Problem 3175: Find The First Player to win K Games in a Row

Problem Information

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

Problem Description

A competition consists of

n

players numbered from

0

to

n - 1

.

You are given an integer array

skills

of size

n

and a

positive

integer

k

, where

skills[i]

is the skill level of player

i

. All integers in

skills

are

unique

.

All players are standing in a queue in order from player

0

to player

n - 1

.

The competition process is as follows:

The first two players in the queue play a game, and the player with the

higher

skill level wins.

After the game, the winner stays at the beginning of the queue, and the loser goes to the end of it.

The winner of the competition is the

first

player who wins

k

games

in a row

.

Return the initial index of the

winning

player.

Example 1:

Input:

skills = [4,2,6,3,9], k = 2

Output:

2

Explanation:

Initially, the queue of players is

[0,1,2,3,4]

. The following process happens:

Players 0 and 1 play a game, since the skill of player 0 is higher than that of player 1, player 0 wins. The resulting queue is

[0,2,3,4,1]

.

Players 0 and 2 play a game, since the skill of player 2 is higher than that of player 0, player 2 wins. The resulting queue is

[2,3,4,1,0]

.

Players 2 and 3 play a game, since the skill of player 2 is higher than that of player 3, player 2 wins. The resulting queue is

[2,4,1,0,3]

.

Player 2 won

k = 2

games in a row, so the winner is player 2.

Example 2:

Input:

skills = [2,5,4], k = 3

Output:

1

Explanation:

Initially, the queue of players is

[0,1,2]

. The following process happens:

Players 0 and 1 play a game, since the skill of player 1 is higher than that of player 0, player 1 wins. The resulting queue is

[1,2,0]

.

Players 1 and 2 play a game, since the skill of player 1 is higher than that of player 2, player 1 wins. The resulting queue is

[1,0,2]

.

Players 1 and 0 play a game, since the skill of player 1 is higher than that of player 0, player 1 wins. The resulting queue is

[1,2,0]

.

Player 1 won

k = 3

games in a row, so the winner is player 1.

Constraints:

n == skills.length

2 <= n <= 10

5

1 <= k <= 10

9

1 <= skills[i] <= 10

6

All integers in

skills

are unique.

Code Snippets

C++:

```
class Solution {
public:
int findWinningPlayer(vector<int>& skills, int k) {

}
};
```

Java:

```
class Solution {
public int findWinningPlayer(int[] skills, int k) {

}
}
```

Python3:

```
class Solution:
def findWinningPlayer(self, skills: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def findWinningPlayer(self, skills, k):
"""
:type skills: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} skills
* @param {number} k
* @return {number}
*/
var findWinningPlayer = function(skills, k) {

};
```

TypeScript:

```
function findWinningPlayer(skills: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int FindWinningPlayer(int[] skills, int k) {

}
}
```

C:

```
int findWinningPlayer(int* skills, int skillsSize, int k) {

}
```

Go:

```
func findWinningPlayer(skills []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun findWinningPlayer(skills: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func findWinningPlayer(_ skills: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_winning_player(skills: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} skills
# @param {Integer} k
# @return {Integer}
def find_winning_player(skills, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $skills
* @param Integer $k
* @return Integer
*/
function findWinningPlayer($skills, $k) {

}
}
```

Dart:

```
class Solution {
int findWinningPlayer(List<int> skills, int k) {

}
}
```

Scala:

```
object Solution {
def findWinningPlayer(skills: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_winning_player(skills :: [integer], k :: integer) :: integer
def find_winning_player(skills, k) do

end
end
```

Erlang:

```
-spec find_winning_player(Skills :: [integer()], K :: integer()) ->
integer().
```

```
find_winning_player(Skills, K) ->
.
```

Racket:

```
(define/contract (find-winning-player skills k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find The First Player to win K Games in a Row
* Difficulty: Medium
* Tags: array, queue
*
* 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 findWinningPlayer(vector<int>& skills, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find The First Player to win K Games in a Row
* Difficulty: Medium
* Tags: array, queue
*
* 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 findWinningPlayer(int[] skills, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find The First Player to win K Games in a Row
Difficulty: Medium
Tags: array, queue

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 findWinningPlayer(self, skills: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findWinningPlayer(self, skills, k):
"""
:type skills: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find The First Player to win K Games in a Row
* Difficulty: Medium
* Tags: array, queue
```

```
*
* 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[]} skills
* @param {number} k
* @return {number}
*/
var findWinningPlayer = function(skills, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find The First Player to win K Games in a Row
* Difficulty: Medium
* Tags: array, queue
*
* 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 findWinningPlayer(skills: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find The First Player to win K Games in a Row
* Difficulty: Medium
* Tags: array, queue
*
* 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 FindWinningPlayer(int[] skills, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find The First Player to win K Games in a Row
* Difficulty: Medium
* Tags: array, queue
*
* 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 findWinningPlayer(int* skills, int skillsSize, int k) {

}
```

Go Solution:

```
// Problem: Find The First Player to win K Games in a Row
// Difficulty: Medium
// Tags: array, queue
//
// 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 findWinningPlayer(skills []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findWinningPlayer(skills: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findWinningPlayer(_ skills: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find The First Player to win K Games in a Row
// Difficulty: Medium
// Tags: array, queue
//
// 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 find_winning_player(skills: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} skills
# @param {Integer} k
# @return {Integer}
def find_winning_player(skills, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $skills
* @param Integer $k
* @return Integer
*/
function findWinningPlayer($skills, $k) {

}
}
```

Dart Solution:

```
class Solution {
int findWinningPlayer(List<int> skills, int k) {

}
}
```

Scala Solution:

```
object Solution {
def findWinningPlayer(skills: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_winning_player(skills :: [integer], k :: integer) :: integer
def find_winning_player(skills, k) do

end
end
```

Erlang Solution:

```
-spec find_winning_player(Skills :: [integer()], K :: integer()) ->
integer().
find_winning_player(Skills, K) ->
```

```
.
```

Racket Solution:

```
(define/contract (find-winning-player skills k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```