Problem 1964: Find the Longest Valid Obstacle Course at Each Position

Problem Information

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

Problem Description

You want to build some obstacle courses. You are given a

0-indexed

integer array

obstacles

of length

n

, where

obstacles[i]

describes the height of the

i

th

obstacle.

For every index

i

between

0

and

n - 1

(

inclusive

), find the length of the

longest obstacle course

in

obstacles

such that:

You choose any number of obstacles between

0

and

i

inclusive

.

You must include the

i

th

obstacle in the course.

You must put the chosen obstacles in the

same order

as they appear in

obstacles

.

Every obstacle (except the first) is

taller

than or the

same height

as the obstacle immediately before it.

Return

an array

ans

of length

n

,

where

ans[i]

is the length of the

longest obstacle course

for index

i

as described above

.

Example 1:

Input:

obstacles = [1,2,3,2]

Output:

[1,2,3,3]

Explanation:

The longest valid obstacle course at each position is: - i = 0: [

1

], [1] has length 1. - i = 1: [

1

,

2

], [1,2] has length 2. - i = 2: [

1

,

2

,

3

], [1,2,3] has length 3. - i = 3: [

1

,

2

,3,

2

], [1,2,2] has length 3.

Example 2:

Input:

obstacles = [2,2,1]

Output:

[1,2,1]

Explanation:

The longest valid obstacle course at each position is: - i = 0: [

2

], [2] has length 1. - i = 1: [

2

,

2

], [2,2] has length 2. - i = 2: [2,2,

1

], [1] has length 1.

Example 3:

Input:

obstacles = [3,1,5,6,4,2]

Output:

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

Explanation:

The longest valid obstacle course at each position is: - i = 0: [

3

], [3] has length 1. - i = 1: [3,

1

], [1] has length 1. - i = 2: [

3

,1,

5

], [3,5] has length 2. [1,5] is also valid. - i = 3: [

3

,1,

5

,

6

], [3,5,6] has length 3. [1,5,6] is also valid. - i = 4: [

3

,1,5,6,

4

], [3,4] has length 2. [1,4] is also valid. - i = 5: [3,

1

,5,6,4,

2

], [1,2] has length 2.

Constraints:

n == obstacles.length

1 <= n <= 10

5

1 <= obstacles[i] <= 10

7

Code Snippets

C++:

```
class Solution {
public:
vector<int> longestObstacleCourseAtEachPosition(vector<int>& obstacles) {

}
};
```

Java:

```
class Solution {
public int[] longestObstacleCourseAtEachPosition(int[] obstacles) {

}
}
```

Python3:

```
class Solution:
def longestObstacleCourseAtEachPosition(self, obstacles: List[int]) ->
List[int]:
```

Python:

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

```
"""
```

JavaScript:

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

};
```

TypeScript:

```
function longestObstacleCourseAtEachPosition(obstacles: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] LongestObstacleCourseAtEachPosition(int[] obstacles) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* longestObstacleCourseAtEachPosition(int* obstacles, int obstaclesSize,
int* returnSize) {

}
```

Go:

```
func longestObstacleCourseAtEachPosition(obstacles []int) []int {

}
```

Kotlin:

```
class Solution {
fun longestObstacleCourseAtEachPosition(obstacles: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func longestObstacleCourseAtEachPosition(_ obstacles: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn longest_obstacle_course_at_each_position(obstacles: Vec<i32>) ->
Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} obstacles
# @return {Integer[]}
def longest_obstacle_course_at_each_position(obstacles)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $obstacles
* @return Integer[]
*/
function longestObstacleCourseAtEachPosition($obstacles) {
```

```
}
}
```

Dart:

```
class Solution {
List<int> longestObstacleCourseAtEachPosition(List<int> obstacles) {

}
}
```

Scala:

```
object Solution {
def longestObstacleCourseAtEachPosition(obstacles: Array[Int]): Array[Int] =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_obstacle_course_at_each_position(obstacles :: [integer]) ::
[integer]
def longest_obstacle_course_at_each_position(obstacles) do

end
end
```

Erlang:

```
-spec longest_obstacle_course_at_each_position(Obstacles :: [integer()]) ->
[integer()].
longest_obstacle_course_at_each_position(Obstacles) ->
.
```

Racket:

```
(define/contract (longest-obstacle-course-at-each-position obstacles)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Longest Valid Obstacle Course at Each Position
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<int> longestObstacleCourseAtEachPosition(vector<int>& obstacles) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Longest Valid Obstacle Course at Each Position
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int[] longestObstacleCourseAtEachPosition(int[] obstacles) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Longest Valid Obstacle Course at Each Position
Difficulty: Hard
Tags: array, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Longest Valid Obstacle Course at Each Position
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

```
};
```

TypeScript Solution:

```
/**
* Problem: Find the Longest Valid Obstacle Course at Each Position
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function longestObstacleCourseAtEachPosition(obstacles: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find the Longest Valid Obstacle Course at Each Position
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int[] LongestObstacleCourseAtEachPosition(int[] obstacles) {

}
}
```

C Solution:

```
/*
* Problem: Find the Longest Valid Obstacle Course at Each Position
```

```
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* longestObstacleCourseAtEachPosition(int* obstacles, int obstaclesSize,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Find the Longest Valid Obstacle Course at Each Position
// Difficulty: Hard
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func longestObstacleCourseAtEachPosition(obstacles []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun longestObstacleCourseAtEachPosition(obstacles: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func longestObstacleCourseAtEachPosition(_ obstacles: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find the Longest Valid Obstacle Course at Each Position
// Difficulty: Hard
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn longest_obstacle_course_at_each_position(obstacles: Vec<i32>) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} obstacles
# @return {Integer[]}
def longest_obstacle_course_at_each_position(obstacles)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $obstacles
* @return Integer[]
*/
function longestObstacleCourseAtEachPosition($obstacles) {

}
```

```
}
```

Dart Solution:

```
class Solution {
List<int> longestObstacleCourseAtEachPosition(List<int> obstacles) {

}
}
```

Scala Solution:

```
object Solution {
def longestObstacleCourseAtEachPosition(obstacles: Array[Int]): Array[Int] =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_obstacle_course_at_each_position(obstacles :: [integer]) ::
[integer]
def longest_obstacle_course_at_each_position(obstacles) do

end
end
```

Erlang Solution:

```
-spec longest_obstacle_course_at_each_position(Obstacles :: [integer()]) ->
[integer()].
longest_obstacle_course_at_each_position(Obstacles) ->
.
```

Racket Solution:

```
(define/contract (longest-obstacle-course-at-each-position obstacles)
(-> (listof exact-integer?) (listof exact-integer?))
)
```