Count Subarrays with Fixed Bounds

2444. Count Subarrays With Fixed Bounds

Solved

Hard | Topics | Companies | Hint

You are given an integer array `nums` and two integers `minK` and `maxK`.

A **fixed-bound subarray** of `nums` is a subarray that satisfies the following conditions:

- The **minimum** value in the subarray is equal to `minK`.
- The **maximum** value in the subarray is equal to `maxK`.

Return *the **number** of fixed-bound subarrays.*

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

Example 1:

```
Input: nums = [1,3,5,2,7,5], minK = 1, maxK = 5
Output: 2
Explanation: The fixed-bound subarrays are [1,3,5] and [1,3,5,2].
```

Example 2:

```
Input: nums = [1,1,1,1], minK = 1, maxK = 1
Output: 10
Explanation: Every subarray of nums is a fixed-bound subarray. There are 10 possible subarrays.
```

Constraints:

- `2 <= nums.length <= 10^5`
- `1 <= nums[i], minK, maxK <= 10^6`

Seen this question in a real interview before? 1/5

Yes No

eg. nums = [1, 3, 5, 2, 7, 5], minK = 1, maxK = 5

min = [1)
max = [1]

[1, 3)
[3, 1]

[1, 3, 5)
[

Two Pointers / Brute Force (TLE) :-

- The brute force approach is to simply traverse through all the subarrays & keep a track of the number of subarrays satisfying the condition stated in the problem statement.
- Although this gives the TLE due to the high constraints

T.C. → $O(n^2)$

S. C. ⟶ O(1)

```cpp
class Solution {
public:
    long long countSubarrays(vector<int>& nums, int minK, int maxK) {
        long count = 0;
        for(int i=0 ; i<nums.size(); i++){
            int mini = INT_MAX , maxi = INT_MIN;
            for(int j=i; j<nums.size();j++){
                mini = min(mini, nums[j]);
                maxi = max(maxi, nums[j]);
                if(mini == minK && maxi == maxK)++count;
            }
        }
        return count;
    }
};
```

Sliding Window :-

Key Observation :-

1. Any number outside the range [minK, maxK] invalidates a subarray.

2. A valid subarray must contain :-

   - At least one minK
   - At least one maxK
   - no values outside the [minK, maxK] range.

So we must :

- keep track of the most recent indices of minK & maxK.

- Track the most recent index where an invalid number appeared.

→ we'll do a single pass using the sliding window technique & maintain:

last_invalid: last index where nums[i] < mink or nums[i] > maxk

last_mink: last index where nums[i] == mink

last_maxk: last index where nums[i] == maxk.

At each index i:

- If nums[i] is invalid → reset all pointers.
- Else:
  - Update last_mink & last_maxk if applicable
  - Calculate valid_start = min(last_mink, last_maxk)
  - If valid_start > last_invalid → we can form (valid_start - last_invalid) valid subarrays ending at i.

C++ Auto

```cpp
class Solution {
public:
    long long countSubarrays(vector<int>& nums, int minK, int maxK) {
        int n = nums.size();
        long long count=0;

        int last_invalid = -1;
        int last_minK = -1;
        int last_maxK = -1;

        for(int i=0; i<n; i++){
            int x=nums[i];
            if(x<minK || x>maxK)last_invalid = i;
            else{
                if(x==minK)last_minK=i;
                if(x==maxK)last_maxK=i;
            }
            if(min(last_minK, last_maxK)>last_invalid)count+=min(last_minK, last_maxK)-last_invalid;
        }
        return count;
    }
};
```

Saved

T.C. O(n)

S.C. O(1)

Using Double Deque / Monotonic Queue

- when we're solving problems where we need to maintain or track the max & min for subarray, the simplest method is to maintain a min and max deque while using sliding window.

- Min Deque Track the indices in a monotonic increasing order to find the minimum element in the window.

- Max Deque Tracks the indices in a monotonic decreasing order to find the maximum element in the window.

C++ Auto

```cpp
class Solution {
public:
    long long countSubarrays(vector<int>& nums, int minK, int maxK) {
        long long count=0, left=0;
        deque<int> dq_min, dq_max;

        for(int i=0;i<nums.size();i++){
            if(nums[i]<minK || nums[i]>maxK){
                dq_min.clear();
                dq_max.clear();
                left=i+1;
                continue;
            }

            while(!dq_min.empty() && nums[dq_min.back()]>=nums[i])dq_min.pop_back();
            dq_min.push_back(i);

            while(!dq_max.empty() && nums[dq_max.back()]<=nums[i])dq_max.pop_back();
            dq_max.push_back(i);

            if(nums[dq_min.front()]==minK && nums[dq_max.front()] == maxK){
                int start = min(dq_min.front(), dq_max.front());
                count+=(start-left+1);
            }
        }
        return count;
    }
};
```

T.C. O(n) & S.C. O(n)