Problem 3155: Maximum Number of Upgradable Servers

Problem Information

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

Problem Description

You have

n

data centers and need to upgrade their servers.

You are given four arrays

count

,

upgrade

,

sell

, and

money

of length

n

, which show:

The number of servers

The cost of upgrading a single server

The money you get by selling a server

The money you initially have

for each data center respectively.

Return an array

answer

, where for each data center, the corresponding element in

answer

represents the

maximum

number of servers that can be upgraded.

Note that the money from one data center

cannot

be used for another data center.

Example 1:

Input:

count = [4,3], upgrade = [3,5], sell = [4,2], money = [8,9]

Output:

[3,2]

Explanation:

For the first data center, if we sell one server, we'll have

8 + 4 = 12

units of money and we can upgrade the remaining 3 servers.

For the second data center, if we sell one server, we'll have

9 + 2 = 11

units of money and we can upgrade the remaining 2 servers.

Example 2:

Input:

count = [1], upgrade = [2], sell = [1], money = [1]

Output:

[0]

Constraints:

1 <= count.length == upgrade.length == sell.length == money.length <= 10

5

1 <= count[i], upgrade[i], sell[i], money[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<int> maxUpgrades(vector<int>& count, vector<int>& upgrade,
vector<int>& sell, vector<int>& money) {

}
};
```

Java:

```
class Solution {
public int[] maxUpgrades(int[] count, int[] upgrade, int[] sell, int[] money)
{

}
}
```

Python3:

```
class Solution:
def maxUpgrades(self, count: List[int], upgrade: List[int], sell: List[int],
money: List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def maxUpgrades(self, count, upgrade, sell, money):
"""
:type count: List[int]
:type upgrade: List[int]
:type sell: List[int]
:type money: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} count
```

```
* @param {number[]} upgrade
* @param {number[]} sell
* @param {number[]} money
* @return {number[]}
*/
var maxUpgrades = function(count, upgrade, sell, money) {

};
```

TypeScript:

```
function maxUpgrades(count: number[], upgrade: number[], sell: number[],
money: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] MaxUpgrades(int[] count, int[] upgrade, int[] sell, int[] money)
{

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maxUpgrades(int* count, int countSize, int* upgrade, int upgradeSize,
int* sell, int sellSize, int* money, int moneySize, int* returnSize) {

}
```

Go:

```
func maxUpgrades(count []int, upgrade []int, sell []int, money []int) []int {

}
```

Kotlin:

```
class Solution {
fun maxUpgrades(count: IntArray, upgrade: IntArray, sell: IntArray, money:
IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func maxUpgrades(_ count: [Int], _ upgrade: [Int], _ sell: [Int], _ money:
[Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn max_upgrades(count: Vec<i32>, upgrade: Vec<i32>, sell: Vec<i32>,
money: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} count
# @param {Integer[]} upgrade
# @param {Integer[]} sell
# @param {Integer[]} money
# @return {Integer[]}
def max_upgrades(count, upgrade, sell, money)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $count
* @param Integer[] $upgrade
```

```
* @param Integer[] $sell
* @param Integer[] $money
* @return Integer[]
*/
function maxUpgrades($count, $upgrade, $sell, $money) {

}
}
```

Dart:

```
class Solution {
List<int> maxUpgrades(List<int> count, List<int> upgrade, List<int> sell,
List<int> money) {

}
}
```

Scala:

```
object Solution {
def maxUpgrades(count: Array[Int], upgrade: Array[Int], sell: Array[Int],
money: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_upgrades(count :: [integer], upgrade :: [integer], sell ::
[integer], money :: [integer]) :: [integer]
def max_upgrades(count, upgrade, sell, money) do

end
end
```

Erlang:

```
-spec max_upgrades(Count :: [integer()], Upgrade :: [integer()], Sell ::
[integer()], Money :: [integer()]) -> [integer()].
max_upgrades(Count, Upgrade, Sell, Money) ->
```

```
.
```

Racket:

```
(define/contract (max-upgrades count upgrade sell money)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
(listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Upgradable Servers
* Difficulty: Medium
* Tags: array, math, search
*
* 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> maxUpgrades(vector<int>& count, vector<int>& upgrade,
vector<int>& sell, vector<int>& money) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Upgradable Servers
* Difficulty: Medium
* Tags: array, math, search
*
* 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[] maxUpgrades(int[] count, int[] upgrade, int[] sell, int[] money)
{

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Upgradable Servers
Difficulty: Medium
Tags: array, math, search

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 maxUpgrades(self, count: List[int], upgrade: List[int], sell: List[int],
money: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxUpgrades(self, count, upgrade, sell, money):
"""
:type count: List[int]
:type upgrade: List[int]
:type sell: List[int]
:type money: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of Upgradable Servers
* Difficulty: Medium
* Tags: array, math, search
*
* 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[]} count
* @param {number[]} upgrade
* @param {number[]} sell
* @param {number[]} money
* @return {number[]}
*/
var maxUpgrades = function(count, upgrade, sell, money) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Upgradable Servers
* Difficulty: Medium
* Tags: array, math, search
*
* 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 maxUpgrades(count: number[], upgrade: number[], sell: number[],
money: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Upgradable Servers
```

```
* Difficulty: Medium
* Tags: array, math, search
*
* 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[] MaxUpgrades(int[] count, int[] upgrade, int[] sell, int[] money)
{

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Upgradable Servers
* Difficulty: Medium
* Tags: array, math, search
*
* 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* maxUpgrades(int* count, int countSize, int* upgrade, int upgradeSize,
int* sell, int sellSize, int* money, int moneySize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Upgradable Servers
// Difficulty: Medium
// Tags: array, math, search
//
```

```
// 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 maxUpgrades(count []int, upgrade []int, sell []int, money []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun maxUpgrades(count: IntArray, upgrade: IntArray, sell: IntArray, money:
IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func maxUpgrades(_ count: [Int], _ upgrade: [Int], _ sell: [Int], _ money:
[Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Upgradable Servers
// Difficulty: Medium
// Tags: array, math, search
//
// 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 max_upgrades(count: Vec<i32>, upgrade: Vec<i32>, sell: Vec<i32>,
money: Vec<i32>) -> Vec<i32> {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[]} count
# @param {Integer[]} upgrade
# @param {Integer[]} sell
# @param {Integer[]} money
# @return {Integer[]}
def max_upgrades(count, upgrade, sell, money)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $count
* @param Integer[] $upgrade
* @param Integer[] $sell
* @param Integer[] $money
* @return Integer[]
*/
function maxUpgrades($count, $upgrade, $sell, $money) {

}
}
```

Dart Solution:

```
class Solution {
List<int> maxUpgrades(List<int> count, List<int> upgrade, List<int> sell,
List<int> money) {

}
}
```

Scala Solution:

```
object Solution {
def maxUpgrades(count: Array[Int], upgrade: Array[Int], sell: Array[Int],
money: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_upgrades(count :: [integer], upgrade :: [integer], sell ::
[integer], money :: [integer]) :: [integer]
def max_upgrades(count, upgrade, sell, money) do

end
end
```

Erlang Solution:

```
-spec max_upgrades(Count :: [integer()], Upgrade :: [integer()], Sell ::
[integer()], Money :: [integer()]) -> [integer()].
max_upgrades(Count, Upgrade, Sell, Money) ->
.
```

Racket Solution:

```
(define/contract (max-upgrades count upgrade sell money)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
(listof exact-integer?) (listof exact-integer?))
)
```