Problem 1862: Sum of Floored Pairs

Problem Information

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

Problem Description

Given an integer array

nums

, return the sum of

floor(nums[i] / nums[j])

for all pairs of indices

0 <= i, j < nums.length

in the array. Since the answer may be too large, return it

modulo

10

9

+ 7

.

The

floor()

function returns the integer part of the division.

Example 1:

Input:

nums = [2,5,9]

Output:

10

Explanation:

floor(2 / 5) = floor(2 / 9) = floor(5 / 9) = 0 floor(2 / 2) = floor(5 / 5) = floor(9 / 9) = 1 floor(5 / 2) = 2 floor(9 / 2) = 4 floor(9 / 5) = 1 We calculate the floor of the division for every pair of indices in the array then sum them up.

Example 2:

Input:

nums = [7,7,7,7,7,7,7]

Output:

49

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sum-of-floored-pairs nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Floored Pairs
* Difficulty: Hard
* Tags: array, math, search
*
* 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:
int sumOfFlooredPairs(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Floored Pairs
* Difficulty: Hard
* Tags: array, math, search
*
* 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 int sumOfFlooredPairs(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Floored Pairs
Difficulty: Hard
Tags: array, math, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Floored Pairs
* Difficulty: Hard
* Tags: array, math, search
*
* 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 sumOfFlooredPairs = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Floored Pairs
* Difficulty: Hard
* Tags: array, math, search
*
* 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 sumOfFlooredPairs(nums: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Sum of Floored Pairs
* Difficulty: Hard
* Tags: array, math, search
*
* 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 int SumOfFlooredPairs(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Floored Pairs
* Difficulty: Hard
* Tags: array, math, search
*
* 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
*/

int sumOfFlooredPairs(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sum of Floored Pairs
// Difficulty: Hard
```

```
// Tags: array, math, search
//
// 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 sumOfFlooredPairs(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Floored Pairs
// Difficulty: Hard
// Tags: array, math, search
//
// 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 sum_of_floored_pairs(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sum-of-floored-pairs nums)
(-> (listof exact-integer?) exact-integer?)
)
```