Problem 1944: Number of Visible People in a Queue

Problem Information

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

Problem Description

There are

n

people standing in a queue, and they numbered from

0

to

n - 1

in

left to right

order. You are given an array

heights

of

distinct

integers where

heights[i]

represents the height of the

i

th

person.

A person can

see

another person to their right in the queue if everybody in between is

shorter

than both of them. More formally, the

i

th

person can see the

j

th

person if

i < j

and

min(heights[i], heights[j]) > max(heights[i+1], heights[i+2], ..., heights[j-1])

.

Return

an array

answer

of length

n

where

answer[i]

is the

number of people

the

i

th

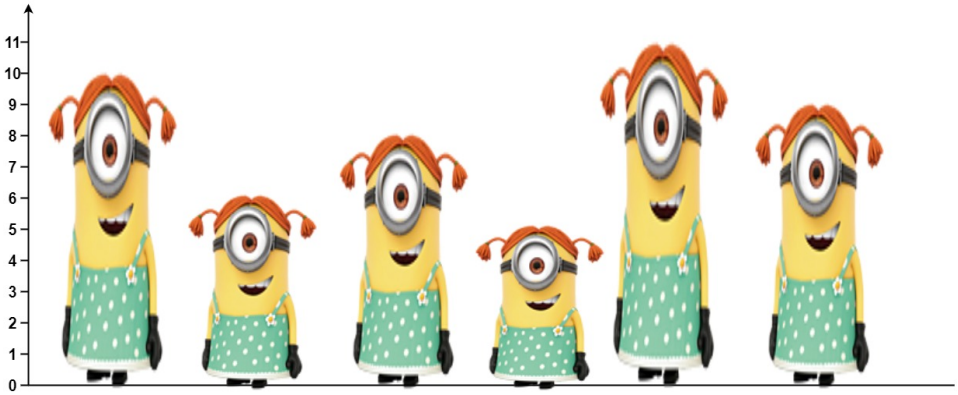
person can

see

to their right in the queue

.

Example 1:



Input:

heights = [10,6,8,5,11,9]

Output:

[3,1,2,1,1,0]

Explanation:

Person 0 can see person 1, 2, and 4. Person 1 can see person 2. Person 2 can see person 3 and 4. Person 3 can see person 4. Person 4 can see person 5. Person 5 can see no one since nobody is to the right of them.

Example 2:

Input:

heights = [5,1,2,3,10]

Output:

[4,1,1,1,0]

Constraints:

n == heights.length

1 <= n <= 10

5

1 <= heights[i] <= 10

5

All the values of

heights

are

unique

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* canSeePersonsCount(int* heights, int heightsSize, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun canSeePersonsCount(heights: IntArray): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $heights
* @return Integer[]
```

```
*/
function canSeePersonsCount($heights) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (can-see-persons-count heights)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Visible People in a Queue
* Difficulty: Hard
* Tags: array, stack, 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:
vector<int> canSeePersonsCount(vector<int>& heights) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Visible People in a Queue
* Difficulty: Hard
* Tags: array, stack, 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[] canSeePersonsCount(int[] heights) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Visible People in a Queue
Difficulty: Hard
Tags: array, stack, 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 canSeePersonsCount(self, heights: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Visible People in a Queue
* Difficulty: Hard
* Tags: array, stack, 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[]} heights
* @return {number[]}
*/
var canSeePersonsCount = function(heights) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Number of Visible People in a Queue
* Difficulty: Hard
* Tags: array, stack, 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 canSeePersonsCount(heights: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Number of Visible People in a Queue
* Difficulty: Hard
* Tags: array, stack, 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[] CanSeePersonsCount(int[] heights) {

}
}
```

C Solution:

```
/*
* Problem: Number of Visible People in a Queue
* Difficulty: Hard
```

```
* Tags: array, stack, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* canSeePersonsCount(int* heights, int heightsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Number of Visible People in a Queue
// Difficulty: Hard
// Tags: array, stack, 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 canSeePersonsCount(heights []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun canSeePersonsCount(heights: IntArray): IntArray {

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Number of Visible People in a Queue
// Difficulty: Hard
// Tags: array, stack, 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 can_see_persons_count(heights: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (can-see-persons-count heights)
(-> (listof exact-integer?) (listof exact-integer?))
)
```