Problem 2778: Sum of Squares of Special Elements

Problem Information

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

Problem Description

You are given a

1-indexed

integer array

nums

of length

n

.

An element

nums[i]

of

nums

is called

special

if

i

divides

n

, i.e.

n % i == 0

.

Return

the

sum of the squares

of all

special

elements of

nums

.

Example 1:

Input:

nums = [1,2,3,4]

Output:

21

Explanation:

There are exactly 3 special elements in nums: nums[1] since 1 divides 4, nums[2] since 2 divides 4, and nums[4] since 4 divides 4. Hence, the sum of the squares of all special elements of nums is nums[1] * nums[1] + nums[2] * nums[2] + nums[4] * nums[4] = 1 * 1 + 2 * 2 + 4 * 4 = 21.

Example 2:

Input:

nums = [2,7,1,19,18,3]

Output:

63

Explanation:

There are exactly 4 special elements in nums: nums[1] since 1 divides 6, nums[2] since 2 divides 6, nums[3] since 3 divides 6, and nums[6] since 6 divides 6. Hence, the sum of the squares of all special elements of nums is nums[1] * nums[1] + nums[2] * nums[2] + nums[3] * nums[3] + nums[6] * nums[6] = 2 * 2 + 7 * 7 + 1 * 1 + 3 * 3 = 63.

Constraints:

1 <= nums.length == n <= 50

1 <= nums[i] <= 50

Code Snippets

C++:

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

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Sum of Squares of Special Elements
* Difficulty: Easy
* Tags: array
*
* 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 sumOfSquares(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Squares of Special Elements
* Difficulty: Easy
* Tags: array
*
* 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 sumOfSquares(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Squares of Special Elements
Difficulty: Easy
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Squares of Special Elements
* Difficulty: Easy
* Tags: array
*
* 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 sumOfSquares = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Squares of Special Elements
* Difficulty: Easy
* Tags: array
*
* 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 sumOfSquares(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Squares of Special Elements
* Difficulty: Easy
* Tags: array
*
* 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 SumOfSquares(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Squares of Special Elements
* Difficulty: Easy
* Tags: array
*
* 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 sumOfSquares(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sum of Squares of Special Elements
// Difficulty: Easy
// Tags: array
//
// 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 sumOfSquares(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Squares of Special Elements
// Difficulty: Easy
// Tags: array
//
// 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_squares(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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