Problem 714: Best Time to Buy and Sell Stock with Transaction Fee

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, and an integer

fee

representing a transaction fee.

Find the maximum profit you can achieve. You may complete as many transactions as you like, but you need to pay the transaction fee for each transaction.

Note:

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

The transaction fee is only charged once for each stock purchase and sale.

Example 1:

Input:

prices = [1,3,2,8,4,9], fee = 2

Output:

8

Explanation:

The maximum profit can be achieved by: - Buying at prices[0] = 1 - Selling at prices[3] = 8 - Buying at prices[4] = 4 - Selling at prices[5] = 9 The total profit is ((8 - 1) - 2) + ((9 - 4) - 2) = 8.

Example 2:

Input:

prices = [1,3,7,5,10,3], fee = 3

Output:

6

Constraints:

1 <= prices.length <= 5 * 10

4

1 <= prices[i] < 5 * 10

4

0 <= fee < 5 * 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

```
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

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

```
* Tags: array, dp, greedy
*
* 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, int fee) {

}
};
```

Java Solution:

```
/**
* Problem: Best Time to Buy and Sell Stock with Transaction Fee
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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, int fee) {

}
}
```

Python3 Solution:

```
"""
Problem: Best Time to Buy and Sell Stock with Transaction Fee
Difficulty: Medium
Tags: array, dp, greedy

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], fee: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Best Time to Buy and Sell Stock with Transaction Fee
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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
* @param {number} fee
* @return {number}
*/
var maxProfit = function(prices, fee) {

};
```

TypeScript Solution:

```
/**
* Problem: Best Time to Buy and Sell Stock with Transaction Fee
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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[], fee: number): number {

};
```

C# Solution:

```
/*
* Problem: Best Time to Buy and Sell Stock with Transaction Fee
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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, int fee) {

}
}
```

C Solution:

```
/*
* Problem: Best Time to Buy and Sell Stock with Transaction Fee
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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, int fee) {

}
```

Go Solution:

```
// Problem: Best Time to Buy and Sell Stock with Transaction Fee
// Difficulty: Medium
// Tags: array, dp, greedy
//
// 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, fee int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Best Time to Buy and Sell Stock with Transaction Fee
// Difficulty: Medium
// Tags: array, dp, greedy
```

```
//
// 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>, fee: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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