Problem 1703: Minimum Adjacent Swaps for K Consecutive Ones

Problem Information

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

Problem Description

You are given an integer array,

nums

, and an integer

k

.

nums

comprises of only

0

's and

1

's. In one move, you can choose two

adjacent

indices and swap their values.

Return

the

minimum

number of moves required so that

nums

has

k

consecutive

1

's

.

Example 1:

Input:

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

Output:

1

Explanation:

In 1 move, nums could be [1,0,0,0,

1

,

1

] and have 2 consecutive 1's.

Example 2:

Input:

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

Output:

5

Explanation:

In 5 moves, the leftmost 1 can be shifted right until nums = [0,0,0,0,0,

1

,

1

,

1

].

Example 3:

Input:

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

Output:

0

Explanation:

nums already has 2 consecutive 1's.

Constraints:

1 <= nums.length <= 10

5

nums[i]

is

0

or

1

.

1 <= k <= sum(nums)

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Adjacent Swaps for K Consecutive 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:
int minMoves(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Adjacent Swaps for K Consecutive 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 int minMoves(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Adjacent Swaps for K Consecutive 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 minMoves(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Adjacent Swaps for K Consecutive 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
* @return {number}
*/
var minMoves = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Adjacent Swaps for K Consecutive 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 minMoves(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Adjacent Swaps for K Consecutive 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 int MinMoves(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Adjacent Swaps for K Consecutive 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
*/

int minMoves(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Minimum Adjacent Swaps for K Consecutive 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 minMoves(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Adjacent Swaps for K Consecutive 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 min_moves(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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