Problem 1762: Buildings With an Ocean View

Problem Information

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

Problem Description

There are

n

buildings in a line. You are given an integer array

heights

of size

n

that represents the heights of the buildings in the line.

The ocean is to the right of the buildings. A building has an ocean view if the building can see the ocean without obstructions. Formally, a building has an ocean view if all the buildings to its right have a

smaller

height.

Return a list of indices

(0-indexed)

of buildings that have an ocean view, sorted in increasing order.

Example 1:

Input:

heights = [4,2,3,1]

Output:

[0,2,3]

Explanation:

Building 1 (0-indexed) does not have an ocean view because building 2 is taller.

Example 2:

Input:

heights = [4,3,2,1]

Output:

[0,1,2,3]

Explanation:

All the buildings have an ocean view.

Example 3:

Input:

heights = [1,3,2,4]

Output:

[3]

Explanation:

Only building 3 has an ocean view.

Constraints:

1 <= heights.length <= 10

5

1 <= heights[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-buildings heights)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Buildings With an Ocean View
* Difficulty: Medium
* Tags: array, tree, sort, stack
*
* 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> findBuildings(vector<int>& heights) {

}
};
```

Java Solution:

```
/**
* Problem: Buildings With an Ocean View
* Difficulty: Medium
* Tags: array, tree, sort, stack
*
* 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[] findBuildings(int[] heights) {

}
}
```

Python3 Solution:

```
"""
Problem: Buildings With an Ocean View
Difficulty: Medium
Tags: array, tree, sort, stack

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 findBuildings(self, heights: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Buildings With an Ocean View
* Difficulty: Medium
* Tags: array, tree, sort, stack
*
* 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[]} heights
* @return {number[]}
*/
var findBuildings = function(heights) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Buildings With an Ocean View
* Difficulty: Medium
* Tags: array, tree, sort, stack
*
* 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 findBuildings(heights: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Buildings With an Ocean View
* Difficulty: Medium
* Tags: array, tree, sort, stack
*
* 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[] FindBuildings(int[] heights) {

}
}
```

C Solution:

```
/*
* Problem: Buildings With an Ocean View
* Difficulty: Medium
```

```
* Tags: array, tree, sort, stack
*
* 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* findBuildings(int* heights, int heightsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Buildings With an Ocean View
// Difficulty: Medium
// Tags: array, tree, sort, stack
//
// 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 findBuildings(heights []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Buildings With an Ocean View
// Difficulty: Medium
// Tags: array, tree, sort, stack
//
// 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 find_buildings(heights: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-buildings heights)
(-> (listof exact-integer?) (listof exact-integer?))
)
```