Problem 3041: Maximize Consecutive Elements in an Array After Modification

Problem Information

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

Problem Description

You are given a

0-indexed

array

nums

consisting of

positive

integers.

Initially, you can increase the value of

any

element in the array by

at most

1

.

After that, you need to select

one or more

elements from the final array such that those elements are

consecutive

when sorted in increasing order. For example, the elements

[3, 4, 5]

are consecutive while

[3, 4, 6]

and

[1, 1, 2, 3]

are not.

Return

the

maximum

number of elements that you can select

.

Example 1:

Input:

nums = [2,1,5,1,1]

Output:

3

Explanation:

We can increase the elements at indices 0 and 3. The resulting array is nums = [3,1,5,2,1]. We select the elements [

3

,

1

,5,

2

,1] and we sort them to obtain [1,2,3], which are consecutive. It can be shown that we cannot select more than 3 consecutive elements.

Example 2:

Input:

nums = [1,4,7,10]

Output:

1

Explanation:

The maximum consecutive elements that we can select is 1.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-selected-elements nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Consecutive Elements in an Array After Modification
* Difficulty: Hard
* 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:
int maxSelectedElements(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Consecutive Elements in an Array After Modification
* Difficulty: Hard
* 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 int maxSelectedElements(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Consecutive Elements in an Array After Modification
Difficulty: Hard
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 maxSelectedElements(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize Consecutive Elements in an Array After Modification
* Difficulty: Hard
* 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
* @return {number}
*/
var maxSelectedElements = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Consecutive Elements in an Array After Modification
* Difficulty: Hard
* 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 maxSelectedElements(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Consecutive Elements in an Array After Modification
* Difficulty: Hard
* 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 int MaxSelectedElements(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Consecutive Elements in an Array After Modification
* Difficulty: Hard
* 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
*/

int maxSelectedElements(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximize Consecutive Elements in an Array After Modification
// Difficulty: Hard
// 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 maxSelectedElements(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize Consecutive Elements in an Array After Modification
// Difficulty: Hard
// 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 max_selected_elements(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-selected-elements nums)
(-> (listof exact-integer?) exact-integer?)
)
```