

Problem 901: Online Stock Span

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Design an algorithm that collects daily price quotes for some stock and returns

the span

of that stock's price for the current day.

The

span

of the stock's price in one day is the maximum number of consecutive days (starting from that day and going backward) for which the stock price was less than or equal to the price of that day.

For example, if the prices of the stock in the last four days is

[7,2,1,2]

and the price of the stock today is

2

, then the span of today is

4

because starting from today, the price of the stock was less than or equal

2

for

4

consecutive days.

Also, if the prices of the stock in the last four days is

[7,34,1,2]

and the price of the stock today is

8

, then the span of today is

3

because starting from today, the price of the stock was less than or equal

8

for

3

consecutive days.

Implement the

StockSpanner

class:

StockSpanner()

Initializes the object of the class.

`int next(int price)`

Returns the

span

of the stock's price given that today's price is

price

.

Example 1:

Input

`["StockSpanner", "next", "next", "next", "next", "next", "next", "next"]` `[[], [100], [80], [60], [70], [60], [75], [85]]`

Output

`[null, 1, 1, 1, 2, 1, 4, 6]`

Explanation

`StockSpanner stockSpanner = new StockSpanner(); stockSpanner.next(100); // return 1`
`stockSpanner.next(80); // return 1 stockSpanner.next(60); // return 1 stockSpanner.next(70); //`
`return 2 stockSpanner.next(60); // return 1 stockSpanner.next(75); // return 4, because the`
`last 4 prices (including today's price of 75) were less than or equal to today's price.`
`stockSpanner.next(85); // return 6`

Constraints:

`1 <= price <= 10`

At most

10

4

calls will be made to

next

.

Code Snippets

C++:

```
class StockSpanner {
public:
    StockSpanner() {

    }

    int next(int price) {

    }
};

/**
 * Your StockSpanner object will be instantiated and called as such:
 * StockSpanner* obj = new StockSpanner();
 * int param_1 = obj->next(price);
 */
```

Java:

```
class StockSpanner {

    public StockSpanner() {

    }

}
```

```

public int next(int price) {

}

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * StockSpanner obj = new StockSpanner();
 * int param_1 = obj.next(price);
 */

```

Python3:

```

class StockSpanner:

    def __init__(self):

    def next(self, price: int) -> int:

    # Your StockSpanner object will be instantiated and called as such:
    # obj = StockSpanner()
    # param_1 = obj.next(price)

```

Python:

```

class StockSpanner(object):

    def __init__(self):

    def next(self, price):
        """
        :type price: int
        :rtype: int
        """

```

```
# Your StockSpanner object will be instantiated and called as such:
# obj = StockSpanner()
# param_1 = obj.next(price)
```

JavaScript:

```
var StockSpanner = function() {

};

/**
 * @param {number} price
 * @return {number}
 */
StockSpanner.prototype.next = function(price) {

};

/**
 * Your StockSpanner object will be instantiated and called as such:
 * var obj = new StockSpanner()
 * var param_1 = obj.next(price)
 */
```

TypeScript:

```
class StockSpanner {
    constructor() {

    }

    next(price: number): number {

    }
}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * var obj = new StockSpanner()
 * var param_1 = obj.next(price)
 */
```

C#:

```
public class StockSpanner {

    public StockSpanner() {

    }

    public int Next(int price) {

    }

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * StockSpanner obj = new StockSpanner();
 * int param_1 = obj.Next(price);
 */
```

C:

```
typedef struct {

} StockSpanner;

StockSpanner* stockSpannerCreate() {

}

int stockSpannerNext(StockSpanner* obj, int price) {

}

void stockSpannerFree(StockSpanner* obj) {

}

/**
```

```

* Your StockSpanner struct will be instantiated and called as such:
* StockSpanner* obj = stockSpannerCreate();
* int param_1 = stockSpannerNext(obj, price);

* stockSpannerFree(obj);
*/

```

Go:

```

type StockSpanner struct {

}

func Constructor() StockSpanner {

}

func (this *StockSpanner) Next(price int) int {

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * obj := Constructor();
 * param_1 := obj.Next(price);
 */

```

Kotlin:

```

class StockSpanner() {

    fun next(price: Int): Int {

    }

}

/**
 * Your StockSpanner object will be instantiated and called as such:

```

```
* var obj = StockSpanner()  
* var param_1 = obj.next(price)  
*/
```

Swift:

```
class StockSpanner {  
  
    init() {  
  
    }  
  
    func next(_ price: Int) -> Int {  
  
    }  
}  
  
/**  
 * Your StockSpanner object will be instantiated and called as such:  
 * let obj = StockSpanner()  
 * let ret_1: Int = obj.next(price)  
 */
```

Rust:

```
struct StockSpanner {  
  
}  
  
/**  
 * `&self` means the method takes an immutable reference.  
 * If you need a mutable reference, change it to `&mut self` instead.  
 */  
impl StockSpanner {  
  
    fn new() -> Self {  
  
    }  
  
    fn next(&self, price: i32) -> i32 {
```

```

}
}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * let obj = StockSpanner::new();
 * let ret_1: i32 = obj.next(price);
 */

```

Ruby:

```

class StockSpanner
  def initialize()

  end

  =begin
  :type price: Integer
  :rtype: Integer
  =end
  def next(price)

  end

  end

  # Your StockSpanner object will be instantiated and called as such:
  # obj = StockSpanner.new()
  # param_1 = obj.next(price)

```

PHP:

```

class StockSpanner {
  /**
  */
  function __construct() {

  }

```

```

/**
 * @param Integer $price
 * @return Integer
 */
function next($price) {

}

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * $obj = StockSpanner();
 * $ret_1 = $obj->next($price);
 */

```

Dart:

```

class StockSpanner {

  StockSpanner() {

  }

  int next(int price) {

  }

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * StockSpanner obj = StockSpanner();
 * int param1 = obj.next(price);
 */

```

Scala:

```

class StockSpanner() {

  def next(price: Int): Int = {

  }

}

```

```

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * val obj = new StockSpanner()
 * val param_1 = obj.next(price)
 */

```

Elixir:

```

defmodule StockSpanner do
  @spec init_() :: any
  def init_() do

  end

  @spec next(price :: integer) :: integer
  def next(price) do

  end
end

# Your functions will be called as such:
# StockSpanner.init_()
# param_1 = StockSpanner.next(price)

# StockSpanner.init_ will be called before every test case, in which you can
do some necessary initializations.

```

Erlang:

```

-spec stock_spanner_init_() -> any().
stock_spanner_init_() ->
.

-spec stock_spanner_next(Price :: integer()) -> integer().
stock_spanner_next(Price) ->
.

%% Your functions will be called as such:
%% stock_spanner_init_(),

```

```
%% Param_1 = stock_spanner_next(Price),

%% stock_spanner_init_ will be called before every test case, in which you
can do some necessary initializations.
```

Racket:

```
(define stock-spanner%
  (class object%
    (super-new)

    (init-field)

    ; next : exact-integer? -> exact-integer?
    (define/public (next price)
      )))

;; Your stock-spanner% object will be instantiated and called as such:
;; (define obj (new stock-spanner%))
;; (define param_1 (send obj next price))
```

Solutions

C++ Solution:

```
/*
 * Problem: Online Stock Span
 * Difficulty: Medium
 * Tags: stack
 *
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

class StockSpanner {
public:
    StockSpanner() {

    }
}
```

```

int next(int price) {

}

};

/**
 * Your StockSpanner object will be instantiated and called as such:
 * StockSpanner* obj = new StockSpanner();
 * int param_1 = obj->next(price);
 */

```

Java Solution:

```

/**
 * Problem: Online Stock Span
 * Difficulty: Medium
 * Tags: stack
 *
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

class StockSpanner {

public StockSpanner() {

}

public int next(int price) {

}

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * StockSpanner obj = new StockSpanner();
 * int param_1 = obj.next(price);
 */

```

Python3 Solution:

```
"""
Problem: Online Stock Span
Difficulty: Medium
Tags: stack

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class StockSpanner:

    def __init__(self):

    def next(self, price: int) -> int:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

```
class StockSpanner(object):

    def __init__(self):

    def next(self, price):
        """
        :type price: int
        :rtype: int
        """

        # Your StockSpanner object will be instantiated and called as such:
        # obj = StockSpanner()
        # param_1 = obj.next(price)
```

JavaScript Solution:

```

/**
 * Problem: Online Stock Span
 * Difficulty: Medium
 * Tags: stack
 *
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

var StockSpanner = function() {

};

/**
 * @param {number} price
 * @return {number}
 */
StockSpanner.prototype.next = function(price) {

};

/**
 * Your StockSpanner object will be instantiated and called as such:
 * var obj = new StockSpanner()
 * var param_1 = obj.next(price)
 */

```

TypeScript Solution:

```

/**
 * Problem: Online Stock Span
 * Difficulty: Medium
 * Tags: stack
 *
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

class StockSpanner {

```

```

constructor() {

}

next(price: number): number {

}

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * var obj = new StockSpanner()
 * var param_1 = obj.next(price)
 */

```

C# Solution:

```

/*
 * Problem: Online Stock Span
 * Difficulty: Medium
 * Tags: stack
 *
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

public class StockSpanner {

    public StockSpanner() {

    }

    public int Next(int price) {

    }

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * StockSpanner obj = new StockSpanner();

```

```
* int param_1 = obj.Next(price);  
*/
```

C Solution:

```
/*  
 * Problem: Online Stock Span  
 * Difficulty: Medium  
 * Tags: stack  
 *  
 * Approach: Optimized algorithm based on problem constraints  
 * Time Complexity: O(n) to O(n^2) depending on approach  
 * Space Complexity: O(1) to O(n) depending on approach  
 */  
  
typedef struct {  
  
} StockSpanner;  
  
StockSpanner* stockSpannerCreate() {  
  
}  
  
int stockSpannerNext(StockSpanner* obj, int price) {  
  
}  
  
void stockSpannerFree(StockSpanner* obj) {  
  
}  
  
/**  
 * Your StockSpanner struct will be instantiated and called as such:  
 * StockSpanner* obj = stockSpannerCreate();  
 * int param_1 = stockSpannerNext(obj, price);  
  
 * stockSpannerFree(obj);  
 */
```

```
*/
```

Go Solution:

```
// Problem: Online Stock Span
// Difficulty: Medium
// Tags: stack
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

type StockSpanner struct {

}

func Constructor() StockSpanner {

}

func (this *StockSpanner) Next(price int) int {

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * obj := Constructor();
 * param_1 := obj.Next(price);
 */
```

Kotlin Solution:

```
class StockSpanner() {

    fun next(price: Int): Int {

    }

}
```

```

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * var obj = StockSpanner()
 * var param_1 = obj.next(price)
 */

```

Swift Solution:

```

class StockSpanner {

    init() {

    }

    func next(_ price: Int) -> Int {

    }

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * let obj = StockSpanner()
 * let ret_1: Int = obj.next(price)
 */

```

Rust Solution:

```

// Problem: Online Stock Span
// Difficulty: Medium
// Tags: stack
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

struct StockSpanner {

}

```

```

/**
 * `&self` means the method takes an immutable reference.
 * If you need a mutable reference, change it to `&mut self` instead.
 */
impl StockSpanner {

    fn new() -> Self {

    }

    fn next(&self, price: i32) -> i32 {

    }
}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * let obj = StockSpanner::new();
 * let ret_1: i32 = obj.next(price);
 */

```

Ruby Solution:

```

class StockSpanner
  def initialize()

  end

  =begin
  :type price: Integer
  :rtype: Integer
  =end
  def next(price)

  end

end

```

```
# Your StockSpanner object will be instantiated and called as such:
# obj = StockSpanner.new()
# param_1 = obj.next(price)
```

PHP Solution:

```
class StockSpanner {
    /**
     *
     */
    function __construct() {

    }

    /**
     * @param Integer $price
     * @return Integer
     */
    function next($price) {

    }
}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * $obj = StockSpanner();
 * $ret_1 = $obj->next($price);
 */
```

Dart Solution:

```
class StockSpanner {

    StockSpanner() {

    }

    int next(int price) {

    }
}
```

```

/**
 * Your StockSpanner object will be instantiated and called as such:
 * StockSpanner obj = StockSpanner();
 * int param1 = obj.next(price);
 */

```

Scala Solution:

```

class StockSpanner() {

  def next(price: Int): Int = {

  }

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * val obj = new StockSpanner()
 * val param_1 = obj.next(price)
 */

```

Elixir Solution:

```

defmodule StockSpanner do
  @spec init_() :: any
  def init_() do

  end

  @spec next(price :: integer) :: integer
  def next(price) do

  end
end

# Your functions will be called as such:
# StockSpanner.init_()
# param_1 = StockSpanner.next(price)

```

```
# StockSpanner.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang Solution:

```
-spec stock_spanner_init_() -> any().
stock_spanner_init_() ->
.

-spec stock_spanner_next(Price :: integer()) -> integer().
stock_spanner_next(Price) ->
.

%% Your functions will be called as such:
%% stock_spanner_init_(),
%% Param_1 = stock_spanner_next(Price),

%% stock_spanner_init_ will be called before every test case, in which you
can do some necessary initializations.
```

Racket Solution:

```
(define stock-spanner%
  (class object%
    (super-new)

    (init-field)

    ; next : exact-integer? -> exact-integer?
    (define/public (next price)
      )))

;; Your stock-spanner% object will be instantiated and called as such:
;; (define obj (new stock-spanner%))
;; (define param_1 (send obj next price))
```