Problem 3632: Subarrays with XOR at Least K

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given an array of positive integers

nums

of length

n

and a non■negative integer

k

.

Return the number of

contiguous

subarrays

whose bitwise XOR of all elements is

greater

than or

equal

to

k

.

Example 1:

Input:

nums = [3,1,2,3], k = 2

Output:

6

Explanation:

The valid subarrays with

XOR >= 2

are

[3]

at index 0,

[3, 1]

at indices 0 - 1,

[3, 1, 2, 3]

at indices 0 - 3,

[1, 2]

at indices 1 - 2,

[2]

at index 2, and

[3]

at index 3; there are 6 in total.

Example 2:

Input:

nums = [0,0,0], k = 0

Output:

6

Explanation:

Every contiguous subarray yields

XOR = 0

, which meets

k = 0

. There are 6 such subarrays in total.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

0 <= k <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long countXorSubarrays(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public long countXorSubarrays(int[] nums, int k) {

}
}
```

Python3:

```
class Solution:
def countXorSubarrays(self, nums: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def countXorSubarrays(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var countXorSubarrays = function(nums, k) {

};
```

TypeScript:

```
function countXorSubarrays(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public long CountXorSubarrays(int[] nums, int k) {

}
}
```

C:

```
long long countXorSubarrays(int* nums, int numsSize, int k) {

}
```

Go:

```
func countXorSubarrays(nums []int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun countXorSubarrays(nums: IntArray, k: Int): Long {

}
```

```
}
```

Swift:

```
class Solution {
func countXorSubarrays(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_xor_subarrays(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def count_xor_subarrays(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function countXorSubarrays($nums, $k) {

}
}
```

Dart:

```
class Solution {
int countXorSubarrays(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def countXorSubarrays(nums: Array[Int], k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_xor_subarrays(nums :: [integer], k :: integer) :: integer
def count_xor_subarrays(nums, k) do

end
end
```

Erlang:

```
-spec count_xor_subarrays(Nums :: [integer()], K :: integer()) -> integer().
count_xor_subarrays(Nums, K) ->
.
```

Racket:

```
(define/contract (count-xor-subarrays nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Subarrays with XOR at Least K
```

```
* Difficulty: Hard
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
long long countXorSubarrays(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Subarrays with XOR at Least K
* Difficulty: Hard
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public long countXorSubarrays(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Subarrays with XOR at Least K
Difficulty: Hard
Tags: array

Approach: Use two pointers or sliding window technique
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def countXorSubarrays(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countXorSubarrays(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Subarrays with XOR at Least K
* Difficulty: Hard
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var countXorSubarrays = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Subarrays with XOR at Least K
* Difficulty: Hard
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function countXorSubarrays(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Subarrays with XOR at Least K
* Difficulty: Hard
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public long CountXorSubarrays(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Subarrays with XOR at Least K
* Difficulty: Hard
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

long long countXorSubarrays(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Subarrays with XOR at Least K
// Difficulty: Hard
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func countXorSubarrays(nums []int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countXorSubarrays(nums: IntArray, k: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func countXorSubarrays(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Subarrays with XOR at Least K
// Difficulty: Hard
// Tags: array
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn count_xor_subarrays(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def count_xor_subarrays(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function countXorSubarrays($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int countXorSubarrays(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def countXorSubarrays(nums: Array[Int], k: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_xor_subarrays(nums :: [integer], k :: integer) :: integer
def count_xor_subarrays(nums, k) do

end
end
```

Erlang Solution:

```
-spec count_xor_subarrays(Nums :: [integer()], K :: integer()) -> integer().
count_xor_subarrays(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (count-xor-subarrays nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```