Problem 2838: Maximum Coins Heroes Can Collect

Problem Information

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

Problem Description

There is a battle and

n

heroes are trying to defeat

m

monsters. You are given two

1-indexed

arrays of

positive

integers

heroes

and

monsters

of length

n

and

m

, respectively.

heroes

[i]

is the power of

i

th

hero, and

monsters

[i]

is the power of

i

th

monster.

The

i

th

hero can defeat the

j

th

monster if

monsters[j] <= heroes[i]

.

You are also given a

1-indexed

array

coins

of length

m

consisting of

positive

integers.

coins[i]

is the number of coins that each hero earns after defeating the

i

th

monster.

Return

an array

ans

of length

n

where

ans[i]

is the

maximum

number of coins that the

i

th

hero can collect from this battle

.

Notes

The health of a hero doesn't get reduced after defeating a monster.

Multiple heroes can defeat a monster, but each monster can be defeated by a given hero only once.

Example 1:

Input:

heroes = [1,4,2], monsters = [1,1,5,2,3], coins = [2,3,4,5,6]

Output:

[5,16,10]

Explanation:

For each hero, we list the index of all the monsters he can defeat: 1

st

hero: [1,2] since the power of this hero is 1 and monsters[1], monsters[2] <= 1. So this hero collects coins[1] + coins[2] = 5 coins. 2

nd

hero: [1,2,4,5] since the power of this hero is 4 and monsters[1], monsters[2], monsters[4], monsters[5] <= 4. So this hero collects coins[1] + coins[2] + coins[4] + coins[5] = 16 coins. 3

rd

hero: [1,2,4] since the power of this hero is 2 and monsters[1], monsters[2], monsters[4] <= 2. So this hero collects coins[1] + coins[2] + coins[4] = 10 coins. So the answer would be [5,16,10].

Example 2:

Input:

heroes = [5], monsters = [2,3,1,2], coins = [10,6,5,2]

Output:

[23]

Explanation:

This hero can defeat all the monsters since monsters[i] <= 5. So he collects all of the coins: coins[1] + coins[2] + coins[3] + coins[4] = 23, and the answer would be [23].

Example 3:

Input:

heroes = [4,4], monsters = [5,7,8], coins = [1,1,1]

Output:

[0,0]

Explanation:

In this example, no hero can defeat a monster. So the answer would be [0,0],

Constraints:

1 <= n == heroes.length <= 10

5

1 <= m == monsters.length <= 10

5

coins.length == m

1 <= heroes[i], monsters[i], coins[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<long long> maximumCoins(vector<int>& heroes, vector<int>& monsters,
vector<int>& coins) {

}
};
```

Java:

```
class Solution {
public long[] maximumCoins(int[] heroes, int[] monsters, int[] coins) {

}
}
```

Python3:

```
class Solution:
def maximumCoins(self, heroes: List[int], monsters: List[int], coins:
List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def maximumCoins(self, heroes, monsters, coins):
"""
:type heroes: List[int]
:type monsters: List[int]
:type coins: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} heroes
* @param {number[]} monsters
* @param {number[]} coins
* @return {number[]}
*/
var maximumCoins = function(heroes, monsters, coins) {

};
```

TypeScript:

```
function maximumCoins(heroes: number[], monsters: number[], coins: number[]):
number[] {

};
```

C#:

```
public class Solution {
public long[] MaximumCoins(int[] heroes, int[] monsters, int[] coins) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* maximumCoins(int* heroes, int heroesSize, int* monsters, int
monstersSize, int* coins, int coinsSize, int* returnSize) {

}
```

Go:

```
func maximumCoins(heroes []int, monsters []int, coins []int) []int64 {

}
```

Kotlin:

```
class Solution {
fun maximumCoins(heroes: IntArray, monsters: IntArray, coins: IntArray):
LongArray {

}
}
```

Swift:

```
class Solution {
func maximumCoins(_ heroes: [Int], _ monsters: [Int], _ coins: [Int]) ->
[Int] {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_coins(heroes: Vec<i32>, monsters: Vec<i32>, coins: Vec<i32>)
-> Vec<i64> {

}
}
```

Ruby:

```
# @param {Integer[]} heroes
# @param {Integer[]} monsters
# @param {Integer[]} coins
# @return {Integer[]}
def maximum_coins(heroes, monsters, coins)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $heroes
* @param Integer[] $monsters
* @param Integer[] $coins
* @return Integer[]
*/
function maximumCoins($heroes, $monsters, $coins) {

}
}
```

Dart:

```
class Solution {
List<int> maximumCoins(List<int> heroes, List<int> monsters, List<int> coins)
{

}
}
```

Scala:

```
object Solution {
def maximumCoins(heroes: Array[Int], monsters: Array[Int], coins:
Array[Int]): Array[Long] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_coins(heroes :: [integer], monsters :: [integer], coins ::
[integer]) :: [integer]
def maximum_coins(heroes, monsters, coins) do

end
end
```

Erlang:

```
-spec maximum_coins(Heroes :: [integer()], Monsters :: [integer()], Coins ::
[integer()]) -> [integer()].
maximum_coins(Heroes, Monsters, Coins) ->
.
```

Racket:

```
(define/contract (maximum-coins heroes monsters coins)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
(listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Coins Heroes Can Collect
* Difficulty: Medium
* Tags: array, 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:
vector<long long> maximumCoins(vector<int>& heroes, vector<int>& monsters,
vector<int>& coins) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Coins Heroes Can Collect
* Difficulty: Medium
* Tags: array, 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 long[] maximumCoins(int[] heroes, int[] monsters, int[] coins) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Coins Heroes Can Collect
Difficulty: Medium
Tags: array, 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 maximumCoins(self, heroes: List[int], monsters: List[int], coins:
List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumCoins(self, heroes, monsters, coins):
"""
:type heroes: List[int]
:type monsters: List[int]
:type coins: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Coins Heroes Can Collect
* Difficulty: Medium
* Tags: array, 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[]} heroes
* @param {number[]} monsters
* @param {number[]} coins
* @return {number[]}
*/
var maximumCoins = function(heroes, monsters, coins) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Coins Heroes Can Collect
* Difficulty: Medium
* Tags: array, 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 maximumCoins(heroes: number[], monsters: number[], coins: number[]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Maximum Coins Heroes Can Collect
* Difficulty: Medium
* Tags: array, 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 long[] MaximumCoins(int[] heroes, int[] monsters, int[] coins) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Coins Heroes Can Collect
* Difficulty: Medium
* Tags: array, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* maximumCoins(int* heroes, int heroesSize, int* monsters, int
monstersSize, int* coins, int coinsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Maximum Coins Heroes Can Collect
// Difficulty: Medium
// Tags: array, 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 maximumCoins(heroes []int, monsters []int, coins []int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maximumCoins(heroes: IntArray, monsters: IntArray, coins: IntArray):
LongArray {

}
}
```

Swift Solution:

```
class Solution {
func maximumCoins(_ heroes: [Int], _ monsters: [Int], _ coins: [Int]) ->
[Int] {

}
}
```

Rust Solution:

```
// Problem: Maximum Coins Heroes Can Collect
// Difficulty: Medium
// Tags: array, 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 maximum_coins(heroes: Vec<i32>, monsters: Vec<i32>, coins: Vec<i32>)
-> Vec<i64> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} heroes
# @param {Integer[]} monsters
# @param {Integer[]} coins
# @return {Integer[]}
def maximum_coins(heroes, monsters, coins)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $heroes
* @param Integer[] $monsters
* @param Integer[] $coins
* @return Integer[]
*/
function maximumCoins($heroes, $monsters, $coins) {

}
}
```

Dart Solution:

```
class Solution {
List<int> maximumCoins(List<int> heroes, List<int> monsters, List<int> coins)
{

}
}
```

Scala Solution:

```
object Solution {
def maximumCoins(heroes: Array[Int], monsters: Array[Int], coins:
Array[Int]): Array[Long] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_coins(heroes :: [integer], monsters :: [integer], coins ::
[integer]) :: [integer]
def maximum_coins(heroes, monsters, coins) do

end
end
```

Erlang Solution:

```
-spec maximum_coins(Heroes :: [integer()], Monsters :: [integer()], Coins ::
[integer()]) -> [integer()].
```

```
maximum_coins(Heroes, Monsters, Coins) ->
.
```

Racket Solution:

```
(define/contract (maximum-coins heroes monsters coins)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
(listof exact-integer?))
)
```