Problem 3712: Sum of Elements With Frequency Divisible by K

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

.

Return an integer denoting the

sum

of all elements in

nums

whose

frequency

is divisible by

k

, or 0 if there are no such elements.

Note:

An element is included in the sum

exactly

as many times as it appears in the array if its total frequency is divisible by

k

.

Example 1:

Input:

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

Output:

16

Explanation:

The number 1 appears once (odd frequency).

The number 2 appears twice (even frequency).

The number 3 appears four times (even frequency).

The number 4 appears once (odd frequency).

So, the total sum is

2 + 2 + 3 + 3 + 3 + 3 = 16

.

Example 2:

Input:

nums = [1,2,3,4,5], k = 2

Output:

0

Explanation:

There are no elements that appear an even number of times, so the total sum is 0.

Example 3:

Input:

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

Output:

12

Explanation:

The number 1 appears once.

The number 2 appears once.

The number 3 appears once.

The number 4 appears three times.

So, the total sum is

4 + 4 + 4 = 12

.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

1 <= k <= 100

Code Snippets

C++:

```
class Solution {
public:
int sumDivisibleByK(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int sumDivisibleByK(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

```
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function sumDivisibleByK(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int SumDivisibleByK(int[] nums, int k) {

}
}
```

C:

```
int sumDivisibleByK(int* nums, int numsSize, int k) {

}
```

Go:

```
func sumDivisibleByK(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun sumDivisibleByK(nums: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func sumDivisibleByK(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn sum_divisible_by_k(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def sum_divisible_by_k(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function sumDivisibleByK($nums, $k) {

}
```

```
}
```

Dart:

```
class Solution {
int sumDivisibleByK(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def sumDivisibleByK(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sum_divisible_by_k(nums :: [integer], k :: integer) :: integer
def sum_divisible_by_k(nums, k) do

end
end
```

Erlang:

```
-spec sum_divisible_by_k(Nums :: [integer()], K :: integer()) -> integer().
sum_divisible_by_k(Nums, K) ->
.
```

Racket:

```
(define/contract (sum-divisible-by-k nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Elements With Frequency Divisible by K
* 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 sumDivisibleByK(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Elements With Frequency Divisible by K
* 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 sumDivisibleByK(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Elements With Frequency Divisible by K
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 sumDivisibleByK(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Elements With Frequency Divisible by K
* 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[]} nums
* @param {number} k
* @return {number}
*/
var sumDivisibleByK = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Elements With Frequency Divisible by K
* 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 sumDivisibleByK(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Elements With Frequency Divisible by K
* 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 SumDivisibleByK(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Elements With Frequency Divisible by K
* 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 sumDivisibleByK(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Sum of Elements With Frequency Divisible by K
// 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 sumDivisibleByK(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun sumDivisibleByK(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func sumDivisibleByK(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Sum of Elements With Frequency Divisible by K
// 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 sum_divisible_by_k(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def sum_divisible_by_k(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function sumDivisibleByK($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int sumDivisibleByK(List<int> nums, int k) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def sumDivisibleByK(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sum_divisible_by_k(nums :: [integer], k :: integer) :: integer
def sum_divisible_by_k(nums, k) do

end
end
```

Erlang Solution:

```
-spec sum_divisible_by_k(Nums :: [integer()], K :: integer()) -> integer().
sum_divisible_by_k(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (sum-divisible-by-k nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```