713. subarray product less than k

I/P: nums = [10, 5, 2, 6], k = 100

O/P: 8

Explanation: The 8 subarrays that have product less than 100 are:

[10], [5], [2], [6], [10,5], [5,2], [2,6], [5,2,6]

Note that [10,5,2] is not included as the product of 100 is not strictly less than k.

I/P: nums = [10, 5, 2, 6] k = 100

we list all subarrays and check their product:

single elements:

[10] → 10 < 100

[5] → 5 < 100

[2] → 2 < 100

[6] → 6 < 100

Two elements;

[10,5] → 50 < 100

[5,2] → 10 < 100

[2,6] → 12 < 100

Three elements:

[10,5,2] → 100 ✗

[5,2,6] → 60 < 100

Four elements:

[10,5,2,6] → 600 ✗

Total valid subarrays = 8.

Brute force:

```
class Solution {
    public int numSubarrayProductLessThanK(int[]
                                    nums, int k) {
        if (k <= 1) return 0;
        int count = 0;
        int n = nums.length;
        for(int start = 0; start < n; start++) {
            int product = 1;
            for (int end = start; end < n; end++) {
                product = product * nums[end];
                if (product < k) {
                    count++;
                }else {
                    break;
                }
            }
        }
        return count;
    }
}
```

Time → $O(n^2)$

Optimal solution (sliding / two pointers):

1. use two pointers: left and right
2. Multiply as you move right
3. If product becomes ≥ k → divide by nums[left] and move left forward
4. Every time product < k → add (right - left + 1) to count.

```
class Solution {
    public int numSubarrayProductLessThank(int[]
                                    nums, int k) {
        if (k <= 1) return 0;
        int left = 0;
        int product = 1;
        int count = 0;
      for(int right = 0; right < nums.length; right++){
          product *= nums[right];
        while (product >= k){
            product /= nums[left];
            left++;
          }
          count += right - left + 1;
        }
        return count;
    }
}
```

Time → O(n) space → (O(1)).