Problem 517: Super Washing Machines

Problem Information

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

Problem Description

You have

n

super washing machines on a line. Initially, each washing machine has some dresses or is empty.

For each move, you could choose any

m

(

1 <= m <= n

) washing machines, and pass one dress of each washing machine to one of its adjacent washing machines at the same time.

Given an integer array

machines

representing the number of dresses in each washing machine from left to right on the line, return

the minimum number of moves to make all the washing machines have the same number of dresses

. If it is not possible to do it, return

-1

.

Example 1:

Input:

machines = [1,0,5]

Output:

3

Explanation:

1st move: 1 0 <-- 5 => 1 1 4 2nd move: 1 <-- 1 <-- 4 => 2 1 3 3rd move: 2 1 <-- 3 => 2 2 2

Example 2:

Input:

machines = [0,3,0]

Output:

2

Explanation:

1st move: 0 <-- 3 0 => 1 2 0 2nd move: 1 2 --> 0 => 1 1 1

Example 3:

Input:

machines = [0,2,0]

Output:

-1

Explanation:

It's impossible to make all three washing machines have the same number of dresses.

Constraints:

n == machines.length

1 <= n <= 10

4

0 <= machines[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int findMinMoves(vector<int>& machines) {

}
};
```

Java:

```
class Solution {
public int findMinMoves(int[] machines) {
```

```
}
}
```

Python3:

```
class Solution:
def findMinMoves(self, machines: List[int]) -> int:
```

Python:

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

JavaScript:

```
/**
* @param {number[]} machines
* @return {number}
*/
var findMinMoves = function(machines) {

};
```

TypeScript:

```
function findMinMoves(machines: number[]): number {

};
```

C#:

```
public class Solution {
public int FindMinMoves(int[] machines) {

}
}
```

C:

```
int findMinMoves(int* machines, int machinesSize) {

}
```

Go:

```
func findMinMoves(machines []int) int {

}
```

Kotlin:

```
class Solution {
fun findMinMoves(machines: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func findMinMoves(_ machines: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_min_moves(machines: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} machines
# @return {Integer}
def find_min_moves(machines)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $machines
* @return Integer
*/
function findMinMoves($machines) {

}
}
```

Dart:

```
class Solution {
int findMinMoves(List<int> machines) {

}
}
```

Scala:

```
object Solution {
def findMinMoves(machines: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_min_moves(machines :: [integer]) :: integer
def find_min_moves(machines) do

end
end
```

Erlang:

```
-spec find_min_moves(Machines :: [integer()]) -> integer().
find_min_moves(Machines) ->
.
```

Racket:

```
(define/contract (find-min-moves machines)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Super Washing Machines
* 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 findMinMoves(vector<int>& machines) {

}
};
```

Java Solution:

```
/**
* Problem: Super Washing Machines
* 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 findMinMoves(int[] machines) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Super Washing Machines
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 findMinMoves(self, machines: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Super Washing Machines
* 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[]} machines
* @return {number}
*/
var findMinMoves = function(machines) {

};
```

TypeScript Solution:

```
/**
* Problem: Super Washing Machines
* 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 findMinMoves(machines: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Super Washing Machines
* 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 FindMinMoves(int[] machines) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Super Washing Machines
* 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 findMinMoves(int* machines, int machinesSize) {

}
```

Go Solution:

```
// Problem: Super Washing Machines
// 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 findMinMoves(machines []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findMinMoves(machines: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func findMinMoves(_ machines: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Super Washing Machines
// 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 find_min_moves(machines: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} machines
# @return {Integer}
def find_min_moves(machines)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $machines
* @return Integer
*/
function findMinMoves($machines) {

}
}
```

Dart Solution:

```
class Solution {
int findMinMoves(List<int> machines) {

}
}
```

Scala Solution:

```
object Solution {
def findMinMoves(machines: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_min_moves(machines :: [integer]) :: integer
def find_min_moves(machines) do

end
end
```

Erlang Solution:

```
-spec find_min_moves(Machines :: [integer()]) -> integer().
find_min_moves(Machines) ->
.
```

Racket Solution:

```
(define/contract (find-min-moves machines)
(-> (listof exact-integer?) exact-integer?)
)
```