

Problem 3191: Minimum Operations to Make Binary Array Elements Equal to One I

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

binary array

nums

.

You can do the following operation on the array

any

number of times (possibly zero):

Choose

any

3

consecutive

elements from the array and

flip

all

of them.

Flipping

an element means changing its value from 0 to 1, and from 1 to 0.

Return the

minimum

number of operations required to make all elements in

nums

equal to 1. If it is impossible, return -1.

Example 1:

Input:

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

Output:

3

Explanation:

We can do the following operations:

Choose the elements at indices 0, 1 and 2. The resulting array is

nums = [

1

,

0

,

0

,1,0,0]

.

Choose the elements at indices 1, 2 and 3. The resulting array is

nums = [1,

1

,

1

,

0

,0,0]

.

Choose the elements at indices 3, 4 and 5. The resulting array is

nums = [1,1,1,

1

,

1

,

1

]

.

Example 2:

Input:

nums = [0,1,1,1]

Output:

-1

Explanation:

It is impossible to make all elements equal to 1.

Constraints:

$3 \leq \text{nums.length} \leq 10$

5

$0 \leq \text{nums}[i] \leq 1$

Code Snippets

C++:

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

```
}  
};
```

Java:

```
class Solution {  
    public int minOperations(int[] nums) {  
  
    }  
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function minOperations(nums: number[]): number {  
  
};
```

C#:

```

public class Solution {
    public int MinOperations(int[] nums) {

    }
}

```

C:

```

int minOperations(int* nums, int numsSize) {

}

```

Go:

```

func minOperations(nums []int) int {

}

```

Kotlin:

```

class Solution {
    fun minOperations(nums: IntArray): Int {

    }
}

```

Swift:

```

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

    }
}

```

Rust:

```

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

    }
}

```

Ruby:

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

end
```

PHP:

```
class Solution {

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

    }

}
```

Dart:

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

  }
}
```

Scala:

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

  }
}
```

Elixir:

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

  end
end
```

Erlang:

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

Racket:

```
(define/contract (min-operations nums)  
  (-> (listof exact-integer?) exact-integer?)  
  )
```

Solutions

C++ Solution:

```
/*  
 * Problem: Minimum Operations to Make Binary Array Elements Equal to One I  
 * Difficulty: Medium  
 * 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 minOperations(vector<int>& nums) {  
  
    }  
};
```

Java Solution:

```
/**  
 * Problem: Minimum Operations to Make Binary Array Elements Equal to One I  
 * Difficulty: Medium  
 * 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 minOperations(int[] nums) {

}

}

```

Python3 Solution:

```

"""
Problem: Minimum Operations to Make Binary Array Elements Equal to One I
Difficulty: Medium
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 minOperations(self, nums: List[int]) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Minimum Operations to Make Binary Array Elements Equal to One I
 * Difficulty: Medium

```

```

* 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
* @return {number}
*/
var minOperations = function(nums) {

};

```

TypeScript Solution:

```

/**
* Problem: Minimum Operations to Make Binary Array Elements Equal to One I
* Difficulty: Medium
* 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 minOperations(nums: number[]): number {

};

```

C# Solution:

```

/*
* Problem: Minimum Operations to Make Binary Array Elements Equal to One I
* Difficulty: Medium
* 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 MinOperations(int[] nums) {

    }
}

```

C Solution:

```

/*
 * Problem: Minimum Operations to Make Binary Array Elements Equal to One I
 * Difficulty: Medium
 * 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 minOperations(int* nums, int numsSize) {

}

```

Go Solution:

```

// Problem: Minimum Operations to Make Binary Array Elements Equal to One I
// Difficulty: Medium
// 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 minOperations(nums []int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun minOperations(nums: IntArray): Int {

    }
}

```

Swift Solution:

```

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

    }
}

```

Rust Solution:

```

// Problem: Minimum Operations to Make Binary Array Elements Equal to One I
// Difficulty: Medium
// 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_operations(nums: Vec<i32>) -> i32 {

    }
}

```

Ruby Solution:

```

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

end

```

PHP Solution:

```

class Solution {

```

```

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

}

}

```

Dart Solution:

```

class Solution {
  int minOperations(List<int> nums) {

  }
}

```

Scala Solution:

```

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

  }
}

```

Elixir Solution:

```

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

  end
end

```

Erlang Solution:

```

-spec min_operations(Nums :: [integer()]) -> integer().
min_operations(Nums) ->

.

```

Racket Solution:

```
(define/contract (min-operations nums)
  (-> (listof exact-integer?) exact-integer?)
)
```