Problem 746: Min Cost Climbing Stairs

Problem Information

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

Problem Description

You are given an integer array

cost

where

cost[i]

is the cost of

i

th

step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index

0

, or the step with index

1

.

Return

the minimum cost to reach the top of the floor

.

Example 1:

Input:

cost = [10,

15

,20]

Output:

15

Explanation:

You will start at index 1. - Pay 15 and climb two steps to reach the top. The total cost is 15.

Example 2:

Input:

cost = [

1

,100,

1

,1,

1

,100,

1

,

1

,100,

1

]

Output:

6

Explanation:

You will start at index 0. - Pay 1 and climb two steps to reach index 2. - Pay 1 and climb two steps to reach index 4. - Pay 1 and climb two steps to reach index 6. - Pay 1 and climb one step to reach index 7. - Pay 1 and climb two steps to reach index 9. - Pay 1 and climb one step to reach the top. The total cost is 6.

Constraints:

2 <= cost.length <= 1000

0 <= cost[i] <= 999

Code Snippets

C++:

```
class Solution {
public:
int minCostClimbingStairs(vector<int>& cost) {



}
};
```

Java:

```
class Solution {
public int minCostClimbingStairs(int[] cost) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minCostClimbingStairs(cost: number[]): number {

};
```

C#:

```
public class Solution {
public int MinCostClimbingStairs(int[] cost) {

}
}
```

C:

```
int minCostClimbingStairs(int* cost, int costSize) {

}
```

Go:

```
func minCostClimbingStairs(cost []int) int {

}
```

Kotlin:

```
class Solution {
fun minCostClimbingStairs(cost: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minCostClimbingStairs(_ cost: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_cost_climbing_stairs(cost: Vec<i32>) -> i32 {

}
}
```

Ruby:

```ruby
# @param {Integer[]} cost
# @return {Integer}
def min_cost_climbing_stairs(cost)

end
```

PHP:

```php
class Solution {

/**
* @param Integer[] $cost
* @return Integer
*/
function minCostClimbingStairs($cost) {

}
}
```

Dart:

```dart
class Solution {
int minCostClimbingStairs(List<int> cost) {

}
}
```

Scala:

```scala
object Solution {
def minCostClimbingStairs(cost: Array[Int]): Int = {

}
}
```

Elixir:

```elixir
defmodule Solution do
@spec min_cost_climbing_stairs(cost :: [integer]) :: integer
def min_cost_climbing_stairs(cost) do
```

```
end
end
```

Erlang:

```
-spec min_cost_climbing_stairs(Cost :: [integer()]) -> integer().
min_cost_climbing_stairs(Cost) ->
.
```

Racket:

```
(define/contract (min-cost-climbing-stairs cost)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Min Cost Climbing Stairs
* Difficulty: Easy
* 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 minCostClimbingStairs(vector<int>& cost) {

}
};
```

Java Solution:

```
/**
* Problem: Min Cost Climbing Stairs
* Difficulty: Easy
* 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 minCostClimbingStairs(int[] cost) {

}
}
```

Python3 Solution:

```
"""
Problem: Min Cost Climbing Stairs
Difficulty: Easy
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 minCostClimbingStairs(self, cost: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Min Cost Climbing Stairs
* Difficulty: Easy
* 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
* @return {number}
*/
var minCostClimbingStairs = function(cost) {

};
```

TypeScript Solution:

```
/**
* Problem: Min Cost Climbing Stairs
* Difficulty: Easy
* 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 minCostClimbingStairs(cost: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Min Cost Climbing Stairs
* Difficulty: Easy
* 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 MinCostClimbingStairs(int[] cost) {

}
}
```

C Solution:

```
/*
* Problem: Min Cost Climbing Stairs
* Difficulty: Easy
* 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 minCostClimbingStairs(int* cost, int costSize) {

}
```

Go Solution:

```
// Problem: Min Cost Climbing Stairs
// Difficulty: Easy
// 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 minCostClimbingStairs(cost []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minCostClimbingStairs(cost: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minCostClimbingStairs(_ cost: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Min Cost Climbing Stairs
// Difficulty: Easy
// 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 min_cost_climbing_stairs(cost: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} cost
# @return {Integer}
def min_cost_climbing_stairs(cost)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $cost
* @return Integer
*/
function minCostClimbingStairs($cost) {

}
}
```

Dart Solution:

```
class Solution {
int minCostClimbingStairs(List<int> cost) {

}
}
```

Scala Solution:

```
object Solution {
def minCostClimbingStairs(cost: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cost_climbing_stairs(cost :: [integer]) :: integer
def min_cost_climbing_stairs(cost) do

end
end
```

Erlang Solution:

```
-spec min_cost_climbing_stairs(Cost :: [integer()]) -> integer().
min_cost_climbing_stairs(Cost) ->
.
```

Racket Solution:

```
(define/contract (min-cost-climbing-stairs cost)
(-> (listof exact-integer?) exact-integer?)
)
```