Problem 1004: Max Consecutive Ones III

Problem Information

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

Problem Description

Given a binary array

nums

and an integer

k

, return

the maximum number of consecutive

1

's in the array if you can flip at most

k

0

's.

Example 1:

Input:

nums = [1,1,1,0,0,0,1,1,1,1,0], k = 2

Output:

6

Explanation:

[1,1,1,0,0,

1

,1,1,1,1,

1

] Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.

Example 2:

Input:

nums = [0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1,1], k = 3

Output:

10

Explanation:

[0,0,

1,1,

1

,

1

,1,1,1,

1

,1,1

,0,0,0,1,1,1,1] Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.

Constraints:

1 <= nums.length <= 10

5

nums[i]

is either

0

or

1

.

0 <= k <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int longestOnes(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int LongestOnes(int[] nums, int k) {

}
}
```

C:

```
int longestOnes(int* nums, int numsSize, int k) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun longestOnes(nums: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (longest-ones nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Max Consecutive Ones III
* Difficulty: Medium
* 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 longestOnes(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Max Consecutive Ones III
```

```
* Difficulty: Medium
* 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 longestOnes(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Max Consecutive Ones III
Difficulty: Medium
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 longestOnes(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Max Consecutive Ones III
* Difficulty: Medium
* 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[]} nums
* @param {number} k
* @return {number}
*/
var longestOnes = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Max Consecutive Ones III
* Difficulty: Medium
* 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 longestOnes(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Max Consecutive Ones III
* Difficulty: Medium
* 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 LongestOnes(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Max Consecutive Ones III
* Difficulty: Medium
* 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 longestOnes(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Max Consecutive Ones III
// Difficulty: Medium
// 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 longestOnes(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun longestOnes(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Max Consecutive Ones III
// Difficulty: Medium
// 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 longest_ones(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (longest-ones nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```