Problem 3583: Count Special Triplets

Problem Information

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

Problem Description

You are given an integer array

nums

.

A

special triplet

is defined as a triplet of indices

(i, j, k)

such that:

0 <= i < j < k < n

, where

n = nums.length

nums[i] == nums[j] * 2

nums[k] == nums[j] * 2

Return the total number of

special triplets

in the array.

Since the answer may be large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

nums = [6,3,6]

Output:

1

Explanation:

The only special triplet is

(i, j, k) = (0, 1, 2)

, where:

nums[0] = 6

,

nums[1] = 3

,

nums[2] = 6

nums[0] = nums[1] * 2 = 3 * 2 = 6

nums[2] = nums[1] * 2 = 3 * 2 = 6

Example 2:

Input:

nums = [0,1,0,0]

Output:

1

Explanation:

The only special triplet is

(i, j, k) = (0, 2, 3)

, where:

nums[0] = 0

,

nums[2] = 0

,

nums[3] = 0

nums[0] = nums[2] * 2 = 0 * 2 = 0

nums[3] = nums[2] * 2 = 0 * 2 = 0

Example 3:

Input:

nums = [8,4,2,8,4]

Output:

2

Explanation:

There are exactly two special triplets:

(i, j, k) = (0, 1, 3)

nums[0] = 8

,

nums[1] = 4

,

nums[3] = 8

nums[0] = nums[1] * 2 = 4 * 2 = 8

nums[3] = nums[1] * 2 = 4 * 2 = 8

(i, j, k) = (1, 2, 4)

nums[1] = 4

,

nums[2] = 2

,

nums[4] = 4

nums[1] = nums[2] * 2 = 2 * 2 = 4

nums[4] = nums[2] * 2 = 2 * 2 = 4

Constraints:

3 <= n == nums.length <= 10

5

0 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Special Triplets
* Difficulty: Medium
* 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 specialTriplets(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Special Triplets
* Difficulty: Medium
* 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 specialTriplets(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Count Special Triplets
Difficulty: Medium
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 specialTriplets(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Special Triplets
* Difficulty: Medium
* 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 specialTriplets = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Special Triplets
* Difficulty: Medium
* 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 specialTriplets(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Special Triplets
* Difficulty: Medium
* 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 SpecialTriplets(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Special Triplets
* Difficulty: Medium
* 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 specialTriplets(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Special Triplets
// Difficulty: Medium
// 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 specialTriplets(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Count Special Triplets
// Difficulty: Medium
// 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 special_triplets(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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