# Problem 123: Best Time to Buy and Sell Stock III

## Problem Information

**Difficulty:** Hard
**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

at most two transactions

.

Note:

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

Example 1:

Input:

prices = [3,3,5,0,0,3,1,4]

Output:

6

Explanation:

Buy on day 4 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3. Then buy on day 7 (price = 1) and sell on day 8 (price = 4), profit = 4-1 = 3.

Example 2:

Input:

prices = [1,2,3,4,5]

Output:

4

Explanation:

Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4. Note that you cannot buy on day 1, buy on day 2 and sell them later, as you are engaging multiple transactions at the same time. You must sell before buying again.

Example 3:

Input:

prices = [7,6,4,3,1]

Output:

0

Explanation:

In this case, no transaction is done, i.e. max profit = 0.

Constraints:

1 <= prices.length <= 10

5

0 <= prices[i] <= 10

5

# Code Snippets

**C++:**

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

    }
};
```

**Java:**

```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:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

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

# Solutions

**C++ Solution:**

```cpp
/*
 * Problem: Best Time to Buy and Sell Stock III
 * 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 maxProfit(vector<int>& prices) {


}
};
```

**Java Solution:**

```java
/**
 * Problem: Best Time to Buy and Sell Stock III
 * 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 maxProfit(int[] prices) {


}
}
```

**Python3 Solution:**

```
"""
Problem: Best Time to Buy and Sell Stock III

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

```
    };
```

**TypeScript Solution:**

```typescript
/**
 * Problem: Best Time to Buy and Sell Stock III
 * 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 maxProfit(prices: number[]): number {


};
```

**C# Solution:**

```csharp
/*
 * Problem: Best Time to Buy and Sell Stock III
 * 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 MaxProfit(int[] prices) {


}
}
```

**C Solution:**

```c
/*
 * Problem: Best Time to Buy and Sell Stock III
 * 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 maxProfit(int* prices, int pricesSize) {


}
```

**Go Solution:**

```go
// Problem: Best Time to Buy and Sell Stock III
// 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 maxProfit(prices []int) int {


}
```

**Kotlin Solution:**

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


}
}
```

**Swift Solution:**

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


}
}
```

**Rust Solution:**

```rust
// Problem: Best Time to Buy and Sell Stock III
// 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 max_profit(prices: Vec<i32>) -> i32 {


}
}
```

**Ruby Solution:**

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


end
```

**PHP Solution:**

```php
class Solution {

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


}
}
```

**Dart Solution:**

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

```
        }
    }
```

## Scala Solution:

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


}
}
```

## Elixir Solution:

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

end
end
```

## Erlang Solution:

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

## Racket Solution:

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