

Problem 2198: Number of Single Divisor Triplets

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

array of positive integers

nums

. A triplet of three

distinct

indices

(i, j, k)

is called a

single divisor triplet

of

nums

if

$\text{nums}[i] + \text{nums}[j] + \text{nums}[k]$

is divisible by

exactly one

of

$\text{nums}[i]$

,

$\text{nums}[j]$

, or

$\text{nums}[k]$

.

Return

the number of

single divisor triplets

of

nums

.

Example 1:

Input:

$\text{nums} = [4, 6, 7, 3, 2]$

Output:

12

Explanation:

The triplets $(0, 3, 4)$, $(0, 4, 3)$, $(3, 0, 4)$, $(3, 4, 0)$, $(4, 0, 3)$, and $(4, 3, 0)$ have the values of $[4, 3, 2]$ (or a permutation of $[4, 3, 2]$). $4 + 3 + 2 = 9$ which is only divisible by 3, so all such triplets are single divisor triplets. The triplets $(0, 2, 3)$, $(0, 3, 2)$, $(2, 0, 3)$, $(2, 3, 0)$, $(3, 0, 2)$, and $(3, 2, 0)$ have the values of $[4, 7, 3]$ (or a permutation of $[4, 7, 3]$). $4 + 7 + 3 = 14$ which is only divisible by 7, so all such triplets are single divisor triplets. There are 12 single divisor triplets in total.

Example 2:

Input:

nums = [1,2,2]

Output:

6

Explanation:

The triplets $(0, 1, 2)$, $(0, 2, 1)$, $(1, 0, 2)$, $(1, 2, 0)$, $(2, 0, 1)$, and $(2, 1, 0)$ have the values of $[1, 2, 2]$ (or a permutation of $[1, 2, 2]$). $1 + 2 + 2 = 5$ which is only divisible by 1, so all such triplets are single divisor triplets. There are 6 single divisor triplets in total.

Example 3:

Input:

nums = [1,1,1]

Output:

0

Explanation:

There are no single divisor triplets. Note that (0, 1, 2) is not a single divisor triplet because $\text{nums}[0] + \text{nums}[1] + \text{nums}[2] = 3$ and 3 is divisible by $\text{nums}[0]$, $\text{nums}[1]$, and $\text{nums}[2]$.

Constraints:

$3 \leq \text{nums.length} \leq 10$

5

$1 \leq \text{nums}[i] \leq 100$

Code Snippets

C++:

```
class Solution {
public:
    long long singleDivisorTriplet(vector<int>& nums) {
        ...
    }
};
```

Java:

```
class Solution {
    public long singleDivisorTriplet(int[] nums) {
        ...
    }
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function singleDivisorTriplet(nums: number[]): number {
}
```

C#:

```
public class Solution {
    public long SingleDivisorTriplet(int[] nums) {
}
```

C:

```
long long singleDivisorTriplet(int* nums, int numsSize) {
}
```

Go:

```
func singleDivisorTriplet(nums []int) int64 {
}
```

Kotlin:

```
class Solution {  
    fun singleDivisorTriplet(nums: IntArray): Long {  
  
    }  
}
```

Swift:

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

Rust:

```
impl Solution {  
    pub fn single_divisor_triplet(nums: Vec<i32>) -> i64 {  
  
    }  
}
```

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @return Integer  
     */  
    function singleDivisorTriplet($nums) {  
  
    }
```

```
}
```

Dart:

```
class Solution {  
    int singleDivisorTriplet(List<int> nums) {  
        }  
    }
```

Scala:

```
object Solution {  
    def singleDivisorTriplet(nums: Array[Int]): Long = {  
        }  
    }
```

Elixir:

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

Erlang:

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

Racket:

```
(define/contract (single-divisor-triplet nums)  
  (-> (listof exact-integer?) exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Number of Single Divisor Triplets
 * 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:
    long long singleDivisorTriplet(vector<int>& nums) {

    }
};
```

Java Solution:

```
/**
 * Problem: Number of Single Divisor Triplets
 * 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 long singleDivisorTriplet(int[] nums) {

    }
}
```

Python3 Solution:

```
"""
Problem: Number of Single Divisor Triplets
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 singleDivisorTriplet(self, nums: List[int]) -> int:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

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

```

JavaScript Solution:

```
/**
 * Problem: Number of Single Divisor Triplets
 * 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 singleDivisorTriplet = function(nums) {

};
```

TypeScript Solution:

```

/**
 * Problem: Number of Single Divisor Triplets
 * 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 singleDivisorTriplet(nums: number[]): number {
}

```

C# Solution:

```

/*
 * Problem: Number of Single Divisor Triplets
 * 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 long SingleDivisorTriplet(int[] nums) {
}
}

```

C Solution:

```

/*
 * Problem: Number of Single Divisor Triplets
 * 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

```

```
*/  
  
long long singleDivisorTriplet(int* nums, int numsSize) {  
  
}  

```

Go Solution:

```
// Problem: Number of Single Divisor Triplets  
// 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 singleDivisorTriplet(nums []int) int64 {  
  
}
```

Kotlin Solution:

```
class Solution {  
    fun singleDivisorTriplet(nums: IntArray): Long {  
  
    }  
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Number of Single Divisor Triplets  
// 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 single_divisor_triplet(nums: Vec<i32>) -> i64 {

}
}

```

Ruby Solution:

```

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

end

```

PHP Solution:

```

class Solution {

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

}
}

```

Dart Solution:

```

class Solution {
int singleDivisorTriplet(List<int> nums) {

}
}

```

Scala Solution:

```
object Solution {  
    def singleDivisorTriplet(nums: Array[Int]): Long = {  
        }  
    }  
}
```

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

```
(define/contract (single-divisor-triplet nums)  
  (-> (listof exact-integer?) exact-integer?)  
)
```