Problem 3638: Maximum Balanced Shipments

Problem Information

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

Problem Description

You are given an integer array

weight

of length

n

, representing the weights of

n

parcels arranged in a straight line. A

shipment

is defined as a contiguous subarray of parcels. A shipment is considered

balanced

if the weight of the

last parcel

is

strictly less

than the

maximum weight

among all parcels in that shipment.

Select a set of

non-overlapping

, contiguous, balanced shipments such that

each parcel appears in at most one shipment

(parcels may remain unshipped).

Return the

maximum possible number

of balanced shipments that can be formed.

Example 1:

Input:

weight = [2,5,1,4,3]

Output:

2

Explanation:

We can form the maximum of two balanced shipments as follows:

Shipment 1:

[2, 5, 1]

Maximum parcel weight = 5

Last parcel weight = 1, which is strictly less than 5. Thus, it's balanced.

Shipment 2:

[4, 3]

Maximum parcel weight = 4

Last parcel weight = 3, which is strictly less than 4. Thus, it's balanced.

It is impossible to partition the parcels to achieve more than two balanced shipments, so the answer is 2.

Example 2:

Input:

weight = [4,4]

Output:

0

Explanation:

No balanced shipment can be formed in this case:

A shipment

[4, 4]

has maximum weight 4 and the last parcel's weight is also 4, which is not strictly less. Thus, it's not balanced.

Single-parcel shipments

[4]

have the last parcel weight equal to the maximum parcel weight, thus not balanced.

As there is no way to form even one balanced shipment, the answer is 0.

Constraints:

2 <= n <= 10

5

1 <= weight[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxBalancedShipments(vector<int>& weight) {

}
};
```

Java:

```
class Solution {
public int maxBalancedShipments(int[] weight) {

}
}
```

Python3:

```
class Solution:
def maxBalancedShipments(self, weight: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxBalancedShipments(weight: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxBalancedShipments(int[] weight) {

}
}
```

C:

```
int maxBalancedShipments(int* weight, int weightSize) {

}
```

Go:

```
func maxBalancedShipments(weight []int) int {
}
```

Kotlin:

```
class Solution {
fun maxBalancedShipments(weight: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxBalancedShipments(_ weight: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_balanced_shipments(weight: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} weight
# @return {Integer}
def max_balanced_shipments(weight)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $weight
* @return Integer
*/
function maxBalancedShipments($weight) {

}
}
```

Dart:

```
class Solution {
int maxBalancedShipments(List<int> weight) {

}
}
```

Scala:

```
object Solution {
def maxBalancedShipments(weight: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_balanced_shipments(weight :: [integer]) :: integer
def max_balanced_shipments(weight) do

end
end
```

Erlang:

```
-spec max_balanced_shipments(Weight :: [integer()]) -> integer().
max_balanced_shipments(Weight) ->
.
```

Racket:

```
(define/contract (max-balanced-shipments weight)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Balanced Shipments
* Difficulty: Medium
* Tags: array, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxBalancedShipments(vector<int>& weight) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Balanced Shipments
* Difficulty: Medium
* Tags: array, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxBalancedShipments(int[] weight) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Balanced Shipments
Difficulty: Medium
Tags: array, dp, greedy, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Balanced Shipments
* Difficulty: Medium
* Tags: array, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} weight
* @return {number}
*/
var maxBalancedShipments = function(weight) {
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Balanced Shipments
* Difficulty: Medium
* Tags: array, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxBalancedShipments(weight: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Balanced Shipments
* Difficulty: Medium
* Tags: array, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxBalancedShipments(int[] weight) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Balanced Shipments
* Difficulty: Medium
* Tags: array, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxBalancedShipments(int* weight, int weightSize) {

}
```

Go Solution:

```
// Problem: Maximum Balanced Shipments
// Difficulty: Medium
// Tags: array, dp, greedy, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxBalancedShipments(weight []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxBalancedShipments(weight: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxBalancedShipments(_ weight: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Balanced Shipments
// Difficulty: Medium
// Tags: array, dp, greedy, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_balanced_shipments(weight: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} weight
# @return {Integer}
def max_balanced_shipments(weight)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $weight
* @return Integer
*/
function maxBalancedShipments($weight) {

}
}
```

Dart Solution:

```
class Solution {
int maxBalancedShipments(List<int> weight) {


}
}
```

Scala Solution:

```
object Solution {
def maxBalancedShipments(weight: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_balanced_shipments(weight :: [integer]) :: integer
def max_balanced_shipments(weight) do

end
end
```

Erlang Solution:

```
-spec max_balanced_shipments(Weight :: [integer()]) -> integer().
max_balanced_shipments(Weight) ->
.
```

Racket Solution:

```
(define/contract (max-balanced-shipments weight)
(-> (listof exact-integer?) exact-integer?)
)
```