Problem 2410: Maximum Matching of Players With Trainers

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

players

, where

players[i]

represents the

ability

of the

i

th

player. You are also given a

0-indexed

integer array

trainers

, where

trainers[j]

represents the

training capacity

of the

j

th

trainer.

The

i

th

player can

match

with the

j

th

trainer if the player's ability is

less than or equal to

the trainer's training capacity. Additionally, the

i

th

player can be matched with at most one trainer, and the

j

th

trainer can be matched with at most one player.

Return

the

maximum

number of matchings between

players

and

trainers

that satisfy these conditions.

Example 1:

Input:

players = [4,7,9], trainers = [8,2,5,8]

Output:

2

Explanation:

One of the ways we can form two matchings is as follows: - players[0] can be matched with trainers[0] since 4 <= 8. - players[1] can be matched with trainers[3] since 7 <= 8. It can be proven that 2 is the maximum number of matchings that can be formed.

Example 2:

Input:

players = [1,1,1], trainers = [10]

Output:

1

Explanation:

The trainer can be matched with any of the 3 players. Each player can only be matched with one trainer, so the maximum answer is 1.

Constraints:

1 <= players.length, trainers.length <= 10

5

1 <= players[i], trainers[j] <= 10

9

Note:

This question is the same as

445: Assign Cookies.

Code Snippets

C++:

```
class Solution {
public:
int matchPlayersAndTrainers(vector<int>& players, vector<int>& trainers) {

}
};
```

Java:

```
class Solution {
public int matchPlayersAndTrainers(int[] players, int[] trainers) {

}
}
```

Python3:

```
class Solution:
def matchPlayersAndTrainers(self, players: List[int], trainers: List[int]) ->
int:
```

Python:

```
class Solution(object):
def matchPlayersAndTrainers(self, players, trainers):
"""
:type players: List[int]
:type trainers: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} players
* @param {number[]} trainers
* @return {number}
*/
```

```
var matchPlayersAndTrainers = function(players, trainers) {

};
```

TypeScript:

```
function matchPlayersAndTrainers(players: number[], trainers: number[]):
number {

};
```

C#:

```
public class Solution {
public int MatchPlayersAndTrainers(int[] players, int[] trainers) {

}
}
```

C:

```
int matchPlayersAndTrainers(int* players, int playersSize, int* trainers, int
trainersSize) {

}
```

Go:

```
func matchPlayersAndTrainers(players []int, trainers []int) int {

}
```

Kotlin:

```
class Solution {
fun matchPlayersAndTrainers(players: IntArray, trainers: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func matchPlayersAndTrainers(_ players: [Int], _ trainers: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn match_players_and_trainers(players: Vec<i32>, trainers: Vec<i32>) ->
i32 {

}
}
```

Ruby:

```
# @param {Integer[]} players
# @param {Integer[]} trainers
# @return {Integer}
def match_players_and_trainers(players, trainers)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $players
* @param Integer[] $trainers
* @return Integer
*/
function matchPlayersAndTrainers($players, $trainers) {

}
}
```

Dart:

```
class Solution {
int matchPlayersAndTrainers(List<int> players, List<int> trainers) {
```

```
}
}
```

Scala:

```
object Solution {
def matchPlayersAndTrainers(players: Array[Int], trainers: Array[Int]): Int =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec match_players_and_trainers(players :: [integer], trainers :: [integer])
:: integer
def match_players_and_trainers(players, trainers) do

end
end
```

Erlang:

```
-spec match_players_and_trainers(Players :: [integer()], Trainers ::
[integer()]) -> integer().
match_players_and_trainers(Players, Trainers) ->
.
```

Racket:

```
(define/contract (match-players-and-trainers players trainers)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Matching of Players With Trainers
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 matchPlayersAndTrainers(vector<int>& players, vector<int>& trainers) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Matching of Players With Trainers
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 matchPlayersAndTrainers(int[] players, int[] trainers) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Matching of Players With Trainers
Difficulty: Medium
Tags: array, greedy, sort
```

```
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 matchPlayersAndTrainers(self, players: List[int], trainers: List[int]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def matchPlayersAndTrainers(self, players, trainers):
"""
:type players: List[int]
:type trainers: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Matching of Players With Trainers
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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[]} players
* @param {number[]} trainers
* @return {number}
*/
var matchPlayersAndTrainers = function(players, trainers) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Matching of Players With Trainers
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 matchPlayersAndTrainers(players: number[], trainers: number[]):
number {

};
```

C# Solution:

```
/*
* Problem: Maximum Matching of Players With Trainers
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 MatchPlayersAndTrainers(int[] players, int[] trainers) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Matching of Players With Trainers
* Difficulty: Medium
* Tags: array, greedy, sort
*
```

```
* 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 matchPlayersAndTrainers(int* players, int playersSize, int* trainers, int
trainersSize) {

}
```

Go Solution:

```
// Problem: Maximum Matching of Players With Trainers
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 matchPlayersAndTrainers(players []int, trainers []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun matchPlayersAndTrainers(players: IntArray, trainers: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func matchPlayersAndTrainers(_ players: [Int], _ trainers: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Matching of Players With Trainers
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 match_players_and_trainers(players: Vec<i32>, trainers: Vec<i32>) ->
i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} players
# @param {Integer[]} trainers
# @return {Integer}
def match_players_and_trainers(players, trainers)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $players
* @param Integer[] $trainers
* @return Integer
*/
function matchPlayersAndTrainers($players, $trainers) {

}
}
```

Dart Solution:

```
class Solution {
int matchPlayersAndTrainers(List<int> players, List<int> trainers) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def matchPlayersAndTrainers(players: Array[Int], trainers: Array[Int]): Int =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec match_players_and_trainers(players :: [integer], trainers :: [integer])
:: integer
def match_players_and_trainers(players, trainers) do

end
end
```

Erlang Solution:

```
-spec match_players_and_trainers(Players :: [integer()], Trainers ::
[integer()]) -> integer().
match_players_and_trainers(Players, Trainers) ->
.
```

Racket Solution:

```
(define/contract (match-players-and-trainers players trainers)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```