Problem 1124: Longest Well-Performing Interval

Problem Information

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

Problem Description

We are given

hours

, a list of the number of hours worked per day for a given employee.

A day is considered to be a

tiring day

if and only if the number of hours worked is (strictly) greater than

8

.

A

well-performing interval

is an interval of days for which the number of tiring days is strictly larger than the number of non-tiring days.

Return the length of the longest well-performing interval.

Example 1:

Input:

hours = [9,9,6,0,6,6,9]

Output:

3

Explanation:

The longest well-performing interval is [9,9,6].

Example 2:

Input:

hours = [6,6,6]

Output:

0

Constraints:

1 <= hours.length <= 10

4

0 <= hours[i] <= 16

Code Snippets

C++:

```
class Solution {
public:
int longestWPI(vector<int>& hours) {
```

```
}
};
```

Java:

```
class Solution {
public int longestWPI(int[] hours) {

}
}
```

Python3:

```
class Solution:
def longestWPI(self, hours: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function longestWPI(hours: number[]): number {

};
```

C#:

```
public class Solution {
public int LongestWPI(int[] hours) {

}
}
```

C:

```
int longestWPI(int* hours, int hoursSize) {

}
```

Go:

```
func longestWPI(hours []int) int {

}
```

Kotlin:

```
class Solution {
fun longestWPI(hours: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func longestWPI(_ hours: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn longest_wpi(hours: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} hours
# @return {Integer}
def longest_wpi(hours)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $hours
* @return Integer
*/
function longestWPI($hours) {

}
}
```

Dart:

```
class Solution {
int longestWPI(List<int> hours) {

}
}
```

Scala:

```
object Solution {
def longestWPI(hours: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_wpi(hours :: [integer]) :: integer
def longest_wpi(hours) do

end
end
```

Erlang:

```
-spec longest_wpi(Hours :: [integer()]) -> integer().
longest_wpi(Hours) ->
.
```

Racket:

```
(define/contract (longest-wpi hours)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Well-Performing Interval
* Difficulty: Medium
* Tags: array, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int longestWPI(vector<int>& hours) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Well-Performing Interval
* Difficulty: Medium
* Tags: array, hash, stack
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int longestWPI(int[] hours) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Well-Performing Interval
Difficulty: Medium
Tags: array, hash, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def longestWPI(self, hours: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Well-Performing Interval
* Difficulty: Medium
```

```
* Tags: array, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} hours
* @return {number}
*/
var longestWPI = function(hours) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Well-Performing Interval
* Difficulty: Medium
* Tags: array, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function longestWPI(hours: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Longest Well-Performing Interval
* Difficulty: Medium
* Tags: array, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

public class Solution {
public int LongestWPI(int[] hours) {

}
}
```

C Solution:

```
/*
* Problem: Longest Well-Performing Interval
* Difficulty: Medium
* Tags: array, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int longestWPI(int* hours, int hoursSize) {

}
```

Go Solution:

```
// Problem: Longest Well-Performing Interval
// Difficulty: Medium
// Tags: array, hash, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func longestWPI(hours []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun longestWPI(hours: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func longestWPI(_ hours: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Longest Well-Performing Interval
// Difficulty: Medium
// Tags: array, hash, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn longest_wpi(hours: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} hours
# @return {Integer}
def longest_wpi(hours)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $hours
* @return Integer
*/
function longestWPI($hours) {

}
}
```

Dart Solution:

```
class Solution {
int longestWPI(List<int> hours) {

}
}
```

Scala Solution:

```
object Solution {
def longestWPI(hours: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_wpi(hours :: [integer]) :: integer
def longest_wpi(hours) do

end
end
```

Erlang Solution:

```
-spec longest_wpi(Hours :: [integer()]) -> integer().
longest_wpi(Hours) ->
.
```

Racket Solution:

```
(define/contract (longest-wpi hours)
(-> (listof exact-integer?) exact-integer?)
)
```