Problem 1726: Tuple with Same Product

Problem Information

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

Problem Description

Given an array

nums

of

distinct

positive integers, return

the number of tuples

(a, b, c, d)

such that

a * b = c * d

where

a

,

b

,

c

, and

d

are elements of

nums

, and

a != b != c != d

.

Example 1:

Input:

nums = [2,3,4,6]

Output:

8

Explanation:

There are 8 valid tuples: (2,6,3,4) , (2,6,4,3) , (6,2,3,4) , (6,2,4,3) (3,4,2,6) , (4,3,2,6) , (3,4,6,2) , (4,3,6,2)

Example 2:

Input:

nums = [1,2,4,5,10]

Output:

16

Explanation:

There are 16 valid tuples: (1,10,2,5) , (1,10,5,2) , (10,1,2,5) , (10,1,5,2) (2,5,1,10) , (2,5,10,1) , (5,2,1,10) , (5,2,10,1) (2,10,4,5) , (2,10,5,4) , (10,2,4,5) , (10,2,5,4) (4,5,2,10) , (4,5,10,2) , (5,4,2,10) , (5,4,10,2)

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 10

4

All elements in

nums

are

distinct

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (tuple-same-product nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Tuple with Same Product
* 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 tupleSameProduct(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Tuple with Same Product
* 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 tupleSameProduct(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Tuple with Same Product
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 tupleSameProduct(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Tuple with Same Product
* 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 tupleSameProduct = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Tuple with Same Product
* 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 tupleSameProduct(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Tuple with Same Product
* 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 TupleSameProduct(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Tuple with Same Product
* 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 tupleSameProduct(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Tuple with Same Product
// 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 tupleSameProduct(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Tuple with Same Product
// 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 tuple_same_product(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (tuple-same-product nums)
(-> (listof exact-integer?) exact-integer?)
)
```