Problem 2742: Painting the Walls

Problem Information

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

Problem Description

You are given two

0-indexed

integer arrays,

cost

and

time

, of size

n

representing the costs and the time taken to paint

n

different walls respectively. There are two painters available:

A

paid painter

that paints the

i

th

wall in

time[i]

units of time and takes

cost[i]

units of money.

A

free painter

that paints

any

wall in

1

unit of time at a cost of

0

. But the free painter can only be used if the paid painter is already

occupied

.

Return

the minimum amount of money required to paint the

n

walls.

Example 1:

Input:

cost = [1,2,3,2], time = [1,2,3,2]

Output:

3

Explanation:

The walls at index 0 and 1 will be painted by the paid painter, and it will take 3 units of time; meanwhile, the free painter will paint the walls at index 2 and 3, free of cost in 2 units of time. Thus, the total cost is 1 + 2 = 3.

Example 2:

Input:

cost = [2,3,4,2], time = [1,1,1,1]

Output:

4

Explanation:

The walls at index 0 and 3 will be painted by the paid painter, and it will take 2 units of time; meanwhile, the free painter will paint the walls at index 1 and 2, free of cost in 2 units of time. Thus, the total cost is 2 + 2 = 4.

Constraints:

1 <= cost.length <= 500

cost.length == time.length

1 <= cost[i] <= 10

6

1 <= time[i] <= 500

Code Snippets

C++:

```
class Solution {
public:
int paintWalls(vector<int>& cost, vector<int>& time) {

}
};
```

Java:

```
class Solution {
public int paintWalls(int[] cost, int[] time) {

}
}
```

Python3:

```
class Solution:
def paintWalls(self, cost: List[int], time: List[int]) -> int:
```

Python:

```
class Solution(object):
def paintWalls(self, cost, time):
"""
:type cost: List[int]
```

```
:type time: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} cost
* @param {number[]} time
* @return {number}
*/
var paintWalls = function(cost, time) {

};
```

TypeScript:

```
function paintWalls(cost: number[], time: number[]): number {

};
```

C#:

```
public class Solution {
public int PaintWalls(int[] cost, int[] time) {

}
}
```

C:

```
int paintWalls(int* cost, int costSize, int* time, int timeSize) {

}
```

Go:

```
func paintWalls(cost []int, time []int) int {

}
```

Kotlin:

```
class Solution {
fun paintWalls(cost: IntArray, time: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func paintWalls(_ cost: [Int], _ time: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn paint_walls(cost: Vec<i32>, time: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} cost
# @param {Integer[]} time
# @return {Integer}
def paint_walls(cost, time)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $cost
* @param Integer[] $time
* @return Integer
*/
function paintWalls($cost, $time) {

}

}
```

Dart:

```
class Solution {
int paintWalls(List<int> cost, List<int> time) {

}
}
```

Scala:

```
object Solution {
def paintWalls(cost: Array[Int], time: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec paint_walls(cost :: [integer], time :: [integer]) :: integer
def paint_walls(cost, time) do

end
end
```

Erlang:

```
-spec paint_walls(Cost :: [integer()], Time :: [integer()]) -> integer().
paint_walls(Cost, Time) ->
.
```

Racket:

```
(define/contract (paint-walls cost time)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Painting the Walls
* Difficulty: Hard
* Tags: array, dp
*
* 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 paintWalls(vector<int>& cost, vector<int>& time) {

}
};
```

Java Solution:

```
/**
* Problem: Painting the Walls
* Difficulty: Hard
* Tags: array, dp
*
* 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 paintWalls(int[] cost, int[] time) {

}
}
```

Python3 Solution:

```
"""
Problem: Painting the Walls
Difficulty: Hard
Tags: array, dp
```

```
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 paintWalls(self, cost: List[int], time: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def paintWalls(self, cost, time):
"""
:type cost: List[int]
:type time: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Painting the Walls
* Difficulty: Hard
* Tags: array, dp
*
* 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[]} cost
* @param {number[]} time
* @return {number}
*/
var paintWalls = function(cost, time) {

};
```

TypeScript Solution:

```
/**
* Problem: Painting the Walls
* Difficulty: Hard
* Tags: array, dp
*
* 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 paintWalls(cost: number[], time: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Painting the Walls
* Difficulty: Hard
* Tags: array, dp
*
* 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 PaintWalls(int[] cost, int[] time) {

}
}
```

C Solution:

```
/*
* Problem: Painting the Walls
* Difficulty: Hard
* Tags: array, dp
*
* 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 paintWalls(int* cost, int costSize, int* time, int timeSize) {

}
```

Go Solution:

```
// Problem: Painting the Walls
// Difficulty: Hard
// Tags: array, dp
//
// 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 paintWalls(cost []int, time []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun paintWalls(cost: IntArray, time: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func paintWalls(_ cost: [Int], _ time: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Painting the Walls
// Difficulty: Hard
// Tags: array, dp
//
// 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 paint_walls(cost: Vec<i32>, time: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} cost
# @param {Integer[]} time
# @return {Integer}
def paint_walls(cost, time)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $cost
* @param Integer[] $time
* @return Integer
*/
function paintWalls($cost, $time) {

}
}
```

Dart Solution:

```
class Solution {
int paintWalls(List<int> cost, List<int> time) {
}
}
```

Scala Solution:

```
object Solution {
def paintWalls(cost: Array[Int], time: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec paint_walls(cost :: [integer], time :: [integer]) :: integer
def paint_walls(cost, time) do

end
end
```

Erlang Solution:

```
-spec paint_walls(Cost :: [integer()], Time :: [integer()]) -> integer().
paint_walls(Cost, Time) ->
.
```

Racket Solution:

```
(define/contract (paint-walls cost time)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```