

Problem 1437: Check If All 1's Are at Least Length K Places Away

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given an binary array

nums

and an integer

k

, return

true

if all

1

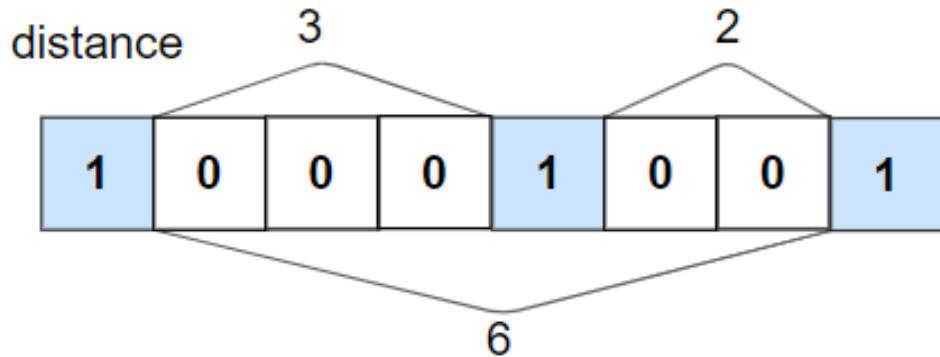
's are at least

k

places away from each other, otherwise return

false

Example 1:



Input:

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

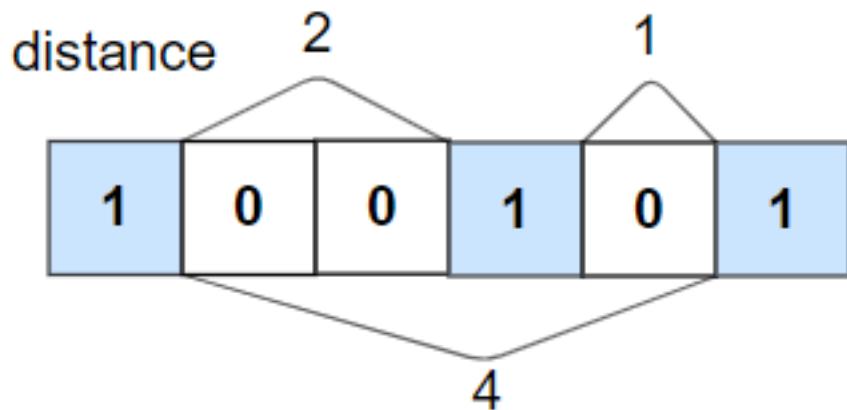
Output:

true

Explanation:

Each of the 1s are at least 2 places away from each other.

Example 2:



Input:

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

Output:

false

Explanation:

The second 1 and third 1 are only one apart from each other.

Constraints:

$1 \leq \text{nums.length} \leq 10$

5

$0 \leq k \leq \text{nums.length}$

nums[i]

is

0

or

1

Code Snippets

C++:

```
class Solution {
public:
    bool kLengthApart(vector<int>& nums, int k) {
        }
};
```

Java:

```
class Solution {  
    public boolean kLengthApart(int[] nums, int k) {  
  
    }  
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function kLengthApart(nums: number[], k: number): boolean {  
  
};
```

C#:

```
public class Solution {  
    public boolean kLengthApart(int[] nums, int k) {  
  
    }  
}
```

C:

```
bool kLengthApart(int* nums, int numsSize, int k) {  
  
}
```

Go:

```
func kLengthApart(nums []int, k int) bool {  
  
}
```

Kotlin:

```
class Solution {  
    fun kLengthApart(nums: IntArray, k: Int): Boolean {  
  
    }  
}
```

Swift:

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

Rust:

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

Ruby:

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

end
```

PHP:

```
class Solution {

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

    }
}
```

Dart:

```
class Solution {
  bool kLengthApart(List<int> nums, int k) {
    }
}
```

Scala:

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

Elixir:

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

```
end  
end
```

Erlang:

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

Racket:

```
(define/contract (k-length-apart nums k)  
  (-> (listof exact-integer?) exact-integer? boolean?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Check If All 1's Are at Least Length K Places Away  
 * Difficulty: Easy  
 * Tags: array  
 *  
 * 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:  
    bool kLengthApart(vector<int>& nums, int k) {  
        }  
    };
```

Java Solution:

```
/**  
 * Problem: Check If All 1's Are at Least Length K Places Away
```

```

* Difficulty: Easy
* Tags: array
*
* 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 boolean kLengthApart(int[] nums, int k) {
}
}

```

Python3 Solution:

```

"""
Problem: Check If All 1's Are at Least Length K Places Away
Difficulty: Easy
Tags: array

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

```

Python Solution:

```

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

```

JavaScript Solution:

```
/**  
 * Problem: Check If All 1's Are at Least Length K Places Away  
 * Difficulty: Easy  
 * Tags: array  
 *  
 * 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 {boolean}  
 */  
var kLengthApart = function(nums, k) {  
  
};
```

TypeScript Solution:

```
/**  
 * Problem: Check If All 1's Are at Least Length K Places Away  
 * Difficulty: Easy  
 * Tags: array  
 *  
 * 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 kLengthApart(nums: number[], k: number): boolean {  
  
};
```

C# Solution:

```
/*  
 * Problem: Check If All 1's Are at Least Length K Places Away  
 * Difficulty: Easy  
 * Tags: array
```

```

/*
 * 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 bool KLengthApart(int[] nums, int k) {
        }

    }
}

```

C Solution:

```

/*
 * Problem: Check If All 1's Are at Least Length K Places Away
 * Difficulty: Easy
 * Tags: array
 *
 * 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
 */

bool kLengthApart(int* nums, int numsSize, int k) {
}

```

Go Solution:

```

// Problem: Check If All 1's Are at Least Length K Places Away
// Difficulty: Easy
// Tags: array
//
// 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 kLengthApart(nums []int, k int) bool {
}

```

Kotlin Solution:

```
class Solution {  
    fun kLengthApart(nums: IntArray, k: Int): Boolean {  
        }  
        }  
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Check If All 1's Are at Least Length K Places Away  
// Difficulty: Easy  
// Tags: array  
//  
// 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 k_length_apart(nums: Vec<i32>, k: i32) -> bool {  
        }  
        }  
}
```

Ruby Solution:

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

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @param Integer $k  
     * @return Boolean  
     */  
    function kLengthApart($nums, $k) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
bool kLengthApart(List<int> nums, int k) {  
  
}  
}
```

Scala Solution:

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

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

```
(define/contract (k-length-apart nums k)  
  (-> (listof exact-integer?) exact-integer? boolean?)  
)
```