# Problem 3512: Minimum Operations to Make Array Sum 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

. You can perform the following operation any number of times:

Select an index

i

and replace

nums[i]

with

nums[i] - 1

.

Return the

minimum

number of operations required to make the sum of the array divisible by

k

.

Example 1:

Input:

nums = [3,9,7], k = 5

Output:

4

Explanation:

Perform 4 operations on

nums[1] = 9

. Now,

nums = [3, 5, 7]

.

The sum is 15, which is divisible by 5.

Example 2:

Input:

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

Output:

0

Explanation:

The sum is 8, which is already divisible by 4. Hence, no operations are needed.

Example 3:

Input:

nums = [3,2], k = 6

Output:

5

Explanation:

Perform 3 operations on

nums[0] = 3

and 2 operations on

nums[1] = 2

. Now,

nums = [0, 0]

.

The sum is 0, which is divisible by 6.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 1000

1 <= k <= 100

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

```
}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

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

## Solutions

### C++ Solution:

```
/*
* Problem: Minimum Operations to Make Array Sum Divisible by K
* Difficulty: Easy
```

```
* Tags: array, math
*
* 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, int k) {

}
};
```

### Java Solution:

```
/**
* Problem: Minimum Operations to Make Array Sum Divisible by K
* Difficulty: Easy
* Tags: array, math
*
* 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 k) {

}
}
```

### Python3 Solution:

```
"""
Problem: Minimum Operations to Make Array Sum Divisible by K
Difficulty: Easy
Tags: array, math

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], k: int) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make Array Sum Divisible by K
* Difficulty: Easy
* Tags: array, math
*
* 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} k
* @return {number}
*/
var minOperations = function(nums, k) {

};
```

### TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make Array Sum Divisible by K
* Difficulty: Easy
* Tags: array, math
*
* 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[], k: number): number {

};
```

### C# Solution:

```
/*
* Problem: Minimum Operations to Make Array Sum Divisible by K
* Difficulty: Easy
* Tags: array, math
*
* 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 k) {

}
}
```

### C Solution:

```
/*
* Problem: Minimum Operations to Make Array Sum Divisible by K
* Difficulty: Easy
* Tags: array, math
*
* 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 k) {

}
```

### Go Solution:

```
// Problem: Minimum Operations to Make Array Sum Divisible by K
// Difficulty: Easy
// Tags: array, math
//
// 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, k int) int {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Minimum Operations to Make Array Sum Divisible by K
// Difficulty: Easy
// Tags: array, math
```

```
//
// 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>, k: i32) -> i32 {

}
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

}
}
```

### Dart Solution:

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

}
}
```

### Scala Solution:

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

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

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