Problem 3452: Sum of Good Numbers

Problem Information

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

Problem Description

Given an array of integers

nums

and an integer

k

, an element

nums[i]

is considered

good

if it is

strictly

greater than the elements at indices

i - k

and

i + k

(if those indices exist). If neither of these indices

exists

,

nums[i]

is still considered

good

.

Return the

sum

of all the

good

elements in the array.

Example 1:

Input:

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

Output:

12

Explanation:

The good numbers are

nums[1] = 3

,

nums[4] = 5

, and

nums[5] = 4

because they are strictly greater than the numbers at indices

i - k

and

i + k

.

Example 2:

Input:

nums = [2,1], k = 1

Output:

2

Explanation:

The only good number is

nums[0] = 2

because it is strictly greater than

nums[1]

.

Constraints:

2 <= nums.length <= 100

1 <= nums[i] <= 1000

1 <= k <= floor(nums.length / 2)

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def sumOfGoodNumbers(self, nums, k):
```

```
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

```
}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sum-of-good-numbers nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Good Numbers
* 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 sumOfGoodNumbers(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Good Numbers
* 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 sumOfGoodNumbers(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Good Numbers
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Good Numbers
* 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} k
* @return {number}
*/
var sumOfGoodNumbers = function(nums, k) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Sum of Good Numbers
* 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 sumOfGoodNumbers(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Good Numbers
* 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 SumOfGoodNumbers(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Good Numbers
```

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

}
```

Go Solution:

```
// Problem: Sum of Good Numbers
// 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 sumOfGoodNumbers(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Good Numbers
// 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 sum_of_good_numbers(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sum-of-good-numbers nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```