Problem 3096: Minimum Levels to Gain More Points

Problem Information

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

Problem Description

You are given a binary array

possible

of length

n

.

Alice and Bob are playing a game that consists of

n

levels. Some of the levels in the game are

impossible

to clear while others can

always

be cleared. In particular, if

possible[i] == 0

, then the

i

th

level is

impossible

to clear for

both

the players. A player gains

1

point on clearing a level and loses

1

point if the player fails to clear it.

At the start of the game, Alice will play some levels in the

given order

starting from the

0

th

level, after which Bob will play for the rest of the levels.

Alice wants to know the

minimum

number of levels she should play to gain more points than Bob, if both players play optimally to

maximize

their points.

Return

the

minimum

number of levels Alice should play to gain more points

.

If this is

not

possible, return

-1

.

Note

that each player must play at least

1

level.

Example 1:

Input:

possible = [1,0,1,0]

Output:

1

Explanation:

Let's look at all the levels that Alice can play up to:

If Alice plays only level 0 and Bob plays the rest of the levels, Alice has 1 point, while Bob has -1 + 1 - 1 = -1 point.

If Alice plays till level 1 and Bob plays the rest of the levels, Alice has 1 - 1 = 0 points, while Bob has 1 - 1 = 0 points.

If Alice plays till level 2 and Bob plays the rest of the levels, Alice has 1 - 1 + 1 = 1 point, while Bob has -1 point.

Alice must play a minimum of 1 level to gain more points.

Example 2:

Input:

possible = [1,1,1,1,1]

Output:

3

Explanation:

Let's look at all the levels that Alice can play up to:

If Alice plays only level 0 and Bob plays the rest of the levels, Alice has 1 point, while Bob has 4 points.

If Alice plays till level 1 and Bob plays the rest of the levels, Alice has 2 points, while Bob has 3 points.

If Alice plays till level 2 and Bob plays the rest of the levels, Alice has 3 points, while Bob has 2 points.

If Alice plays till level 3 and Bob plays the rest of the levels, Alice has 4 points, while Bob has 1 point.

Alice must play a minimum of 3 levels to gain more points.

Example 3:

Input:

possible = [0,0]

Output:

-1

Explanation:

The only possible way is for both players to play 1 level each. Alice plays level 0 and loses 1 point. Bob plays level 1 and loses 1 point. As both players have equal points, Alice can't gain more points than Bob.

Constraints:

2 <= n == possible.length <= 10

5

possible[i]

is either

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
int minimumLevels(vector<int>& possible) {

}
};
```

Java:

```
class Solution {
public int minimumLevels(int[] possible) {

}
}
```

Python3:

```
class Solution:
def minimumLevels(self, possible: List[int]) -> int:
```

Python:

```
class Solution(object):
def minimumLevels(self, possible):
"""
:type possible: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} possible
* @return {number}
*/
var minimumLevels = function(possible) {

};
```

TypeScript:

```
function minimumLevels(possible: number[]): number {

};
```

C#:

```
public class Solution {
public int MinimumLevels(int[] possible) {

}
}
```

C:

```
int minimumLevels(int* possible, int possibleSize) {

}
```

Go:

```
func minimumLevels(possible []int) int {

}
```

Kotlin:

```
class Solution {
fun minimumLevels(possible: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minimumLevels(_ possible: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_levels(possible: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} possible
# @return {Integer}
def minimum_levels(possible)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $possible
* @return Integer
*/
function minimumLevels($possible) {

}
}
```

Dart:

```
class Solution {
int minimumLevels(List<int> possible) {

}
}
```

Scala:

```
object Solution {
def minimumLevels(possible: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_levels(possible :: [integer]) :: integer
def minimum_levels(possible) do

end
end
```

Erlang:

```
-spec minimum_levels(Possible :: [integer()]) -> integer().
minimum_levels(Possible) ->
.
```

Racket:

```
(define/contract (minimum-levels possible)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Levels to Gain More Points
* Difficulty: Medium
* Tags: array
*
* 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 minimumLevels(vector<int>& possible) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Levels to Gain More Points
* Difficulty: Medium
* Tags: array
*
* 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 minimumLevels(int[] possible) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Levels to Gain More Points
Difficulty: Medium
Tags: array

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 minimumLevels(self, possible: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def minimumLevels(self, possible):
"""
:type possible: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Levels to Gain More Points
* Difficulty: Medium
* Tags: array
*
* 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[]} possible
* @return {number}
*/
var minimumLevels = function(possible) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Levels to Gain More Points
* Difficulty: Medium
* Tags: array
*
* 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 minimumLevels(possible: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Levels to Gain More Points
* Difficulty: Medium
* Tags: array
*
* 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 MinimumLevels(int[] possible) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Levels to Gain More Points
* Difficulty: Medium
* Tags: array
*
* 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 minimumLevels(int* possible, int possibleSize) {

}
```

Go Solution:

```
// Problem: Minimum Levels to Gain More Points
// Difficulty: Medium
// Tags: array
//
// 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 minimumLevels(possible []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumLevels(possible: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumLevels(_ possible: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Levels to Gain More Points
// Difficulty: Medium
// Tags: array
//
// 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 minimum_levels(possible: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[]} possible
# @return {Integer}
def minimum_levels(possible)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $possible
* @return Integer
*/
function minimumLevels($possible) {

}
}
```

Dart Solution:

```
class Solution {
int minimumLevels(List<int> possible) {

}
}
```

Scala Solution:

```
object Solution {
def minimumLevels(possible: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_levels(possible :: [integer]) :: integer
def minimum_levels(possible) do

end
end
```

Erlang Solution:

```
-spec minimum_levels(Possible :: [integer()]) -> integer().
minimum_levels(Possible) ->
.
```

Racket Solution:

```
(define/contract (minimum-levels possible)
(-> (listof exact-integer?) exact-integer?)
)
```