Problem 3718: Smallest Missing Multiple of K

Problem Information

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

Problem Description

Given an integer array

nums

and an integer

k

, return the

smallest positive multiple

of

k

that is

missing

from

nums

.

A

multiple

of

k

is any positive integer divisible by

k

.

Example 1:

Input:

nums = [8,2,3,4,6], k = 2

Output:

10

Explanation:

The multiples of

k = 2

are 2, 4, 6, 8, 10, 12... and the smallest multiple missing from

nums

is 10.

Example 2:

Input:

nums = [1,4,7,10,15], k = 5

Output:

5

Explanation:

The multiples of

k = 5

are 5, 10, 15, 20... and the smallest multiple missing from

nums

is 5.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

1 <= k <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (missing-multiple nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Missing Multiple of 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 missingMultiple(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest Missing Multiple of 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 missingMultiple(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Missing Multiple of 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 missingMultiple(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Smallest Missing Multiple of 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 missingMultiple = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Missing Multiple of 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 missingMultiple(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Smallest Missing Multiple of 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 MissingMultiple(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Missing Multiple of 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 missingMultiple(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Smallest Missing Multiple of 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 missingMultiple(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Smallest Missing Multiple of 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 missing_multiple(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (missing-multiple nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```