Problem 3238: Find the Number of Winning Players

Problem Information

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

Problem Description

You are given an integer

n

representing the number of players in a game and a 2D array

pick

where

pick[i] = [x

i

, y

i

]

represents that the player

x

i

...

picked a ball of color

y

i

.

Player

i

wins

the game if they pick

strictly more

than

i

balls of the

same

color. In other words,

Player 0 wins if they pick any ball.

Player 1 wins if they pick at least two balls of the

same

color.

...

Player

i

wins if they pick at least

i + 1

balls of the

same

color.

Return the number of players who

win

the game.

Note

that

multiple

players can win the game.

Example 1:

Input:

n = 4, pick = [[0,0],[1,0],[1,0],[2,1],[2,1],[2,0]]

Output:

2

Explanation:

Player 0 and player 1 win the game, while players 2 and 3 do not win.

Example 2:

Input:

n = 5, pick = [[1,1],[1,2],[1,3],[1,4]]

Output:

0

Explanation:

No player wins the game.

Example 3:

Input:

n = 5, pick = [[1,1],[2,4],[2,4],[2,4]]

Output:

1

Explanation:

Player 2 wins the game by picking 3 balls with color 4.

Constraints:

2 <= n <= 10

1 <= pick.length <= 100

pick[i].length == 2

0 <= x

i

<= n - 1

0 <= y

i

<= 10

Code Snippets

C++:

```
class Solution {
public:
int winningPlayerCount(int n, vector<vector<int>>& pick) {

}
};
```

Java:

```
class Solution {
public int winningPlayerCount(int n, int[][] pick) {

}
}
```

Python3:

```
class Solution:
def winningPlayerCount(self, n: int, pick: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def winningPlayerCount(self, n, pick):
```

```
"""
:type n: int
:type pick: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} pick
* @return {number}
*/
var winningPlayerCount = function(n, pick) {

};
```

TypeScript:

```
function winningPlayerCount(n: number, pick: number[][]): number {

};
```

C#:

```
public class Solution {
public int WinningPlayerCount(int n, int[][] pick) {

}
}
```

C:

```
int winningPlayerCount(int n, int** pick, int pickSize, int* pickColSize) {

}
```

Go:

```
func winningPlayerCount(n int, pick [][]int) int {

}
```

Kotlin:

```
class Solution {
fun winningPlayerCount(n: Int, pick: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func winningPlayerCount(_ n: Int, _ pick: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn winning_player_count(n: i32, pick: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} pick
# @return {Integer}
def winning_player_count(n, pick)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $pick
* @return Integer
*/
function winningPlayerCount($n, $pick) {
}
}
```

Dart:

```
class Solution {
int winningPlayerCount(int n, List<List<int>> pick) {

}
}
```

Scala:

```
object Solution {
def winningPlayerCount(n: Int, pick: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec winning_player_count(n :: integer, pick :: [[integer]]) :: integer
def winning_player_count(n, pick) do

end
end
```

Erlang:

```
-spec winning_player_count(N :: integer(), Pick :: [[integer()]]) ->
integer().
winning_player_count(N, Pick) ->
.
```

Racket:

```
(define/contract (winning-player-count n pick)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Number of Winning Players
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int winningPlayerCount(int n, vector<vector<int>>& pick) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Number of Winning Players
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int winningPlayerCount(int n, int[][] pick) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Number of Winning Players
Difficulty: Easy
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def winningPlayerCount(self, n: int, pick: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def winningPlayerCount(self, n, pick):
"""
:type n: int
:type pick: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Number of Winning Players
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} n
* @param {number[][]} pick
* @return {number}
*/
```

```
var winningPlayerCount = function(n, pick) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Number of Winning Players
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function winningPlayerCount(n: number, pick: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Find the Number of Winning Players
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int WinningPlayerCount(int n, int[][] pick) {

}
}
```

C Solution:

```
/*
* Problem: Find the Number of Winning Players
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int winningPlayerCount(int n, int** pick, int pickSize, int* pickColSize) {

}
```

Go Solution:

```
// Problem: Find the Number of Winning Players
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func winningPlayerCount(n int, pick [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun winningPlayerCount(n: Int, pick: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func winningPlayerCount(_ n: Int, _ pick: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find the Number of Winning Players
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn winning_player_count(n: i32, pick: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} pick
# @return {Integer}
def winning_player_count(n, pick)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $pick
* @return Integer
*/
function winningPlayerCount($n, $pick) {

}
}
```

Dart Solution:

```
class Solution {
int winningPlayerCount(int n, List<List<int>> pick) {

}
}
```

Scala Solution:

```
object Solution {
def winningPlayerCount(n: Int, pick: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec winning_player_count(n :: integer, pick :: [[integer]]) :: integer
def winning_player_count(n, pick) do

end
end
```

Erlang Solution:

```
-spec winning_player_count(N :: integer(), Pick :: [[integer()]]) ->
integer().
winning_player_count(N, Pick) ->
.
```

Racket Solution:

```
(define/contract (winning-player-count n pick)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```