Problem 2344: Minimum Deletions to Make Array Divisible

Problem Information

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

Problem Description

You are given two positive integer arrays

nums

and

numsDivide

. You can delete any number of elements from

nums

.

Return

the

minimum

number of deletions such that the

smallest

element in

nums

divides

all the elements of

numsDivide

. If this is not possible, return

-1

.

Note that an integer

x

divides

y

if

y % x == 0

.

Example 1:

Input:

nums = [2,3,2,4,3], numsDivide = [9,6,9,3,15]

Output:

2

Explanation:

The smallest element in [2,3,2,4,3] is 2, which does not divide all the elements of numsDivide. We use 2 deletions to delete the elements in nums that are equal to 2 which makes nums = [3,4,3]. The smallest element in [3,4,3] is 3, which divides all the elements of numsDivide. It can be shown that 2 is the minimum number of deletions needed.

Example 2:

Input:

nums = [4,3,6], numsDivide = [8,2,6,10]

Output:

-1

Explanation:

We want the smallest element in nums to divide all the elements of numsDivide. There is no way to delete elements from nums to allow this.

Constraints:

1 <= nums.length, numsDivide.length <= 10

5

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

9

Code Snippets

C++:

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

Java:

```
class Solution {
public int minOperations(int[] nums, int[] numsDivide) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minOperations(nums: number[], numsDivide: number[]): number {
};
```

C#:

```
public class Solution {
public int MinOperations(int[] nums, int[] numsDivide) {

}
}
```

C:

```
int minOperations(int* nums, int numsSize, int* numsDivide, int
numsDivideSize) {

}
```

Go:

```
func minOperations(nums []int, numsDivide []int) int {

}
```

Kotlin:

```
class Solution {
fun minOperations(nums: IntArray, numsDivide: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} nums_divide
# @return {Integer}
def min_operations(nums, nums_divide)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $numsDivide
* @return Integer
*/
function minOperations($nums, $numsDivide) {

}
}
```

Dart:

```
class Solution {
int minOperations(List<int> nums, List<int> numsDivide) {

}
}
```

Scala:

```
object Solution {
def minOperations(nums: Array[Int], numsDivide: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations(nums :: [integer], nums_divide :: [integer]) :: integer
def min_operations(nums, nums_divide) do

end
end
```

Erlang:

```
-spec min_operations(Nums :: [integer()], NumsDivide :: [integer()]) ->
integer().
min_operations(Nums, NumsDivide) ->
.
```

Racket:

```
(define/contract (min-operations nums numsDivide)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Deletions to Make Array Divisible
* Difficulty: Hard
* Tags: array, math, sort, queue, heap
*
* 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 minOperations(vector<int>& nums, vector<int>& numsDivide) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Deletions to Make Array Divisible
* Difficulty: Hard
* Tags: array, math, sort, queue, heap
*
* 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 minOperations(int[] nums, int[] numsDivide) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Deletions to Make Array Divisible
Difficulty: Hard
Tags: array, math, sort, queue, heap

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Deletions to Make Array Divisible
* Difficulty: Hard
* Tags: array, math, sort, queue, heap
*
* 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[]} numsDivide
* @return {number}
*/
var minOperations = function(nums, numsDivide) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Deletions to Make Array Divisible
* Difficulty: Hard
* Tags: array, math, sort, queue, heap
*
* 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 minOperations(nums: number[], numsDivide: number[]): number {
};
```

C# Solution:

```
/*
* Problem: Minimum Deletions to Make Array Divisible
* Difficulty: Hard
* Tags: array, math, sort, queue, heap
*
* 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 MinOperations(int[] nums, int[] numsDivide) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Deletions to Make Array Divisible
* Difficulty: Hard
* Tags: array, math, sort, queue, heap
*
* 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 minOperations(int* nums, int numsSize, int* numsDivide, int
numsDivideSize) {

}
```

Go Solution:

```
// Problem: Minimum Deletions to Make Array Divisible
// Difficulty: Hard
// Tags: array, math, sort, queue, heap
//
// 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 minOperations(nums []int, numsDivide []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minOperations(nums: IntArray, numsDivide: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Deletions to Make Array Divisible
// Difficulty: Hard
// Tags: array, math, sort, queue, heap
//
// 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 min_operations(nums: Vec<i32>, nums_divide: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} nums_divide
# @return {Integer}
def min_operations(nums, nums_divide)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $numsDivide
* @return Integer
*/
function minOperations($nums, $numsDivide) {

}
}
```

Dart Solution:

```
class Solution {
int minOperations(List<int> nums, List<int> numsDivide) {

}
}
```

Scala Solution:

```
object Solution {
def minOperations(nums: Array[Int], numsDivide: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations(nums :: [integer], nums_divide :: [integer]) :: integer
def min_operations(nums, nums_divide) do

end
end
```

Erlang Solution:

```
-spec min_operations(Nums :: [integer()], NumsDivide :: [integer()]) ->
integer().
min_operations(Nums, NumsDivide) ->
.
```

Racket Solution:

```
(define/contract (min-operations nums numsDivide)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```