Problem 1051: Height Checker

Problem Information

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

Problem Description

A school is trying to take an annual photo of all the students. The students are asked to stand in a single file line in

non-decreasing order

by height. Let this ordering be represented by the integer array

expected

where

expected[i]

is the expected height of the

i

th

student in line.

You are given an integer array

heights

representing the

current order

that the students are standing in. Each

heights[i]

is the height of the

i

th

student in line (

0-indexed

).

Return

the

number of indices

where

heights[i] != expected[i]

.

Example 1:

Input:

heights = [1,1,4,2,1,3]

Output:

3

Explanation:

heights: [1,1,

4

,2,

1

,

3

] expected: [1,1,

1

,2,

3

,

4

] Indices 2, 4, and 5 do not match.

Example 2:

Input:

heights = [5,1,2,3,4]

Output:

5

Explanation:

heights: [

5

,

1

,

2

,

3

,

4

] expected: [

1

,

2

,

3

,

4

,

5

] All indices do not match.

Example 3:

Input:

heights = [1,2,3,4,5]

Output:

0

Explanation:

heights: [1,2,3,4,5] expected: [1,2,3,4,5] All indices match.

Constraints:

1 <= heights.length <= 100

1 <= heights[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int heightChecker(vector<int>& heights) {

}
};
```

Java:

```
class Solution {
public int heightChecker(int[] heights) {

}
}
```

Python3:

```
class Solution:
def heightChecker(self, heights: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function heightChecker(heights: number[]): number {

};
```

C#:

```
public class Solution {
public int HeightChecker(int[] heights) {

}
}
```

C:

```
int heightChecker(int* heights, int heightsSize) {

}
```

Go:

```
func heightChecker(heights []int) int {

}
```

Kotlin:

```
class Solution {
fun heightChecker(heights: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func heightChecker(_ heights: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn height_checker(heights: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} heights
# @return {Integer}
def height_checker(heights)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $heights
* @return Integer
*/
function heightChecker($heights) {

}
}
```

Dart:

```
class Solution {
int heightChecker(List<int> heights) {

}
}
```

Scala:

```
object Solution {
def heightChecker(heights: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec height_checker(heights :: [integer]) :: integer
def height_checker(heights) do

end
end
```

Erlang:

```
-spec height_checker(Heights :: [integer()]) -> integer().
height_checker(Heights) ->
.
```

Racket:

```
(define/contract (height-checker heights)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Height Checker
* Difficulty: Easy
* Tags: array, 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 heightChecker(vector<int>& heights) {

}
};
```

Java Solution:

```
/**
* Problem: Height Checker
* Difficulty: Easy
* Tags: array, 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 heightChecker(int[] heights) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Height Checker
Difficulty: Easy
Tags: array, 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 heightChecker(self, heights: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Height Checker
* Difficulty: Easy
* Tags: array, 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[]} heights
* @return {number}
*/
var heightChecker = function(heights) {

};
```

TypeScript Solution:

```
/**
* Problem: Height Checker
* Difficulty: Easy
* Tags: array, 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 heightChecker(heights: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Height Checker
* Difficulty: Easy
* Tags: array, 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 HeightChecker(int[] heights) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Height Checker
* Difficulty: Easy
* Tags: array, 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 heightChecker(int* heights, int heightsSize) {

}
```

Go Solution:

```
// Problem: Height Checker
// Difficulty: Easy
// Tags: array, 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 heightChecker(heights []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun heightChecker(heights: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func heightChecker(_ heights: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Height Checker
// Difficulty: Easy
// Tags: array, 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 height_checker(heights: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} heights
# @return {Integer}
def height_checker(heights)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $heights
* @return Integer
*/
function heightChecker($heights) {

}
}
```

Dart Solution:

```
class Solution {
int heightChecker(List<int> heights) {

}
}
```

Scala Solution:

```
object Solution {
def heightChecker(heights: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec height_checker(heights :: [integer]) :: integer
def height_checker(heights) do

end
end
```

Erlang Solution:

```
-spec height_checker(Heights :: [integer()]) -> integer().
height_checker(Heights) ->
.
```

Racket Solution:

```
(define/contract (height-checker heights)
(-> (listof exact-integer?) exact-integer?)
)
```