Problem 982: Triples with Bitwise AND Equal To Zero

Problem Information

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

Problem Description

Given an integer array nums, return

the number of

AND triples

.

An

AND triple

is a triple of indices

(i, j, k)

such that:

0 <= i < nums.length

0 <= j < nums.length

0 <= k < nums.length

nums[i] & nums[j] & nums[k] == 0

, where

&

represents the bitwise-AND operator.

Example 1:

Input:

nums = [2,1,3]

Output:

12

Explanation:

We could choose the following i, j, k triples: (i=0, j=0, k=1) : 2 & 2 & 1 (i=0, j=1, k=0) : 2 & 1 & 2 (i=0, j=1, k=1) : 2 & 1 & 1 (i=0, j=1, k=2) : 2 & 1 & 3 (i=0, j=2, k=1) : 2 & 3 & 1 (i=1, j=0, k=0) : 1 & 2 & 2 (i=1, j=0, k=1) : 1 & 2 & 1 (i=1, j=0, k=2) : 1 & 2 & 3 (i=1, j=1, k=0) : 1 & 1 & 2 (i=1, j=2, k=0) : 1 & 3 & 2 (i=2, j=0, k=1) : 3 & 2 & 1 (i=2, j=1, k=0) : 3 & 1 & 2

Example 2:

Input:

nums = [0,0,0]

Output:

27

Constraints:

1 <= nums.length <= 1000

0 <= nums[i] < 2

16

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @return {number}
*/
```

```
var countTriplets = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Triples with Bitwise AND Equal To Zero
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Triples with Bitwise AND Equal To Zero
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int countTriplets(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Triples with Bitwise AND Equal To Zero
Difficulty: Hard
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Triples with Bitwise AND Equal To Zero
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Triples with Bitwise AND Equal To Zero
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function countTriplets(nums: number[]): number {
};
```

C# Solution:

```
/*
* Problem: Triples with Bitwise AND Equal To Zero
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int CountTriplets(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Triples with Bitwise AND Equal To Zero
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int countTriplets(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Triples with Bitwise AND Equal To Zero
// Difficulty: Hard
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func countTriplets(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Triples with Bitwise AND Equal To Zero
// Difficulty: Hard
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

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