Problem 309: Best Time to Buy and Sell Stock with Cooldown

Problem Information

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

Problem Description

You are given an array

prices

where

prices[i]

is the price of a given stock on the

i

th

day.

Find the maximum profit you can achieve. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times) with the following restrictions:

After you sell your stock, you cannot buy stock on the next day (i.e., cooldown one day).

Note:

You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

Example 1:

Input:

prices = [1,2,3,0,2]

Output:

3

Explanation:

transactions = [buy, sell, cooldown, buy, sell]

Example 2:

Input:

prices = [1]

Output:

0

Constraints:

1 <= prices.length <= 5000

0 <= prices[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
int maxProfit(vector<int>& prices) {
```

```
}
};
```

Java:

```
class Solution {
public int maxProfit(int[] prices) {

}
}
```

Python3:

```
class Solution:
def maxProfit(self, prices: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxProfit(prices: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxProfit(int[] prices) {

}
}
```

C:

```
int maxProfit(int* prices, int pricesSize) {

}
```

Go:

```
func maxProfit(prices []int) int {

}
```

Kotlin:

```
class Solution {
fun maxProfit(prices: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxProfit(_ prices: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_profit(prices: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} prices
# @return {Integer}
def max_profit(prices)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $prices
* @return Integer
*/
function maxProfit($prices) {

}
}
```

Dart:

```
class Solution {
int maxProfit(List<int> prices) {

}
}
```

Scala:

```
object Solution {
def maxProfit(prices: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_profit(prices :: [integer]) :: integer
def max_profit(prices) do

end
end
```

Erlang:

```
-spec max_profit(Prices :: [integer()]) -> integer().
max_profit(Prices) ->
.
```

Racket:

```
(define/contract (max-profit prices)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Best Time to Buy and Sell Stock with Cooldown
* Difficulty: Medium
* 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 maxProfit(vector<int>& prices) {

}
};
```

Java Solution:

```
/**
* Problem: Best Time to Buy and Sell Stock with Cooldown
* Difficulty: Medium
* 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 maxProfit(int[] prices) {

}
}
```

Python3 Solution:

```
"""
Problem: Best Time to Buy and Sell Stock with Cooldown
Difficulty: Medium
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 maxProfit(self, prices: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Best Time to Buy and Sell Stock with Cooldown
* Difficulty: Medium
```

```
* 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[]} prices
* @return {number}
*/
var maxProfit = function(prices) {

};
```

TypeScript Solution:

```
/**
* Problem: Best Time to Buy and Sell Stock with Cooldown
* Difficulty: Medium
* 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 maxProfit(prices: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Best Time to Buy and Sell Stock with Cooldown
* Difficulty: Medium
* 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 MaxProfit(int[] prices) {

}
}
```

C Solution:

```
/*
* Problem: Best Time to Buy and Sell Stock with Cooldown
* Difficulty: Medium
* 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 maxProfit(int* prices, int pricesSize) {

}
```

Go Solution:

```
// Problem: Best Time to Buy and Sell Stock with Cooldown
// Difficulty: Medium
// 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 maxProfit(prices []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxProfit(prices: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxProfit(_ prices: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Best Time to Buy and Sell Stock with Cooldown
// Difficulty: Medium
// 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 max_profit(prices: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} prices
# @return {Integer}
def max_profit(prices)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $prices
* @return Integer
*/
function maxProfit($prices) {

}
}
```

Dart Solution:

```
class Solution {
int maxProfit(List<int> prices) {

}
}
```

Scala Solution:

```
object Solution {
def maxProfit(prices: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_profit(prices :: [integer]) :: integer
def max_profit(prices) do

end
end
```

Erlang Solution:

```
-spec max_profit(Prices :: [integer()]) -> integer().
max_profit(Prices) ->
.
```

Racket Solution:

```
(define/contract (max-profit prices)
(-> (listof exact-integer?) exact-integer?)
)
```