Problem 2865: Beautiful Towers I

Problem Information

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

Problem Description

You are given an array

heights

of

n

integers representing the number of bricks in

n

consecutive towers. Your task is to remove some bricks to form a

mountain-shaped

tower arrangement. In this arrangement, the tower heights are non-decreasing, reaching a maximum peak value with one or multiple consecutive towers and then non-increasing.

Return the

maximum possible sum

of heights of a mountain-shaped tower arrangement.

Example 1:

Input:

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

Output:

13

Explanation:

We remove some bricks to make

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

, the peak is at index 0.

Example 2:

Input:

heights = [6,5,3,9,2,7]

Output:

22

Explanation:

We remove some bricks to make

heights = [3,3,3,9,2,2]

, the peak is at index 3.

Example 3:

Input:

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

Output:

18

Explanation:

We remove some bricks to make

heights = [2,2,5,5,2,2]

, the peak is at index 2 or 3.

Constraints:

1 <= n == heights.length <= 10

3

1 <= heights[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long maximumSumOfHeights(int[] heights) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long MaximumSumOfHeights(int[] heights) {

}
}
```

C:

```
long long maximumSumOfHeights(int* heights, int heightsSize) {

}
```

Go:

```
func maximumSumOfHeights(heights []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maximumSumOfHeights(heights: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn maximum_sum_of_heights(heights: Vec<i32>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximum-sum-of-heights heights)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Beautiful Towers I
* Difficulty: Medium
* Tags: array, stack
*
* 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:
long long maximumSumOfHeights(vector<int>& heights) {

}
};
```

Java Solution:

```
/**
* Problem: Beautiful Towers I
* Difficulty: Medium
* Tags: array, stack
*
* 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 long maximumSumOfHeights(int[] heights) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Beautiful Towers I
Difficulty: Medium
Tags: array, stack

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Beautiful Towers I
* Difficulty: Medium
* Tags: array, stack
*
* 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 maximumSumOfHeights = function(heights) {

};
```

TypeScript Solution:

```
/**
* Problem: Beautiful Towers I
* Difficulty: Medium
* Tags: array, stack
*
* 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 maximumSumOfHeights(heights: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Beautiful Towers I
* Difficulty: Medium
* Tags: array, stack
*
* 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 long MaximumSumOfHeights(int[] heights) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Beautiful Towers I
* Difficulty: Medium
* Tags: array, stack
*
* 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
*/

long long maximumSumOfHeights(int* heights, int heightsSize) {

}
```

Go Solution:

```
// Problem: Beautiful Towers I
// Difficulty: Medium
// Tags: array, stack
//
// 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 maximumSumOfHeights(heights []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maximumSumOfHeights(heights: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Beautiful Towers I
// Difficulty: Medium
// Tags: array, stack
//
// 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 maximum_sum_of_heights(heights: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximum-sum-of-heights heights)
(-> (listof exact-integer?) exact-integer?)
)
```