Problem 2898: Maximum Linear Stock Score

Problem Information

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

Problem Description

Given a

1-indexed

integer array

prices

, where

prices[i]

is the price of a particular stock on the

i

th

day, your task is to select some of the elements of

prices

such that your selection is

linear

.

A selection

indexes

, where

indexes

is a

1-indexed

integer array of length

k

which is a subsequence of the array

[1, 2, ..., n]

, is

linear

if:

For every

1 < j <= k

,

prices[indexes[j]] - prices[indexes[j - 1]] == indexes[j] - indexes[j - 1]

.

A

subsequence

is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

The

score

of a selection

indexes

, is equal to the sum of the following array:

[prices[indexes[1]], prices[indexes[2]], ..., prices[indexes[k]]

.

Return

the

maximum

score

that a linear selection can have

.

Example 1:

Input:

prices = [1,5,3,7,8]

Output:

20

Explanation:

We can select the indexes [2,4,5]. We show that our selection is linear: For j = 2, we have: indexes[2] - indexes[1] = 4 - 2 = 2. prices[4] - prices[2] = 7 - 5 = 2. For j = 3, we have: indexes[3] - indexes[2] = 5 - 4 = 1. prices[5] - prices[4] = 8 - 7 = 1. The sum of the elements is: prices[2] + prices[4] + prices[5] = 20. It can be shown that the maximum sum a linear selection can have is 20.

Example 2:

Input:

prices = [5,6,7,8,9]

Output:

35

Explanation:

We can select all of the indexes [1,2,3,4,5]. Since each element has a difference of exactly 1 from its previous element, our selection is linear. The sum of all the elements is 35 which is the maximum possible some out of every selection.

Constraints:

1 <= prices.length <= 10

5

1 <= prices[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long maxScore(int[] prices) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long MaxScore(int[] prices) {

}
}
```

C:

```
long long maxScore(int* prices, int pricesSize) {

}
```

Go:

```
func maxScore(prices []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxScore(prices: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn max_score(prices: Vec<i32>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Linear Stock Score
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
long long maxScore(vector<int>& prices) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Linear Stock Score
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public long maxScore(int[] prices) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Linear Stock Score
Difficulty: Medium
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Linear Stock Score
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Linear Stock Score
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

C# Solution:

```
/*
* Problem: Maximum Linear Stock Score
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public long MaxScore(int[] prices) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Linear Stock Score
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

long long maxScore(int* prices, int pricesSize) {

}
```

Go Solution:

```
// Problem: Maximum Linear Stock Score
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
func maxScore(prices []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxScore(prices: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Linear Stock Score
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn max_score(prices: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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