Problem 3687: Library Late Fee Calculator

Problem Information

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

Problem Description

You are given an integer array

daysLate

where

daysLate[i]

indicates how many days late the

i

th

book was returned.

The penalty is calculated as follows:

If

daysLate[i] == 1

, penalty is 1.

If

2 <= daysLate[i] <= 5

, penalty is

2 * daysLate[i]

.

If

daysLate[i] > 5

, penalty is

3 * daysLate[i]

.

Return the total penalty for all books.

Example 1:

Input:

daysLate = [5,1,7]

Output:

32

Explanation:

daysLate[0] = 5

: Penalty is

2 * daysLate[0] = 2 * 5 = 10

.

daysLate[1] = 1

: Penalty is

1

.

daysLate[2] = 7

: Penalty is

3 * daysLate[2] = 3 * 7 = 21

.

Thus, the total penalty is

10 + 1 + 21 = 32

.

Example 2:

Input:

daysLate = [1,1]

Output:

2

Explanation:

daysLate[0] = 1

: Penalty is

1

.

daysLate[1] = 1

: Penalty is

1

.

Thus, the total penalty is

1 + 1 = 2

.

Constraints:

1 <= daysLate.length <= 100

1 <= daysLate[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int lateFee(vector<int>& daysLate) {

}
};
```

Java:

```
class Solution {
public int lateFee(int[] daysLate) {
```

```
}
}
```

Python3:

```
class Solution:
def lateFee(self, daysLate: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function lateFee(daysLate: number[]): number {

};
```

C#:

```
public class Solution {
public int LateFee(int[] daysLate) {

}
}
```

C:

```
int lateFee(int* daysLate, int daysLateSize) {

}
```

Go:

```
func lateFee(daysLate []int) int {

}
```

Kotlin:

```
class Solution {
fun lateFee(daysLate: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func lateFee(_ daysLate: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn late_fee(days_late: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} days_late
# @return {Integer}
def late_fee(days_late)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $daysLate
* @return Integer
*/
function lateFee($daysLate) {

}
}
```

Dart:

```
class Solution {
int lateFee(List<int> daysLate) {

}
}
```

Scala:

```
object Solution {
def lateFee(daysLate: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec late_fee(days_late :: [integer]) :: integer
def late_fee(days_late) do

end
end
```

Erlang:

```
-spec late_fee(DaysLate :: [integer()]) -> integer().
late_fee(DaysLate) ->
.
```

Racket:

```
(define/contract (late-fee daysLate)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Library Late Fee Calculator
* Difficulty: Easy
* Tags: array
*
* 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 lateFee(vector<int>& daysLate) {

}
};
```

Java Solution:

```
/**
* Problem: Library Late Fee Calculator
* Difficulty: Easy
* Tags: array
*
* 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 lateFee(int[] daysLate) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Library Late Fee Calculator
Difficulty: Easy
Tags: array

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 lateFee(self, daysLate: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Library Late Fee Calculator
* Difficulty: Easy
* Tags: array
*
* 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[]} daysLate
* @return {number}
*/
var lateFee = function(daysLate) {

};
```

TypeScript Solution:

```
/**
* Problem: Library Late Fee Calculator
* Difficulty: Easy
* Tags: array
*
* 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 lateFee(daysLate: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Library Late Fee Calculator
* Difficulty: Easy
* Tags: array
*
* 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 LateFee(int[] daysLate) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Library Late Fee Calculator
* Difficulty: Easy
* Tags: array
*
* 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 lateFee(int* daysLate, int daysLateSize) {

}
```

Go Solution:

```
// Problem: Library Late Fee Calculator
// Difficulty: Easy
// Tags: array
//
// 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 lateFee(daysLate []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun lateFee(daysLate: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func lateFee(_ daysLate: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Library Late Fee Calculator
// Difficulty: Easy
// Tags: array
//
// 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 late_fee(days_late: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} days_late
# @return {Integer}
def late_fee(days_late)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $daysLate
* @return Integer
*/
function lateFee($daysLate) {

}
}
```

Dart Solution:

```
class Solution {
int lateFee(List<int> daysLate) {

}
}
```

Scala Solution:

```
object Solution {
def lateFee(daysLate: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec late_fee(days_late :: [integer]) :: integer
def late_fee(days_late) do

end
end
```

Erlang Solution:

```
-spec late_fee(DaysLate :: [integer()]) -> integer().
late_fee(DaysLate) ->
.
```

Racket Solution:

```
(define/contract (late-fee daysLate)
(-> (listof exact-integer?) exact-integer?)
)
```