Problem 2420: Find All Good Indices

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

of size

n

and a positive integer

k

.

We call an index

i

in the range

k <= i < n - k

good

if the following conditions are satisfied:

The

k

elements that are just

before

the index

i

are in

non-increasing

order.

The

k

elements that are just

after

the index

i

are in

non-decreasing

order.

Return

an array of all good indices sorted in

increasing

order

.

Example 1:

Input:

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

Output:

[2,3]

Explanation:

There are two good indices in the array: - Index 2. The subarray [2,1] is in non-increasing order, and the subarray [1,3] is in non-decreasing order. - Index 3. The subarray [1,1] is in non-increasing order, and the subarray [3,4] is in non-decreasing order. Note that the index 4 is not good because [4,1] is not non-decreasing.

Example 2:

Input:

nums = [2,1,1,2], k = 2

Output:

[]

Explanation:

There are no good indices in this array.

Constraints:

n == nums.length

3 <= n <= 10

5

1 <= nums[i] <= 10

6

1 <= k <= n / 2

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Integer> goodIndices(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<int> GoodIndices(int[] nums, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* goodIndices(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun goodIndices(nums: IntArray, k: Int): List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (good-indices nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find All Good Indices
* Difficulty: Medium
* Tags: array, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Find All Good Indices
* Difficulty: Medium
* Tags: array, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public List<Integer> goodIndices(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find All Good Indices
Difficulty: Medium
Tags: array, dp, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find All Good Indices
* Difficulty: Medium
* Tags: array, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Find All Good Indices
* Difficulty: Medium
* Tags: array, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function goodIndices(nums: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Find All Good Indices
* Difficulty: Medium
* Tags: array, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public IList<int> GoodIndices(int[] nums, int k) {

}

}
```

C Solution:

```c
/*
* Problem: Find All Good Indices
* Difficulty: Medium
* Tags: array, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* goodIndices(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go Solution:

```go
// Problem: Find All Good Indices
// Difficulty: Medium
// Tags: array, dp, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func goodIndices(nums []int, k int) []int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun goodIndices(nums: IntArray, k: Int): List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find All Good Indices
// Difficulty: Medium
// Tags: array, dp, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (good-indices nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```