Problem 2111: Minimum Operations to Make the Array K-Increasing

Problem Information

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

Problem Description

You are given a

0-indexed

array

arr

consisting of

n

positive integers, and a positive integer

k

.

The array

arr

is called

K-increasing

if

arr[i-k] <= arr[i]

holds for every index

i

, where

k <= i <= n-1

.

For example,

arr = [4, 1, 5, 2, 6, 2]

is K-increasing for

k = 2

because:

arr[0] <= arr[2] (4 <= 5)

arr[1] <= arr[3] (1 <= 2)

arr[2] <= arr[4] (5 <= 6)

arr[3] <= arr[5] (2 <= 2)

However, the same

arr

is not K-increasing for

k = 1

(because

arr[0] > arr[1]

) or

k = 3

(because

arr[0] > arr[3]

).

In one

operation

, you can choose an index

i

and

change

arr[i]

into

any

positive integer.

Return

the

minimum number of operations

required to make the array K-increasing for the given

k

.

Example 1:

Input:

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

Output:

4

Explanation:

For k = 1, the resultant array has to be non-decreasing. Some of the K-increasing arrays that can be formed are [5,

6

,

7

,

8

,

9

], [

1

,

1

,

1

,

1

,1], [

2

,

2

,3,

4

,

4

]. All of them require 4 operations. It is suboptimal to change the array to, for example, [

6

,

7

,

8

,

9

,

10

] because it would take 5 operations. It can be shown that we cannot make the array K-increasing in less than 4 operations.

Example 2:

Input:

arr = [4,1,5,2,6,2], k = 2

Output:

0

Explanation:

This is the same example as the one in the problem description. Here, for every index i where 2 <= i <= 5, arr[i-2] <=

arr[i]. Since the given array is already K-increasing, we do not need to perform any operations.

Example 3:

Input:

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

Output:

2

Explanation:

Indices 3 and 5 are the only ones not satisfying arr[i-3] <= arr[i] for 3 <= i <= 5. One of the ways we can make the array K-increasing is by changing arr[3] to 4 and arr[5] to 5. The array will now be [4,1,5,

4

,6,

5

]. Note that there can be other ways to make the array K-increasing, but none of them require less than 2 operations.

Constraints:

1 <= arr.length <= 10

5

1 <= arr[i], k <= arr.length

Code Snippets

C++:

```
class Solution {
public:
int kIncreasing(vector<int>& arr, int k) {

}
};
```

Java:

```
class Solution {
public int kIncreasing(int[] arr, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function kIncreasing(arr: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int KIncreasing(int[] arr, int k) {
```

```
}
}
```

C:

```
int kIncreasing(int* arr, int arrSize, int k) {

}
```

Go:

```
func kIncreasing(arr []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun kIncreasing(arr: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @param {Integer} k
# @return {Integer}
def k_increasing(arr, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $k
* @return Integer
*/
function kIncreasing($arr, $k) {

}
}
```

Dart:

```
class Solution {
int kIncreasing(List<int> arr, int k) {

}
}
```

Scala:

```
object Solution {
def kIncreasing(arr: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec k_increasing(arr :: [integer], k :: integer) :: integer
def k_increasing(arr, k) do
```

```
end
end
```

Erlang:

```
-spec k_increasing(Arr :: [integer()], K :: integer()) -> integer().
k_increasing(Arr, K) ->
.
```

Racket:

```
(define/contract (k-increasing arr k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make the Array K-Increasing
* Difficulty: Hard
* Tags: array, search
*
* 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 kIncreasing(vector<int>& arr, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make the Array K-Increasing
```

```
* Difficulty: Hard
* Tags: array, search
*
* 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 kIncreasing(int[] arr, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make the Array K-Increasing
Difficulty: Hard
Tags: array, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make the Array K-Increasing
* Difficulty: Hard
* Tags: array, search
*
* 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[]} arr
* @param {number} k
* @return {number}
*/
var kIncreasing = function(arr, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make the Array K-Increasing
* Difficulty: Hard
* Tags: array, search
*
* 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 kIncreasing(arr: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make the Array K-Increasing
* Difficulty: Hard
* Tags: array, search
```

```
*
* 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 KIncreasing(int[] arr, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make the Array K-Increasing
* Difficulty: Hard
* Tags: array, search
*
* 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 kIncreasing(int* arr, int arrSize, int k) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Make the Array K-Increasing
// Difficulty: Hard
// Tags: array, search
//
// 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 kIncreasing(arr []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun kIncreasing(arr: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make the Array K-Increasing
// Difficulty: Hard
// Tags: array, search
//
// 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 k_increasing(arr: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer} k
# @return {Integer}
def k_increasing(arr, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $k
* @return Integer
*/
function kIncreasing($arr, $k) {

}
}
```

Dart Solution:

```
class Solution {
int kIncreasing(List<int> arr, int k) {

}
}
```

Scala Solution:

```
object Solution {
def kIncreasing(arr: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec k_increasing(arr :: [integer], k :: integer) :: integer
def k_increasing(arr, k) do

end
end
```

Erlang Solution:

```
-spec k_increasing(Arr :: [integer()], K :: integer()) -> integer().
k_increasing(Arr, K) ->
.
```

Racket Solution:

```
(define/contract (k-increasing arr k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```