Problem 826: Most Profit Assigning Work

Problem Information

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

Problem Description

You have

n

jobs and

m

workers. You are given three arrays:

difficulty

,

profit

, and

worker

where:

difficulty[i]

and

profit[i]

are the difficulty and the profit of the

i

th

job, and

worker[j]

is the ability of

j

th

worker (i.e., the

j

th

worker can only complete a job with difficulty at most

worker[j]

).

Every worker can be assigned

at most one job

, but one job can be

completed multiple times

.

For example, if three workers attempt the same job that pays

$1

, then the total profit will be

$3

. If a worker cannot complete any job, their profit is

$0

.

Return the maximum profit we can achieve after assigning the workers to the jobs.

Example 1:

Input:

difficulty = [2,4,6,8,10], profit = [10,20,30,40,50], worker = [4,5,6,7]

Output:

100

Explanation:

Workers are assigned jobs of difficulty [4,4,6,6] and they get a profit of [20,20,30,30] separately.

Example 2:

Input:

difficulty = [85,47,57], profit = [24,66,99], worker = [40,25,25]

Output:

0

Constraints:

n == difficulty.length

n == profit.length

m == worker.length

1 <= n, m <= 10

4

1 <= difficulty[i], profit[i], worker[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int maxProfitAssignment(vector<int>& difficulty, vector<int>& profit,
vector<int>& worker) {

}
};
```

Java:

```
class Solution {
public int maxProfitAssignment(int[] difficulty, int[] profit, int[] worker)
{

}
}
```

Python3:

```
class Solution:
def maxProfitAssignment(self, difficulty: List[int], profit: List[int],
worker: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxProfitAssignment(self, difficulty, profit, worker):
"""
:type difficulty: List[int]
:type profit: List[int]
:type worker: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} difficulty
* @param {number[]} profit
* @param {number[]} worker
* @return {number}
*/
var maxProfitAssignment = function(difficulty, profit, worker) {

};
```

TypeScript:

```
function maxProfitAssignment(difficulty: number[], profit: number[], worker:
number[]): number {

};
```

C#:

```
public class Solution {
public int MaxProfitAssignment(int[] difficulty, int[] profit, int[] worker)
{

}
}
```

C:

```
int maxProfitAssignment(int* difficulty, int difficultySize, int* profit, int
profitSize, int* worker, int workerSize) {

}
```

Go:

```
func maxProfitAssignment(difficulty []int, profit []int, worker []int) int {

}
```

Kotlin:

```
class Solution {
fun maxProfitAssignment(difficulty: IntArray, profit: IntArray, worker:
IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxProfitAssignment(_ difficulty: [Int], _ profit: [Int], _ worker:
[Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_profit_assignment(difficulty: Vec<i32>, profit: Vec<i32>, worker:
Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} difficulty
# @param {Integer[]} profit
# @param {Integer[]} worker
# @return {Integer}
def max_profit_assignment(difficulty, profit, worker)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $difficulty
* @param Integer[] $profit
* @param Integer[] $worker
* @return Integer
*/
function maxProfitAssignment($difficulty, $profit, $worker) {

}
}
```

Dart:

```
class Solution {
int maxProfitAssignment(List<int> difficulty, List<int> profit, List<int>
worker) {

}
}
```

Scala:

```
object Solution {
def maxProfitAssignment(difficulty: Array[Int], profit: Array[Int], worker:
Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_profit_assignment(difficulty :: [integer], profit :: [integer],
worker :: [integer]) :: integer
def max_profit_assignment(difficulty, profit, worker) do

end
end
```

Erlang:

```
-spec max_profit_assignment(Difficulty :: [integer()], Profit :: [integer()],
Worker :: [integer()]) -> integer().
max_profit_assignment(Difficulty, Profit, Worker) ->
.
```

Racket:

```
(define/contract (max-profit-assignment difficulty profit worker)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Most Profit Assigning Work
* Difficulty: Medium
* Tags: array, greedy, sort, 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 maxProfitAssignment(vector<int>& difficulty, vector<int>& profit,
vector<int>& worker) {
}
};
```

Java Solution:

```
/**
* Problem: Most Profit Assigning Work
* Difficulty: Medium
* Tags: array, greedy, sort, 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 maxProfitAssignment(int[] difficulty, int[] profit, int[] worker)
{

}
}
```

Python3 Solution:

```
"""
Problem: Most Profit Assigning Work
Difficulty: Medium
Tags: array, greedy, sort, 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 maxProfitAssignment(self, difficulty: List[int], profit: List[int],
worker: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxProfitAssignment(self, difficulty, profit, worker):
"""
:type difficulty: List[int]
:type profit: List[int]
:type worker: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Most Profit Assigning Work
* Difficulty: Medium
* Tags: array, greedy, sort, 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[]} difficulty
* @param {number[]} profit
* @param {number[]} worker
* @return {number}
*/
var maxProfitAssignment = function(difficulty, profit, worker) {

};
```

TypeScript Solution:

```
/**
* Problem: Most Profit Assigning Work
* Difficulty: Medium
* Tags: array, greedy, sort, 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 maxProfitAssignment(difficulty: number[], profit: number[], worker:
number[]): number {

};
```

C# Solution:

```
/*
* Problem: Most Profit Assigning Work
* Difficulty: Medium
* Tags: array, greedy, sort, 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 MaxProfitAssignment(int[] difficulty, int[] profit, int[] worker)
{

}
}
```

C Solution:

```
/*
* Problem: Most Profit Assigning Work
* Difficulty: Medium
* Tags: array, greedy, sort, 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
*/

int maxProfitAssignment(int* difficulty, int difficultySize, int* profit, int
profitSize, int* worker, int workerSize) {

}
```

Go Solution:

```
// Problem: Most Profit Assigning Work
// Difficulty: Medium
// Tags: array, greedy, sort, 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 maxProfitAssignment(difficulty []int, profit []int, worker []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxProfitAssignment(difficulty: IntArray, profit: IntArray, worker:
IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxProfitAssignment(_ difficulty: [Int], _ profit: [Int], _ worker:
[Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Most Profit Assigning Work
// Difficulty: Medium
// Tags: array, greedy, sort, 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_profit_assignment(difficulty: Vec<i32>, profit: Vec<i32>, worker: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} difficulty
# @param {Integer[]} profit
# @param {Integer[]} worker
# @return {Integer}
def max_profit_assignment(difficulty, profit, worker)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $difficulty
* @param Integer[] $profit
* @param Integer[] $worker
* @return Integer
*/
function maxProfitAssignment($difficulty, $profit, $worker) {

}
}
```

Dart Solution:

```
class Solution {
int maxProfitAssignment(List<int> difficulty, List<int> profit, List<int> worker) {
}
}
```

Scala Solution:

```
object Solution {
def maxProfitAssignment(difficulty: Array[Int], profit: Array[Int], worker:
Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_profit_assignment(difficulty :: [integer], profit :: [integer],
worker :: [integer]) :: integer
def max_profit_assignment(difficulty, profit, worker) do

end
end
```

Erlang Solution:

```
-spec max_profit_assignment(Difficulty :: [integer()], Profit :: [integer()],
Worker :: [integer()]) -> integer().
max_profit_assignment(Difficulty, Profit, Worker) ->
.
```

Racket Solution:

```
(define/contract (max-profit-assignment difficulty profit worker)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```