Problem 3431: Minimum Unlocked Indices to Sort Nums

Problem Information

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

Problem Description

You are given an array

nums

consisting of integers between 1 and 3, and a

binary

array

locked

of the same size.

We consider

nums

sortable

if it can be sorted using adjacent swaps, where a swap between two indices

i

and

i + 1

is allowed if

nums[i] - nums[i + 1] == 1

and

locked[i] == 0

.

In one operation, you can unlock any index

i

by setting

locked[i]

to 0.

Return the

minimum

number of operations needed to make

nums

sortable

. If it is not possible to make

nums

sortable, return -1.

Example 1:

Input:

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

Output:

0

Explanation:

We can sort

nums

using the following swaps:

swap indices 1 with 2

swap indices 4 with 5

So, there is no need to unlock any index.

Example 2:

Input:

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

Output:

2

Explanation:

If we unlock indices 2 and 5, we can sort

nums

using the following swaps:

swap indices 1 with 2

swap indices 2 with 3

swap indices 4 with 5

swap indices 5 with 6

Example 3:

Input:

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

Output:

-1

Explanation:

Even if all indices are unlocked, it can be shown that

nums

is not sortable.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 3

locked.length == nums.length

0 <= locked[i] <= 1

Code Snippets

C++:

```
class Solution {
public:
int minUnlockedIndices(vector<int>& nums, vector<int>& locked) {

}
};
```

Java:

```
class Solution {
public int minUnlockedIndices(int[] nums, int[] locked) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @param {number[]} locked
```

```
* @return {number}
*/
var minUnlockedIndices = function(nums, locked) {

};
```

TypeScript:

```
function minUnlockedIndices(nums: number[], locked: number[]): number {

};
```

C#:

```
public class Solution {
public int MinUnlockedIndices(int[] nums, int[] locked) {

}
}
```

C:

```
int minUnlockedIndices(int* nums, int numsSize, int* locked, int lockedSize)
{

}
```

Go:

```
func minUnlockedIndices(nums []int, locked []int) int {

}
```

Kotlin:

```
class Solution {
fun minUnlockedIndices(nums: IntArray, locked: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} locked
# @return {Integer}
def min_unlocked_indices(nums, locked)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $locked
* @return Integer
*/
function minUnlockedIndices($nums, $locked) {

}
}
```

Dart:

```
class Solution {
int minUnlockedIndices(List<int> nums, List<int> locked) {

}
```

```
}
```

Scala:

```
object Solution {
def minUnlockedIndices(nums: Array[Int], locked: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_unlocked_indices(nums :: [integer], locked :: [integer]) :: integer
def min_unlocked_indices(nums, locked) do

end
end
```

Erlang:

```
-spec min_unlocked_indices(Nums :: [integer()], Locked :: [integer()]) ->
integer().
min_unlocked_indices(Nums, Locked) ->
.
```

Racket:

```
(define/contract (min-unlocked-indices nums locked)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Unlocked Indices to Sort Nums
* Difficulty: Medium
* Tags: array, hash, sort
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int minUnlockedIndices(vector<int>& nums, vector<int>& locked) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Unlocked Indices to Sort Nums
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int minUnlockedIndices(int[] nums, int[] locked) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Unlocked Indices to Sort Nums
Difficulty: Medium
Tags: array, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
```

```
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Unlocked Indices to Sort Nums
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @param {number[]} locked
* @return {number}
*/
var minUnlockedIndices = function(nums, locked) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Unlocked Indices to Sort Nums
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function minUnlockedIndices(nums: number[], locked: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Unlocked Indices to Sort Nums
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MinUnlockedIndices(int[] nums, int[] locked) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Unlocked Indices to Sort Nums
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

int minUnlockedIndices(int* nums, int numsSize, int* locked, int lockedSize)
{

}
```

Go Solution:

```
// Problem: Minimum Unlocked Indices to Sort Nums
// Difficulty: Medium
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func minUnlockedIndices(nums []int, locked []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minUnlockedIndices(nums: IntArray, locked: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Unlocked Indices to Sort Nums
// Difficulty: Medium
```

```
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn min_unlocked_indices(nums: Vec<i32>, locked: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} locked
# @return {Integer}
def min_unlocked_indices(nums, locked)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $locked
* @return Integer
*/
function minUnlockedIndices($nums, $locked) {

}
}
```

Dart Solution:

```
class Solution {
int minUnlockedIndices(List<int> nums, List<int> locked) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def minUnlockedIndices(nums: Array[Int], locked: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_unlocked_indices(nums :: [integer], locked :: [integer]) :: integer
def min_unlocked_indices(nums, locked) do

end
end
```

Erlang Solution:

```
-spec min_unlocked_indices(Nums :: [integer()], Locked :: [integer()]) ->
integer().
min_unlocked_indices(Nums, Locked) ->
.
```

Racket Solution:

```
(define/contract (min-unlocked-indices nums locked)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```