Problem 1590: Make Sum Divisible by P

Problem Information

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

Problem Description

Given an array of positive integers

nums

, remove the

smallest

subarray (possibly

empty

) such that the

sum

of the remaining elements is divisible by

p

. It is

not

allowed to remove the whole array.

Return

the length of the smallest subarray that you need to remove, or

-1

if it's impossible

.

A

subarray

is defined as a contiguous block of elements in the array.

Example 1:

Input:

nums = [3,1,4,2], p = 6

Output:

1

Explanation:

The sum of the elements in nums is 10, which is not divisible by 6. We can remove the subarray [4], and the sum of the remaining elements is 6, which is divisible by 6.

Example 2:

Input:

nums = [6,3,5,2], p = 9

Output:

2

Explanation:

We cannot remove a single element to get a sum divisible by 9. The best way is to remove the subarray [5,2], leaving us with [6,3] with sum 9.

Example 3:

Input:

nums = [1,2,3], p = 3

Output:

0

Explanation:

Here the sum is 6. which is already divisible by 3. Thus we do not need to remove anything.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

1 <= p <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minSubarray(vector<int>& nums, int p) {

}
};
```

Java:

```
class Solution {
public int minSubarray(int[] nums, int p) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minSubarray(nums: number[], p: number): number {

};
```

C#:

```
public class Solution {
public int MinSubarray(int[] nums, int p) {

}
}
```

C:

```
int minSubarray(int* nums, int numsSize, int p) {

}
```

Go:

```
func minSubarray(nums []int, p int) int {

}
```

Kotlin:

```
class Solution {
fun minSubarray(nums: IntArray, p: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} p
# @return {Integer}
def min_subarray(nums, p)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $p
* @return Integer
*/
function minSubarray($nums, $p) {

}
}
```

Dart:

```
class Solution {
int minSubarray(List<int> nums, int p) {

}
}
```

Scala:

```
object Solution {
def minSubarray(nums: Array[Int], p: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec min_subarray(nums :: [integer], p :: integer) :: integer
def min_subarray(nums, p) do

end
end
```

Erlang:

```
-spec min_subarray(Nums :: [integer()], P :: integer()) -> integer().
min_subarray(Nums, P) ->
.
```

Racket:

```
(define/contract (min-subarray nums p)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Make Sum Divisible by P
* Difficulty: Medium
* 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 minSubarray(vector<int>& nums, int p) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Make Sum Divisible by P
* Difficulty: Medium
* 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 minSubarray(int[] nums, int p) {

}
}
```

Python3 Solution:

```
"""
Problem: Make Sum Divisible by P
Difficulty: Medium
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 minSubarray(self, nums: List[int], p: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Make Sum Divisible by P
* Difficulty: Medium
* 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} p
* @return {number}
*/
var minSubarray = function(nums, p) {

};
```

TypeScript Solution:

```
/**
* Problem: Make Sum Divisible by P
* Difficulty: Medium
* 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 minSubarray(nums: number[], p: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Make Sum Divisible by P
* Difficulty: Medium
* 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 MinSubarray(int[] nums, int p) {

}
}
```

C Solution:

```
/*
* Problem: Make Sum Divisible by P
* Difficulty: Medium
* 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 minSubarray(int* nums, int numsSize, int p) {

}
```

Go Solution:

```
// Problem: Make Sum Divisible by P
// Difficulty: Medium
```

```
// 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 minSubarray(nums []int, p int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minSubarray(nums: IntArray, p: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Make Sum Divisible by P
// Difficulty: Medium
// 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 min_subarray(nums: Vec<i32>, p: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} p
# @return {Integer}
def min_subarray(nums, p)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $p
* @return Integer
*/
function minSubarray($nums, $p) {

}
}
```

Dart Solution:

```
class Solution {
int minSubarray(List<int> nums, int p) {

}
}
```

Scala Solution:

```
object Solution {
def minSubarray(nums: Array[Int], p: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_subarray(nums :: [integer], p :: integer) :: integer
def min_subarray(nums, p) do

end
end
```

Erlang Solution:

```
-spec min_subarray(Nums :: [integer()], P :: integer()) -> integer().
min_subarray(Nums, P) ->
.
```

Racket Solution:

```
(define/contract (min-subarray nums p)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```