Problem 3649: Number of Perfect Pairs

Problem Information

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

Problem Description

You are given an integer array

nums

.

A pair of indices

(i, j)

is called

perfect

if the following conditions are satisfied:

i < j

Let

a = nums[i]

,

b = nums[j]

. Then:

min(|a - b|, |a + b|) <= min(|a|, |b|)

max(|a - b|, |a + b|) >= max(|a|, |b|)

Return the number of

distinct

perfect pairs.

Note:

The absolute value

|x|

refers to the

non-negative

value of

x

.

Example 1:

Input:

nums = [0,1,2,3]

Output:

2

Explanation:

There are 2 perfect pairs:

(i, j)

(a, b)

min(|a – b|, |a + b|)

min(|a|, |b|)

max(|a – b|, |a + b|)

max(|a|, |b|)

(1, 2)

(1, 2)

min(|1 – 2|, |1 + 2|) = 1

1

max(|1 – 2|, |1 + 2|) = 3

2

(2, 3)

(2, 3)

min(|2 – 3|, |2 + 3|) = 1

2

max(|2 – 3|, |2 + 3|) = 5

3

Example 2:

Input:

nums = [-3,2,-1,4]

Output:

4

Explanation:

There are 4 perfect pairs:

(i, j)

(a, b)

min(|a – b|, |a + b|)

min(|a|, |b|)

max(|a – b|, |a + b|)

max(|a|, |b|)

(0, 1)

(-3, 2)

min(|-3 - 2|, |-3 + 2|) = 1

2

max(|-3 - 2|, |-3 + 2|) = 5

3

(0, 3)

(-3, 4)

$\min(|-3 - 4|, |-3 + 4|) = 1$

3

$\max(|-3 - 4|, |-3 + 4|) = 7$

4

(1, 2)

(2, -1)

$\min(|2 - (-1)|, |2 + (-1)|) = 1$

1

$\max(|2 - (-1)|, |2 + (-1)|) = 3$

2

(1, 3)

(2, 4)

$\min(|2 - 4|, |2 + 4|) = 2$

2

$\max(|2 - 4|, |2 + 4|) = 6$

4

Example 3:

Input:

nums = [1,10,100,1000]

Output:

0

Explanation:

There are no perfect pairs. Thus, the answer is 0.

Constraints:

2 <= nums.length <= 10

5

-10

9

<= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
long long perfectPairs(int* nums, int numsSize) {

}
```

Go:

```
func perfectPairs(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun perfectPairs(nums: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (perfect-pairs nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Perfect Pairs
* Difficulty: Medium
* Tags: array, math, sort
*
* 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 perfectPairs(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Perfect Pairs
* Difficulty: Medium
* Tags: array, math, sort
*
* 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 perfectPairs(int[] nums) {
}
}
```

Python3 Solution:

```
"""
Problem: Number of Perfect Pairs
Difficulty: Medium
Tags: array, math, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Perfect Pairs
* Difficulty: Medium
* Tags: array, math, sort
*
* 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 perfectPairs = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Perfect Pairs
* Difficulty: Medium
* Tags: array, math, sort
*
* 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 perfectPairs(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Perfect Pairs
* Difficulty: Medium
* Tags: array, math, sort
*
* 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 PerfectPairs(int[] nums) {

}
}
```

C Solution:

```c
/*
* Problem: Number of Perfect Pairs
* Difficulty: Medium
* Tags: array, math, sort
*
* 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 perfectPairs(int* nums, int numsSize) {

}
```

Go Solution:

```go
// Problem: Number of Perfect Pairs
// Difficulty: Medium
// Tags: array, math, sort
//
// 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 perfectPairs(nums []int) int64 {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun perfectPairs(nums: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Perfect Pairs
// Difficulty: Medium
// Tags: array, math, sort
//
// 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 perfect_pairs(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (perfect-pairs nums)
(-> (listof exact-integer?) exact-integer?)
)
```