Problem 2644: Find the Maximum Divisibility Score

Problem Information

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

Problem Description

You are given two integer arrays

nums

and

divisors

.

The

divisibility score

of

divisors[i]

is the number of indices

j

such that

nums[j]

is divisible by

divisors[i]

.

Return the integer

divisors[i]

with the

maximum

divisibility score. If multiple integers have the maximum score, return the smallest one.

Example 1:

Input:

nums = [2,9,15,50], divisors = [5,3,7,2]

Output:

2

Explanation:

The divisibility score of

divisors[0]

is 2 since

nums[2]

and

nums[3]

are divisible by 5.

The divisibility score of

divisors[1]

is 2 since

nums[1]

and

nums[2]

are divisible by 3.

The divisibility score of

divisors[2]

is 0 since none of the numbers in

nums

is divisible by 7.

The divisibility score of

divisors[3]

is 2 since

nums[0]

and

nums[3]

are divisible by 2.

As

divisors[0]

,

divisors[1]

, and

divisors[3]

have the same divisibility score, we return the smaller one which is

divisors[3]

.

Example 2:

Input:

nums = [4,7,9,3,9], divisors = [5,2,3]

Output:

3

Explanation:

The divisibility score of

divisors[0]

is 0 since none of numbers in

nums

is divisible by 5.

The divisibility score of

divisors[1]

is 1 since only

nums[0]

is divisible by 2.

The divisibility score of

divisors[2]

is 3 since

nums[2]

,

nums[3]

and

nums[4]

are divisible by 3.

Example 3:

Input:

nums = [20,14,21,10], divisors = [10,16,20]

Output:

10

Explanation:

The divisibility score of

divisors[0]

is 2 since

nums[0]

and

nums[3]

are divisible by 10.

The divisibility score of

divisors[1]

is 0 since none of the numbers in

nums

is divisible by 16.

The divisibility score of

divisors[2]

is 1 since

nums[0]

is divisible by 20.

Constraints:

1 <= nums.length, divisors.length <= 1000

1 <= nums[i], divisors[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxDivScore(vector<int>& nums, vector<int>& divisors) {

}
};
```

Java:

```
class Solution {
public int maxDivScore(int[] nums, int[] divisors) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxDivScore(nums: number[], divisors: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxDivScore(int[] nums, int[] divisors) {

}
}
```

C:

```
int maxDivScore(int* nums, int numsSize, int* divisors, int divisorsSize) {

}
```

Go:

```
func maxDivScore(nums []int, divisors []int) int {

}
```

Kotlin:

```
class Solution {
fun maxDivScore(nums: IntArray, divisors: IntArray): Int {

}
```

```
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} divisors
# @return {Integer}
def max_div_score(nums, divisors)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $divisors
* @return Integer
*/
function maxDivScore($nums, $divisors) {

}
}
```

Dart:

```
class Solution {
int maxDivScore(List<int> nums, List<int> divisors) {

}
}
```

Scala:

```
object Solution {
def maxDivScore(nums: Array[Int], divisors: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_div_score(nums :: [integer], divisors :: [integer]) :: integer
def max_div_score(nums, divisors) do

end
end
```

Erlang:

```
-spec max_div_score(Nums :: [integer()], Divisors :: [integer()]) ->
integer().
max_div_score(Nums, Divisors) ->
.
```

Racket:

```
(define/contract (max-div-score nums divisors)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Maximum Divisibility Score
* 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 maxDivScore(vector<int>& nums, vector<int>& divisors) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Maximum Divisibility Score
* 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 maxDivScore(int[] nums, int[] divisors) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Maximum Divisibility Score
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 maxDivScore(self, nums: List[int], divisors: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Maximum Divisibility Score
* 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
* @param {number[]} divisors
* @return {number}
*/
var maxDivScore = function(nums, divisors) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Maximum Divisibility Score
* 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 maxDivScore(nums: number[], divisors: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Find the Maximum Divisibility Score
* 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 MaxDivScore(int[] nums, int[] divisors) {

}
}
```

C Solution:

```
/*
* Problem: Find the Maximum Divisibility Score
* 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 maxDivScore(int* nums, int numsSize, int* divisors, int divisorsSize) {

}
```

Go Solution:

```
// Problem: Find the Maximum Divisibility Score
// 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 maxDivScore(nums []int, divisors []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxDivScore(nums: IntArray, divisors: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Maximum Divisibility Score
// 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 max_div_score(nums: Vec<i32>, divisors: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} divisors
# @return {Integer}
def max_div_score(nums, divisors)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $divisors
* @return Integer
*/
function maxDivScore($nums, $divisors) {

}
}
```

Dart Solution:

```
class Solution {
int maxDivScore(List<int> nums, List<int> divisors) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def maxDivScore(nums: Array[Int], divisors: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_div_score(nums :: [integer], divisors :: [integer]) :: integer
def max_div_score(nums, divisors) do

end
end
```

Erlang Solution:

```
-spec max_div_score(Nums :: [integer()], Divisors :: [integer()]) ->
integer().
max_div_score(Nums, Divisors) ->
.
```

Racket Solution:

```
(define/contract (max-div-score nums divisors)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```