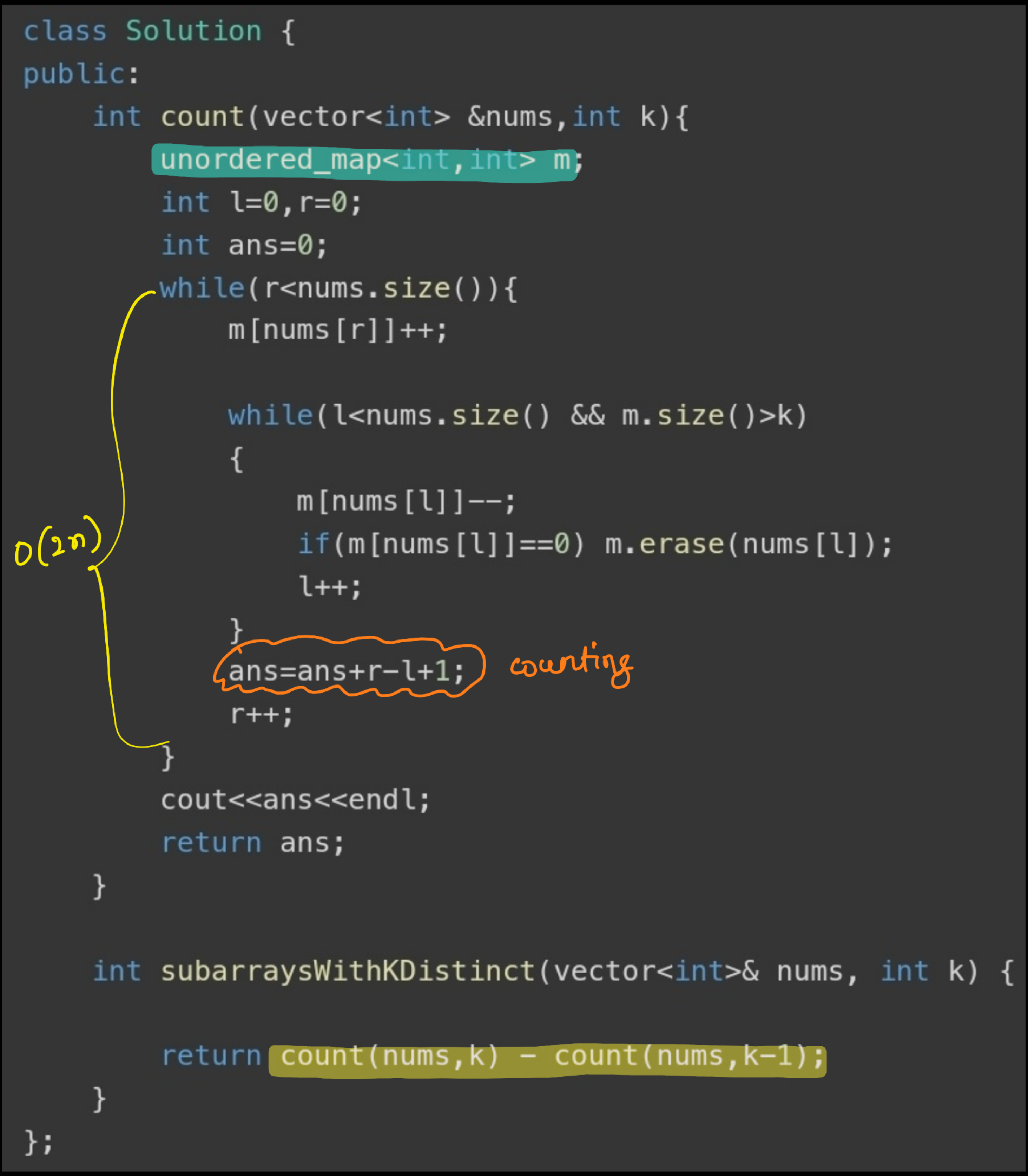
no. of subarrays with exactly K distinct integers = no. of subarrays with atmost K distinct integers − no. of subarrays with atmost (K−1) distinct integers

integers integers integers

Here we are asked to find out the count of subarrays with exactly requirement. " It is not possible to find the count of subarrays with exactly requirements because inner subarrays of a large subarray will also satisfy the requirement and we can't have a fixed protocol on when to shrink, increase or slide the window.

So we go for "atmost" idea.

→ increase the window in each iteration.
→ The moment window contains more than x distinct integers, shrink the window from left till window has only atmost x distinct integers.
→ Repeat.



```
class Solution {
public:
    int count(vector<int> &nums,int k){
        unordered_map<int,int> m;
        int l=0,r=0;
        int ans=0;
        while(r<nums.size()){
            m[nums[r]]++;

            while(l<nums.size() && m.size()>k)
            {
                m[nums[l]]--;
                if(m[nums[l]]==0) m.erase(nums[l]);
                l++;
            }
            ans=ans+r-l+1;
            r++;
        }
        cout<<ans<<endl;
        return ans;
    }

    int subarraysWithKDistinct(vector<int>& nums, int k) {

        return count(nums,k) - count(nums,k-1);
    }
};
```

$T(n) : O(2n) + O(2n)$

$: O(n)$

$S(n) : O(k) + O(k-1)$

$: O(k)$