

# Problem 3576: Transform Array to All Equal Elements

## Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

You are given an integer array

nums

of size

n

containing only

1

and

-1

, and an integer

k

You can perform the following operation at most

k

times:

Choose an index

i

(

$0 \leq i < n - 1$

), and

multiply

both

`nums[i]`

and

`nums[i + 1]`

by

-1

.

Note

that you can choose the same index

i

more than once in

different

operations.

Return

true

if it is possible to make all elements of the array

equal

after at most

k

operations, and

false

otherwise.

Example 1:

Input:

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

Output:

true

Explanation:

We can make all elements in the array equal in 2 operations as follows:

Choose index

i = 1

, and multiply both

nums[1]

and

nums[2]

by -1. Now

nums = [1,1,-1,-1,1]

Choose index

i = 2

, and multiply both

nums[2]

and

nums[3]

by -1. Now

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

Example 2:

Input:

nums = [-1,-1,-1,1,1,1], k = 5

Output:

false

Explanation:

It is not possible to make all array elements equal in at most 5 operations.

Constraints:

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

5

$\text{nums}[i]$

is either -1 or 1.

$1 \leq k \leq n$

## Code Snippets

**C++:**

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

**Java:**

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

**Python3:**

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

### Python:

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

### JavaScript:

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

### TypeScript:

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

### C#:

```
public class Solution {  
    public bool CanMakeEqual(int[] nums, int k) {  
        }  
    }
```

### C:

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

**Go:**

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

**Kotlin:**

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

**Swift:**

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

**Rust:**

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

**Ruby:**

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

**PHP:**

```
class Solution {
```

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

### Dart:

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

### Scala:

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

### Elixir:

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

### Erlang:

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

### Racket:

```
(define/contract (can-make-equal nums k)
  (-> (listof exact-integer?) exact-integer? boolean?))
)
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Transform Array to All Equal Elements
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 canMakeEqual(vector<int>& nums, int k) {

    }
};
```

### Java Solution:

```
/**
 * Problem: Transform Array to All Equal Elements
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 canMakeEqual(int[] nums, int k) {

    }
}
```

```
}
```

### Python3 Solution:

```
"""
Problem: Transform Array to All Equal Elements
Difficulty: Medium
Tags: array, greedy

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

### Python Solution:

```
class Solution(object):

    def canMakeEqual(self, nums, k):

        """
        :type nums: List[int]
        :type k: int
        :rtype: bool
        """


```

### JavaScript Solution:

```
/**
 * Problem: Transform Array to All Equal Elements
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 canMakeEqual = function(nums, k) {

};

```

### TypeScript Solution:

```

/**
 * Problem: Transform Array to All Equal Elements
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 canMakeEqual(nums: number[], k: number): boolean {

};

```

### C# Solution:

```

/*
 * Problem: Transform Array to All Equal Elements
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 CanMakeEqual(int[] nums, int k) {

    }
}
```

```
}
```

### C Solution:

```
/*
 * Problem: Transform Array to All Equal Elements
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 canMakeEqual(int* nums, int numsSize, int k) {

}
```

### Go Solution:

```
// Problem: Transform Array to All Equal Elements
// Difficulty: Medium
// Tags: array, greedy
//
// 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 canMakeEqual(nums []int, k int) bool {

}
```

### Kotlin Solution:

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

    }
}
```

### Swift Solution:

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

### Rust Solution:

```
// Problem: Transform Array to All Equal Elements  
// Difficulty: Medium  
// Tags: array, greedy  
//  
// 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 can_make_equal(nums: Vec<i32>, k: i32) -> bool {  
  
}  
}
```

### Ruby Solution:

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

### PHP Solution:

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

```
}
```

```
}
```

### Dart Solution:

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

### Scala Solution:

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

### Elixir Solution:

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

### Erlang Solution:

```
-spec can_make_equal(list :: [integer()], K :: integer()) -> boolean().  
can_make_equal(list, K) ->  
.
```

### Racket Solution:

```
(define/contract (can-make-equal list k)  
  (-> (listof exact-integer?) exact-integer? boolean?)  
)
```