Problem 3086: Minimum Moves to Pick K Ones

Problem Information

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

Problem Description

You are given a binary array

nums

of length

n

, a

positive

integer

k

and a

non-negative

integer

maxChanges

.

Alice plays a game, where the goal is for Alice to pick up

k

ones from

nums

using the

minimum

number of

moves

. When the game starts, Alice picks up any index

aliceIndex

in the range

[0, n - 1]

and stands there. If

nums[aliceIndex] == 1

, Alice picks up the one and

nums[aliceIndex]

becomes

0

(this

does not

count as a move). After this, Alice can make

any

number of

moves

(

including

zero

) where in each move Alice must perform

exactly

one of the following actions:

Select any index

j != aliceIndex

such that

nums[j] == 0

and set

nums[j] = 1

. This action can be performed

at

most

maxChanges

times.

Select any two adjacent indices

x

and

y

(

|x - y| == 1

) such that

nums[x] == 1

,

nums[y] == 0

, then swap their values (set

nums[y] = 1

and

nums[x] = 0

). If

y == aliceIndex

, Alice picks up the one after this move and

nums[y]

becomes

0

.

Return

the

minimum

number of moves required by Alice to pick

exactly

k

ones

.

Example 1:

Input:

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

Output:

3

Explanation:

Alice can pick up

3

ones in

3

moves, if Alice performs the following actions in each move when standing at

aliceIndex == 1

:

At the start of the game Alice picks up the one and

nums[1]

becomes

0

.

nums

becomes

[1,

0

,0,0,0,1,1,0,0,1]

.

Select

j == 2

and perform an action of the first type.

nums

becomes

[1,

0

,1,0,0,1,1,0,0,1]

Select

x == 2

and

y == 1

, and perform an action of the second type.

nums

becomes

[1,

1

,0,0,0,1,1,0,0,1]

. As

y == aliceIndex

, Alice picks up the one and

nums

becomes

[1,

0

,0,0,0,1,1,0,0,1]

.

Select

x == 0

and

y == 1

, and perform an action of the second type.

nums

becomes

[0,

1

,0,0,0,1,1,0,0,1]

. As

y == aliceIndex

, Alice picks up the one and

nums

becomes

[0,

0

,0,0,0,1,1,0,0,1]

.

Note that it may be possible for Alice to pick up

3

ones using some other sequence of

3

moves.

Example 2:

Input:

nums = [0,0,0,0], k = 2, maxChanges = 3

Output:

4

Explanation:

Alice can pick up

2

ones in

4

moves, if Alice performs the following actions in each move when standing at aliceIndex == 0

:

Select

j == 1

and perform an action of the first type.

nums

becomes

[

0

,1,0,0]

.

Select

x == 1

and

y == 0

, and perform an action of the second type.

nums

becomes

[

1

,0,0,0]

. As

y == aliceIndex

, Alice picks up the one and

nums

becomes

[

0

,0,0,0]

.

Select

j == 1

again and perform an action of the first type.

nums

becomes

[

0

,1,0,0]

.

Select

x == 1

and

y == 0

again, and perform an action of the second type.

nums

becomes

[

1

,0,0,0]

. As

y == aliceIndex

, Alice picks up the one and

nums

becomes

[

0

,0,0,0]

.

Constraints:

2 <= n <= 10

5

0 <= nums[i] <= 1

1 <= k <= 10

5

0 <= maxChanges <= 10

5

maxChanges + sum(nums) >= k

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long minimumMoves(int[] nums, int k, int maxChanges) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long MinimumMoves(int[] nums, int k, int maxChanges) {

}
}
```

C:

```
long long minimumMoves(int* nums, int numsSize, int k, int maxChanges) {

}
```

Go:

```
func minimumMoves(nums []int, k int, maxChanges int) int64 {

}
```

Kotlin:

```
class Solution {
fun minimumMoves(nums: IntArray, k: Int, maxChanges: Int): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn minimum_moves(nums: Vec<i32>, k: i32, max_changes: i32) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

```
minimum_moves(Nums, K, MaxChanges) ->
.
```

Racket:

```
(define/contract (minimum-moves nums k maxChanges)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Moves to Pick K Ones
* Difficulty: Hard
* 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:
long long minimumMoves(vector<int>& nums, int k, int maxChanges) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Moves to Pick K Ones
* Difficulty: Hard
* 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 long minimumMoves(int[] nums, int k, int maxChanges) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Moves to Pick K Ones
Difficulty: Hard
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 minimumMoves(self, nums: List[int], k: int, maxChanges: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Moves to Pick K Ones
* Difficulty: Hard
* 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
* @param {number} maxChanges
* @return {number}
*/
var minimumMoves = function(nums, k, maxChanges) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Moves to Pick K Ones
* Difficulty: Hard
* 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 minimumMoves(nums: number[], k: number, maxChanges: number): number
{

};
```

C# Solution:

```
/*
* Problem: Minimum Moves to Pick K Ones
* Difficulty: Hard
* 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 long MinimumMoves(int[] nums, int k, int maxChanges) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Moves to Pick K Ones
* Difficulty: Hard
* 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
*/

long long minimumMoves(int* nums, int numsSize, int k, int maxChanges) {

}
```

Go Solution:

```
// Problem: Minimum Moves to Pick K Ones
// Difficulty: Hard
// 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 minimumMoves(nums []int, k int, maxChanges int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minimumMoves(nums: IntArray, k: Int, maxChanges: Int): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Moves to Pick K Ones
// Difficulty: Hard
// 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 minimum_moves(nums: Vec<i32>, k: i32, max_changes: i32) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-moves nums k maxChanges)
  (-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
  )
```