Problem 995: Minimum Number of K Consecutive Bit Flips

Problem Information

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

Problem Description

You are given a binary array

nums

and an integer

k

.

A

k-bit flip

is choosing a

subarray

of length

k

from

nums

and simultaneously changing every

0

in the subarray to

1

, and every

1

in the subarray to

0

.

Return

the minimum number of

k-bit flips

required so that there is no

0

in the array

. If it is not possible, return

-1

.

A

subarray

is a

contiguous

part of an array.

Example 1:

Input:

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

Output:

2

Explanation:

Flip nums[0], then flip nums[2].

Example 2:

Input:

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

Output:

-1

Explanation:

No matter how we flip subarrays of size 2, we cannot make the array become [1,1,1].

Example 3:

Input:

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

Output:

3

Explanation:

Flip nums[0],nums[1],nums[2]: nums becomes [1,1,1,1,0,1,1,0] Flip nums[4],nums[5],nums[6]: nums becomes [1,1,1,1,1,0,0,0] Flip nums[5],nums[6],nums[7]: nums becomes [1,1,1,1,1,1,1,1]

Constraints:

1 <= nums.length <= 10

5

1 <= k <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

```
.
```

Racket:

```
(define/contract (min-k-bit-flips nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of K Consecutive Bit Flips
* Difficulty: Hard
* Tags: array, queue
*
* 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 minKBitFlips(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of K Consecutive Bit Flips
* Difficulty: Hard
* Tags: array, queue
*
* 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 minKBitFlips(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of K Consecutive Bit Flips
Difficulty: Hard
Tags: array, queue

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of K Consecutive Bit Flips
* Difficulty: Hard
* Tags: array, queue
*
```

```
* 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 minKBitFlips = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of K Consecutive Bit Flips
* Difficulty: Hard
* Tags: array, queue
*
* 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 minKBitFlips(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of K Consecutive Bit Flips
* Difficulty: Hard
* Tags: array, queue
*
* 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 MinKBitFlips(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of K Consecutive Bit Flips
* Difficulty: Hard
* Tags: array, queue
*
* 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 minKBitFlips(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Minimum Number of K Consecutive Bit Flips
// Difficulty: Hard
// Tags: array, queue
//
// 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 minKBitFlips(nums []int, k int) int {

}
```

Kotlin Solution:

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

```
}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number of K Consecutive Bit Flips
// Difficulty: Hard
// Tags: array, queue
//
// 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 min_k_bit_flips(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-k-bit-flips nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```