Problem 2291: Maximum Profit From Trading Stocks

Problem Information

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

Problem Description

You are given two

0-indexed

integer arrays of the same length

present

and

future

where

present[i]

is the current price of the

i

th

stock and

future[i]

is the price of the

i

th

stock a year in the future. You may buy each stock at most

once

. You are also given an integer

budget

representing the amount of money you currently have.

Return

the maximum amount of profit you can make.

Example 1:

Input:

present = [5,4,6,2,3], future = [8,5,4,3,5], budget = 10

Output:

6

Explanation:

One possible way to maximize your profit is to: Buy the 0

th

, 3

rd

, and 4

th

stocks for a total of 5 + 2 + 3 = 10. Next year, sell all three stocks for a total of 8 + 3 + 5 = 16. The profit you made is 16 - 10 = 6. It can be shown that the maximum profit you can make is 6.

Example 2:

Input:

present = [2,2,5], future = [3,4,10], budget = 6

Output:

5

Explanation:

The only possible way to maximize your profit is to: Buy the 2

nd

stock, and make a profit of 10 - 5 = 5. It can be shown that the maximum profit you can make is 5.

Example 3:

Input:

present = [3,3,12], future = [0,3,15], budget = 10

Output:

0

Explanation:

One possible way to maximize your profit is to: Buy the 1

st

stock, and make a profit of 3 - 3 = 0. It can be shown that the maximum profit you can make is 0.

Constraints:

n == present.length == future.length

1 <= n <= 1000

0 <= present[i], future[i] <= 100

0 <= budget <= 1000

Code Snippets

C++:

```
class Solution {
public:
int maximumProfit(vector<int>& present, vector<int>& future, int budget) {

}
};
```

Java:

```
class Solution {
public int maximumProfit(int[] present, int[] future, int budget) {

}
}
```

Python3:

```
class Solution:
def maximumProfit(self, present: List[int], future: List[int], budget: int)
-> int:
```

Python:

```
class Solution(object):
def maximumProfit(self, present, future, budget):
"""
:type present: List[int]
:type future: List[int]
:type budget: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} present
* @param {number[]} future
* @param {number} budget
* @return {number}
*/
var maximumProfit = function(present, future, budget) {

};
```

TypeScript:

```
function maximumProfit(present: number[], future: number[], budget: number):
number {

};
```

C#:

```
public class Solution {
public int MaximumProfit(int[] present, int[] future, int budget) {

}
}
```

C:

```
int maximumProfit(int* present, int presentSize, int* future, int futureSize,
int budget) {

}
```

Go:

```
func maximumProfit(present []int, future []int, budget int) int {

}
```

Kotlin:

```
class Solution {
fun maximumProfit(present: IntArray, future: IntArray, budget: Int): Int {

}
}
```

Swift:

```
class Solution {
func maximumProfit(_ present: [Int], _ future: [Int], _ budget: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_profit(present: Vec<i32>, future: Vec<i32>, budget: i32) ->
i32 {

}
}
```

Ruby:

```
# @param {Integer[]} present
# @param {Integer[]} future
# @param {Integer} budget
# @return {Integer}
def maximum_profit(present, future, budget)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $present
* @param Integer[] $future
* @param Integer $budget
* @return Integer
*/
function maximumProfit($present, $future, $budget) {

}
}
```

Dart:

```
class Solution {
int maximumProfit(List<int> present, List<int> future, int budget) {

}
}
```

Scala:

```
object Solution {
def maximumProfit(present: Array[Int], future: Array[Int], budget: Int): Int
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_profit(present :: [integer], future :: [integer], budget ::
integer) :: integer
def maximum_profit(present, future, budget) do
```

```
end
end
```

Erlang:

```
-spec maximum_profit(Present :: [integer()], Future :: [integer()], Budget ::
integer()) -> integer().
maximum_profit(Present, Future, Budget) ->
.
```

Racket:

```
(define/contract (maximum-profit present future budget)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Profit From Trading Stocks
* Difficulty: Medium
* 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 maximumProfit(vector<int>& present, vector<int>& future, int budget) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Profit From Trading Stocks
* Difficulty: Medium
* 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 maximumProfit(int[] present, int[] future, int budget) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Profit From Trading Stocks
Difficulty: Medium
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 maximumProfit(self, present: List[int], future: List[int], budget: int)
-> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumProfit(self, present, future, budget):
"""
:type present: List[int]
:type future: List[int]
:type budget: int
```

```
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Profit From Trading Stocks
* Difficulty: Medium
* 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[]} present
* @param {number[]} future
* @param {number} budget
* @return {number}
*/
var maximumProfit = function(present, future, budget) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Profit From Trading Stocks
* Difficulty: Medium
* 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 maximumProfit(present: number[], future: number[], budget: number):
number {

};
```

C# Solution:

```
/*
* Problem: Maximum Profit From Trading Stocks
* Difficulty: Medium
* 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 MaximumProfit(int[] present, int[] future, int budget) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Profit From Trading Stocks
* Difficulty: Medium
* 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 maximumProfit(int* present, int presentSize, int* future, int futureSize,
int budget) {

}
```

Go Solution:

```
// Problem: Maximum Profit From Trading Stocks
// Difficulty: Medium
// 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 maximumProfit(present []int, future []int, budget int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumProfit(present: IntArray, future: IntArray, budget: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumProfit(_ present: [Int], _ future: [Int], _ budget: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Profit From Trading Stocks
// Difficulty: Medium
// 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 maximum_profit(present: Vec<i32>, future: Vec<i32>, budget: i32) ->
i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} present
# @param {Integer[]} future
# @param {Integer} budget
# @return {Integer}
def maximum_profit(present, future, budget)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $present
* @param Integer[] $future
* @param Integer $budget
* @return Integer
*/
function maximumProfit($present, $future, $budget) {

}
}
```

Dart Solution:

```
class Solution {
int maximumProfit(List<int> present, List<int> future, int budget) {

}
}
```

Scala Solution:

```
object Solution {
def maximumProfit(present: Array[Int], future: Array[Int], budget: Int): Int
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_profit(present :: [integer], future :: [integer], budget ::
integer) :: integer
def maximum_profit(present, future, budget) do

end
end
```

Erlang Solution:

```
-spec maximum_profit(Present :: [integer()], Future :: [integer()], Budget ::
integer()) -> integer().
maximum_profit(Present, Future, Budget) ->
.
```

Racket Solution:

```
(define/contract (maximum-profit present future budget)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer?)
)
```