Problem 1475: Final Prices With a Special Discount in a Shop

Problem Information

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

Problem Description

You are given an integer array

prices

where

prices[i]

is the price of the

i

th

item in a shop.

There is a special discount for items in the shop. If you buy the

i

th

item, then you will receive a discount equivalent to

prices[j]

where

j

is the minimum index such that

j > i

and

prices[j] <= prices[i]

. Otherwise, you will not receive any discount at all.

Return an integer array

answer

where

answer[i]

is the final price you will pay for the

i

th

item of the shop, considering the special discount.

Example 1:

Input:

prices = [8,4,6,2,3]

Output:

[4,2,4,2,3]

Explanation:

For item 0 with price[0]=8 you will receive a discount equivalent to prices[1]=4, therefore, the final price you will pay is 8 - 4 = 4. For item 1 with price[1]=4 you will receive a discount equivalent to prices[3]=2, therefore, the final price you will pay is 4 - 2 = 2. For item 2 with price[2]=6 you will receive a discount equivalent to prices[3]=2, therefore, the final price you will pay is 6 - 2 = 4. For items 3 and 4 you will not receive any discount at all.

Example 2:

Input:

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

Output:

[1,2,3,4,5]

Explanation:

In this case, for all items, you will not receive any discount at all.

Example 3:

Input:

prices = [10,1,1,6]

Output:

[9,0,1,6]

Constraints:

1 <= prices.length <= 500

1 <= prices[i] <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* finalPrices(int* prices, int pricesSize, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun finalPrices(prices: IntArray): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Final Prices With a Special Discount in a Shop
* Difficulty: Easy
* Tags: array, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Final Prices With a Special Discount in a Shop
* Difficulty: Easy
* Tags: array, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[] finalPrices(int[] prices) {

}
}
```

Python3 Solution:

```
"""
Problem: Final Prices With a Special Discount in a Shop
Difficulty: Easy
Tags: array, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def finalPrices(self, prices: List[int]) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Final Prices With a Special Discount in a Shop
* Difficulty: Easy
* Tags: array, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Final Prices With a Special Discount in a Shop
* Difficulty: Easy
* Tags: array, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

function finalPrices(prices: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Final Prices With a Special Discount in a Shop
* Difficulty: Easy
* Tags: array, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[] FinalPrices(int[] prices) {

}
}
```

C Solution:

```
/*
* Problem: Final Prices With a Special Discount in a Shop
* Difficulty: Easy
* Tags: array, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* finalPrices(int* prices, int pricesSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Final Prices With a Special Discount in a Shop
// Difficulty: Easy
// Tags: array, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func finalPrices(prices []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun finalPrices(prices: IntArray): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Final Prices With a Special Discount in a Shop
// Difficulty: Easy
// Tags: array, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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