<https://leetcode.com/problems/count-subarrays-with-median-k/description/>

You are given an array nums of size n consisting of **distinct**integers from 1 to n and a positive integer k.

Return **the number of non-empty subarrays in**nums**that have a median equal to**k.

**Note**:

The median of an array is the **middle**element after sorting the array in **ascending**order. If the array is of even length, the median is the **left**middle element.

For example, the median of [2,3,1,4] is 2, and the median of [8,4,3,5,1] is 4.

A subarray is a contiguous part of an array.

**Example 1:**

**Input:** nums = [3,2,1,4,5], k = 4

**Output:** 3

**Explanation:** The subarrays that have a median equal to 4 are: [4], [4,5] and [1,4,5].

**Example 2:**

**Input:** nums = [2,3,1], k = 3

**Output:** 1

**Explanation:** [3] is the only subarray that has a median equal to 3.

**Constraints:**

n == nums.length

1 <= n <= 10^5

1 <= nums[i], k <= n

The integers in nums are distinct.

**Attempt 1: 2023-02-10**

**Solution 1: Hash Table + Math (60 min)**

class Solution {

    public int countSubarrays(int[] nums, int k) {

        int pivot = -1;

        // Since not a sorted array, need looply scan to find the 'pivot' index

        for(int i = 0; i < nums.length; i++) {

            if(nums[i] == k) {

                pivot = i;

                break;

            }

        }

        // k not found in nums

        if(pivot == -1) {

            return 0;

        }

        // Considering the following 3 scenario with an example nums = [4,2,3,5,1]:

        // Scenario 1 - the subarrays are end with nums[pivot], such as [4,2,3]

        // Scenario 2 - the subarrays are begin with nums[pivot], such as [3,5] and [3,5,1]

        // Scenario 3 - the subarrays are across nums[pivot], such as [4,2,3,5,1]

        // And in any of above Scenario, we must include nums[pivot] itself,

        // that's also why 'result' initialized as 1

        int result = 1;

        int score = 0;

        // Store the number of scores at the left, used for Scenario 3

        Map<Integer, Integer> map = new HashMap<>();

        // Scan on the pivot's left side, and critical point is we have to scan

        // from right to left (pivot - 1 -> 0), because the 'cardinal element'

        // is 'nums[pivot]', if scan from (0 -> pivot - 1), it might not be a

        // subarray, e.g nums = [10, 3, 8, 5, 6, 7, 2, 9, 4, 1], k = 9, if

        // scan from (0 -> pivot - 1), '10' will make score += 1 but '10' and '9'

        // not form a subarray

        //for(int i = 0; i < pivot; i++) {

        for(int i = pivot - 1; i >= 0; i--) {

            // Actually 'nums[i] == k' won't include between [0, pivot - 1],

            // because integers in given 'nums' are distinct, the 'nums[i] == k'

            // will only happen when 'i == pivot'

            // Scenario 1: end with 'nums[pivot]'

            score += (nums[i] < k ? -1 : 1);

            // Total score 0 happen when subarray has odd number of elements,

            // number of elements larger than median equal to number of elements

            // smaller than median.

            // Total score 1 happen when subarray has even number of elements,

            // elements larger than median has one more count than elements

            // smaller than median.

            if(score == 0 || score == 1) {

                result++;

            }

            // Store the frequency of a 'score', prepare for the subarrays are across nums[pivot]

            map.put(score, map.getOrDefault(score, 0) + 1);

        }

        // Restore the 'score' to 0 and scan for right side

        score = 0;

        for(int i = pivot + 1; i < nums.length; i++) {

            // Scenario 2: start with 'nums[pivot]'

            score += (nums[i] < k ? -1 : 1);

            if(score == 0 || score == 1) {

                result++;

            }

            // Scenario 3: across 'nums[pivot]'

            result += map.getOrDefault(-score, 0) + map.getOrDefault(1 - score, 0);

        }

        return result;

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

**Refer to**

<https://leetcode.com/problems/count-subarrays-with-median-k/solutions/2852074/c-counting-o-n-detailed-explaination/>

**Approach**

The goal is to find the number of subarrays that have a median equal to k.

Let center be the index of value k (nums[center] == k), obviously nums[center] itself is a valid subarray, and let's expand the subarray to the left and right.

The key to have a median = k is balancing the number of elements less than k and greater than k.

Since the integers in nums are distinct, except nums[center], nums[i] is eithor < k or > k.

Let's assign the elements a score:

< k scores -1

> k scores 1

= k scores 0

A valid subarray should have total score 0 or 1

Total score 0 happen when number of elements is odd, number of elements larger than median equal to number of elements smaller than median.

Total score 1 happen when number of elements is even, elements larger than median has one more count than elements smaller than median.

Considering the following 3 scenario with an example nums = [4, 2, 3, 5, 1]:

the subarrays are end with center, such as [4, 2, 3].

the subarrays are begin with center, such as [3, 5] and [3, 5, 1].

the subarrays are across center, such as [4, 2, 3, 5, 1].

and let's see how to compute the answer with these three scenario

**Code**

class Solution {

public:

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

int n = nums.size();

int center = -1;

for ( int i=0; i<n; ++i )

if ( nums[i] == k ) center = i;

// k not found in nums

if ( center == -1 ) return 0;

// init with 1 because nums[center] is an answer

int ans = 1;

// store the number of scores at the left, used for scenario 3

unordered\_map<int,int> cnt\_l;

for ( int i=center-1, sum=0; i>=0; --i ) {

// sum is the accumulate score from center-1 to the left

sum += (nums[i] < k) ? -1 : 1;

// scenario 1: end with `center`

if ( sum == 0 || sum == 1 ) ans++;

// update the counter of this score

cnt\_l[sum]++;

}

for ( int i=center+1, sum=0; i<n; ++i ) {

// sum is the accumulate score from center+1 to the right

sum += (nums[i] < k) ? -1 : 1;

// scenario 2: start with `center`

if ( sum == 0 || sum == 1 ) ans++;

// scenario 3: across `center`

ans += cnt\_l[-sum] + cnt\_l[1-sum];

}

return ans;

}

};