Problem 3107: Minimum Operations to Make Median of Array Equal to K

Problem Information

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

Problem Description

You are given an integer array

nums

and a

non-negative

integer

k

. In one operation, you can increase or decrease any element by 1.

Return the

minimum

number of operations needed to make the

median

of

nums

equal

to

k

.

The median of an array is defined as the middle element of the array when it is sorted in non-decreasing order. If there are two choices for a median, the larger of the two values is taken.

Example 1:

Input:

nums = [2,5,6,8,5], k = 4

Output:

2

Explanation:

We can subtract one from

nums[1]

and

nums[4]

to obtain

[2, 4, 6, 8, 4]

. The median of the resulting array is equal to

k

.

Example 2:

Input:

nums = [2,5,6,8,5], k = 7

Output:

3

Explanation:

We can add one to

nums[1]

twice and add one to

nums[2]

once to obtain

[2, 7, 7, 8, 5]

.

Example 3:

Input:

nums = [1,2,3,4,5,6], k = 4

Output:

0

Explanation:

The median of the array is already equal to

k

.

Constraints:

1 <= nums.length <= 2 * 10

5

1 <= nums[i] <= 10

9

1 <= k <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
long long minOperationsToMakeMedianK(int* nums, int numsSize, int k) {

}
```

Go:

```
func minOperationsToMakeMedianK(nums []int, k int) int64 {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn min_operations_to_make_median_k(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby:

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

```
end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make Median of Array Equal to K
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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:
long long minOperationsToMakeMedianK(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make Median of Array Equal to K
* Difficulty: Medium
* Tags: array, greedy, sort
*
```

```
* 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 long minOperationsToMakeMedianK(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make Median of Array Equal to K
Difficulty: Medium
Tags: array, greedy, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make Median of Array Equal to K
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minOperationsToMakeMedianK = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make Median of Array Equal to K
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minOperationsToMakeMedianK(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make Median of Array Equal to K
* Difficulty: Medium
* Tags: array, greedy, sort
*
```

```
* 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 long MinOperationsToMakeMedianK(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make Median of Array Equal to K
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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
*/

long long minOperationsToMakeMedianK(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Make Median of Array Equal to K
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 minOperationsToMakeMedianK(nums []int, k int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make Median of Array Equal to K
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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_to_make_median_k(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

```
min_operations_to_make_median_k(Nums, K) ->
.
```

Racket Solution:

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