Problem 3513: Number of Unique XOR Triplets I

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

, where

nums

is a

permutation

of the numbers in the range

[1, n]

.

A

XOR triplet

is defined as the XOR of three elements

nums[i] XOR nums[j] XOR nums[k]

where

i <= j <= k

.

Return the number of

unique

XOR triplet values from all possible triplets

(i, j, k)

.

Example 1:

Input:

nums = [1,2]

Output:

2

Explanation:

The possible XOR triplet values are:

(0, 0, 0) → 1 XOR 1 XOR 1 = 1

(0, 0, 1) → 1 XOR 1 XOR 2 = 2

(0, 1, 1) → 1 XOR 2 XOR 2 = 1

(1, 1, 1) → 2 XOR 2 XOR 2 = 2

The unique XOR values are

{1, 2}

, so the output is 2.

Example 2:

Input:

nums = [3,1,2]

Output:

4

Explanation:

The possible XOR triplet values include:

(0, 0, 0) → 3 XOR 3 XOR 3 = 3

(0, 0, 1) → 3 XOR 3 XOR 1 = 1

(0, 0, 2) → 3 XOR 3 XOR 2 = 2

(0, 1, 2) → 3 XOR 1 XOR 2 = 0

The unique XOR values are

{0, 1, 2, 3}

, so the output is 4.

Constraints:

1 <= n == nums.length <= 10

5

1 <= nums[i] <= n

nums

is a permutation of integers from

1

to

n

.

Code Snippets

C++:

```
class Solution {
public:
int uniqueXorTriplets(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int uniqueXorTriplets(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function uniqueXorTriplets(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int UniqueXorTriplets(int[] nums) {

}
}
```

C:

```
int uniqueXorTriplets(int* nums, int numsSize) {

}
```

Go:

```
func uniqueXorTriplets(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun uniqueXorTriplets(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func uniqueXorTriplets(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn unique_xor_triplets(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def unique_xor_triplets(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
```

```
*/
function uniqueXorTriplets($nums) {

}
}
```

Dart:

```
class Solution {
int uniqueXorTriplets(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def uniqueXorTriplets(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec unique_xor_triplets(nums :: [integer]) :: integer
def unique_xor_triplets(nums) do

end
end
```

Erlang:

```
-spec unique_xor_triplets(Nums :: [integer()]) -> integer().
unique_xor_triplets(Nums) ->
.
```

Racket:

```
(define/contract (unique-xor-triplets nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Unique XOR Triplets I
* Difficulty: Medium
* Tags: array, math
*
* 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:
int uniqueXorTriplets(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Unique XOR Triplets I
* Difficulty: Medium
* Tags: array, math
*
* 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 int uniqueXorTriplets(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Unique XOR Triplets I
Difficulty: Medium
Tags: array, math

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 uniqueXorTriplets(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Unique XOR Triplets I
* Difficulty: Medium
* Tags: array, math
*
* 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
* @return {number}
*/
var uniqueXorTriplets = function(nums) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Number of Unique XOR Triplets I
* Difficulty: Medium
* Tags: array, math
*
* 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 uniqueXorTriplets(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Unique XOR Triplets I
* Difficulty: Medium
* Tags: array, math
*
* 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 int UniqueXorTriplets(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Number of Unique XOR Triplets I
* Difficulty: Medium
```

```
* Tags: array, math
*
* 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
*/

int uniqueXorTriplets(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Number of Unique XOR Triplets I
// Difficulty: Medium
// Tags: array, math
//
// 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 uniqueXorTriplets(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun uniqueXorTriplets(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func uniqueXorTriplets(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Unique XOR Triplets I
// Difficulty: Medium
// Tags: array, math
//
// 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 unique_xor_triplets(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def unique_xor_triplets(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function uniqueXorTriplets($nums) {

}
}
```

Dart Solution:

```
class Solution {
int uniqueXorTriplets(List<int> nums) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def uniqueXorTriplets(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec unique_xor_triplets(nums :: [integer]) :: integer
def unique_xor_triplets(nums) do

end
end
```

Erlang Solution:

```
-spec unique_xor_triplets(Nums :: [integer()]) -> integer().
unique_xor_triplets(Nums) ->
.
```

Racket Solution:

```
(define/contract (unique-xor-triplets nums)
(-> (listof exact-integer?) exact-integer?)
)
```