Problem 798: Smallest Rotation with Highest Score

Problem Information

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

Problem Description

You are given an array

nums

. You can rotate it by a non-negative integer

k

so that the array becomes

[nums[k], nums[k + 1], ... nums[nums.length - 1], nums[0], nums[1], ..., nums[k-1]]

. Afterward, any entries that are less than or equal to their index are worth one point.

For example, if we have

nums = [2,4,1,3,0]

, and we rotate by

k = 2

, it becomes

[1,3,0,2,4]

. This is worth

3

points because

1 > 0

[no points],

3 > 1

[no points],

0 <= 2

[one point],

2 <= 3

[one point],

4 <= 4

[one point].

Return

the rotation index

k

that corresponds to the highest score we can achieve if we rotated

nums

by it

. If there are multiple answers, return the smallest such index

k

.

Example 1:

Input:

nums = [2,3,1,4,0]

Output:

3

Explanation:

Scores for each k are listed below: k = 0, nums = [2,3,1,4,0], score 2 k = 1, nums = [3,1,4,0,2], score 3 k = 2, nums = [1,4,0,2,3], score 3 k = 3, nums = [4,0,2,3,1], score 4 k = 4, nums = [0,2,3,1,4], score 3 So we should choose k = 3, which has the highest score.

Example 2:

Input:

nums = [1,3,0,2,4]

Output:

0

Explanation:

nums will always have 3 points no matter how it shifts. So we will choose the smallest k, which is 0.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] < nums.length

Code Snippets

C++:

```
class Solution {
public:
int bestRotation(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int bestRotation(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function bestRotation(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int BestRotation(int[] nums) {

}
}
```

C:

```
int bestRotation(int* nums, int numsSize) {

}
```

Go:

```
func bestRotation(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun bestRotation(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func bestRotation(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn best_rotation(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def best_rotation(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function bestRotation($nums) {

}
}
```

Dart:

```
class Solution {
int bestRotation(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def bestRotation(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec best_rotation(nums :: [integer]) :: integer
def best_rotation(nums) do

end
end
```

Erlang:

```
-spec best_rotation(Nums :: [integer()]) -> integer().
best_rotation(Nums) ->
.
```

Racket:

```
(define/contract (best-rotation nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Rotation with Highest Score
* Difficulty: Hard
* 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 bestRotation(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest Rotation with Highest Score
* Difficulty: Hard
* 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 bestRotation(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Rotation with Highest Score
Difficulty: Hard
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 bestRotation(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Smallest Rotation with Highest Score
* Difficulty: Hard
* 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
* @return {number}
*/
var bestRotation = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Rotation with Highest Score
* Difficulty: Hard
* 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 bestRotation(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Smallest Rotation with Highest Score
* Difficulty: Hard
* 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 BestRotation(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Rotation with Highest Score
* Difficulty: Hard
* 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 bestRotation(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Smallest Rotation with Highest Score
// Difficulty: Hard
// 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 bestRotation(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun bestRotation(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func bestRotation(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Smallest Rotation with Highest Score
// Difficulty: Hard
// 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 best_rotation(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def best_rotation(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function bestRotation($nums) {

}
}
```

Dart Solution:

```
class Solution {
int bestRotation(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def bestRotation(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec best_rotation(nums :: [integer]) :: integer
def best_rotation(nums) do

end
end
```

Erlang Solution:

```
-spec best_rotation(Nums :: [integer()]) -> integer().
best_rotation(Nums) ->
.
```

Racket Solution:

```
(define/contract (best-rotation nums)
(-> (listof exact-integer?) exact-integer?)
)
```