Problem 3184: Count Pairs That Form a Complete Day I

Problem Information

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

Problem Description

Given an integer array

hours

representing times in

hours

, return an integer denoting the number of pairs

i

,

j

where

i < j

and

hours[i] + hours[j]

forms a

complete day

.

A

complete day

is defined as a time duration that is an

exact

multiple

of 24 hours.

For example, 1 day is 24 hours, 2 days is 48 hours, 3 days is 72 hours, and so on.

Example 1:

Input:

hours = [12,12,30,24,24]

Output:

2

Explanation:

The pairs of indices that form a complete day are

(0, 1)

and

(3, 4)

.

Example 2:

Input:

hours = [72,48,24,3]

Output:

3

Explanation:

The pairs of indices that form a complete day are

(0, 1)

,

(0, 2)

, and

(1, 2)

.

Constraints:

1 <= hours.length <= 100

1 <= hours[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int countCompleteDayPairs(vector<int>& hours) {

}
};
```

Java:

```
class Solution {
public int countCompleteDayPairs(int[] hours) {

}
}
```

Python3:

```
class Solution:
def countCompleteDayPairs(self, hours: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countCompleteDayPairs(hours: number[]): number {

};
```

C#:

```
public class Solution {
public int CountCompleteDayPairs(int[] hours) {

}
}
```

C:

```
int countCompleteDayPairs(int* hours, int hoursSize) {

}
```

Go:

```
func countCompleteDayPairs(hours []int) int {

}
```

Kotlin:

```
class Solution {
fun countCompleteDayPairs(hours: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func countCompleteDayPairs(_ hours: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_complete_day_pairs(hours: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} hours
# @return {Integer}
def count_complete_day_pairs(hours)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $hours
* @return Integer
*/
function countCompleteDayPairs($hours) {

}
}
```

Dart:

```
class Solution {
int countCompleteDayPairs(List<int> hours) {

}
}
```

Scala:

```
object Solution {
def countCompleteDayPairs(hours: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_complete_day_pairs(hours :: [integer]) :: integer
def count_complete_day_pairs(hours) do

end
end
```

Erlang:

```
-spec count_complete_day_pairs(Hours :: [integer()]) -> integer().
count_complete_day_pairs(Hours) ->
.
```

Racket:

```
(define/contract (count-complete-day-pairs hours)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Pairs That Form a Complete Day I
* Difficulty: Easy
* 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 countCompleteDayPairs(vector<int>& hours) {

}
};
```

Java Solution:

```
/**
* Problem: Count Pairs That Form a Complete Day I
* Difficulty: Easy
* 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 countCompleteDayPairs(int[] hours) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Pairs That Form a Complete Day I
Difficulty: Easy
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 countCompleteDayPairs(self, hours: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Count Pairs That Form a Complete Day I
* Difficulty: Easy
* 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[]} hours
* @return {number}
*/
var countCompleteDayPairs = function(hours) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Pairs That Form a Complete Day I
* Difficulty: Easy
* 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 countCompleteDayPairs(hours: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Pairs That Form a Complete Day I
* Difficulty: Easy
* 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 CountCompleteDayPairs(int[] hours) {

}
}
```

C Solution:

```
/*
* Problem: Count Pairs That Form a Complete Day I
* Difficulty: Easy
* 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 countCompleteDayPairs(int* hours, int hoursSize) {

}
```

Go Solution:

```
// Problem: Count Pairs That Form a Complete Day I
// Difficulty: Easy
// 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 countCompleteDayPairs(hours []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countCompleteDayPairs(hours: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func countCompleteDayPairs(_ hours: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Pairs That Form a Complete Day I
// Difficulty: Easy
// 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_complete_day_pairs(hours: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} hours
# @return {Integer}
def count_complete_day_pairs(hours)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $hours
* @return Integer
*/
function countCompleteDayPairs($hours) {

}
}
```

Dart Solution:

```
class Solution {
int countCompleteDayPairs(List<int> hours) {

}
}
```

Scala Solution:

```
object Solution {
def countCompleteDayPairs(hours: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_complete_day_pairs(hours :: [integer]) :: integer
def count_complete_day_pairs(hours) do

end
end
```

Erlang Solution:

```
-spec count_complete_day_pairs(Hours :: [integer()]) -> integer().
count_complete_day_pairs(Hours) ->
.
```

Racket Solution:

```
(define/contract (count-complete-day-pairs hours)
(-> (listof exact-integer?) exact-integer?)
)
```