Problem 3301: Maximize the Total Height of Unique Towers

Problem Information

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

Problem Description

You are given an array

maximumHeight

, where

maximumHeight[i]

denotes the

maximum

height the

i

th

tower can be assigned.

Your task is to assign a height to each tower so that:

The height of the

i

th

tower is a positive integer and does not exceed

maximumHeight[i]

.

No two towers have the same height.

Return the

maximum

possible total sum of the tower heights. If it's not possible to assign heights, return

-1

.

Example 1:

Input:

maximumHeight

= [2,3,4,3]

Output:

10

Explanation:

We can assign heights in the following way:

[1, 2, 4, 3]

.

Example 2:

Input:

maximumHeight

= [15,10]

Output:

25

Explanation:

We can assign heights in the following way:

[15, 10]

.

Example 3:

Input:

maximumHeight

= [2,2,1]

Output:

-1

Explanation:

It's impossible to assign positive heights to each index so that no two towers have the same height.

Constraints:

1 <= maximumHeight.length <= 10

5

1 <= maximumHeight[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long maximumTotalSum(vector<int>& maximumHeight) {

}
};
```

Java:

```
class Solution {
public long maximumTotalSum(int[] maximumHeight) {

}
}
```

Python3:

```
class Solution:
def maximumTotalSum(self, maximumHeight: List[int]) -> int:
```

Python:

```
class Solution(object):
def maximumTotalSum(self, maximumHeight):
"""
:type maximumHeight: List[int]
```

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function maximumTotalSum(maximumHeight: number[]): number {

};
```

C#:

```
public class Solution {
public long MaximumTotalSum(int[] maximumHeight) {

}
}
```

C:

```
long long maximumTotalSum(int* maximumHeight, int maximumHeightSize) {

}
```

Go:

```
func maximumTotalSum(maximumHeight []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maximumTotalSum(maximumHeight: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func maximumTotalSum(_ maximumHeight: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_total_sum(maximum_height: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} maximum_height
# @return {Integer}
def maximum_total_sum(maximum_height)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $maximumHeight
* @return Integer
*/
function maximumTotalSum($maximumHeight) {

}
}
```

Dart:

```
class Solution {
int maximumTotalSum(List<int> maximumHeight) {

}
}
```

Scala:

```
object Solution {
def maximumTotalSum(maximumHeight: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_total_sum(maximum_height :: [integer]) :: integer
def maximum_total_sum(maximum_height) do

end
end
```

Erlang:

```
-spec maximum_total_sum(MaximumHeight :: [integer()]) -> integer().
maximum_total_sum(MaximumHeight) ->
.
```

Racket:

```
(define/contract (maximum-total-sum maximumHeight)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize the Total Height of Unique Towers
* Difficulty: Medium
* Tags: array, greedy, 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:
long long maximumTotalSum(vector<int>& maximumHeight) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize the Total Height of Unique Towers
* Difficulty: Medium
* Tags: array, greedy, 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 long maximumTotalSum(int[] maximumHeight) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize the Total Height of Unique Towers
Difficulty: Medium
Tags: array, greedy, 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 maximumTotalSum(self, maximumHeight: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize the Total Height of Unique Towers
* Difficulty: Medium
* Tags: array, greedy, 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[]} maximumHeight
* @return {number}
*/
var maximumTotalSum = function(maximumHeight) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize the Total Height of Unique Towers
* Difficulty: Medium
* Tags: array, greedy, 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 maximumTotalSum(maximumHeight: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximize the Total Height of Unique Towers
* Difficulty: Medium
* Tags: array, greedy, 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 long MaximumTotalSum(int[] maximumHeight) {

}
}
```

C Solution:

```
/*
* Problem: Maximize the Total Height of Unique Towers
* Difficulty: Medium
* Tags: array, greedy, 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
```

```
*/

long long maximumTotalSum(int* maximumHeight, int maximumHeightSize) {

}
```

Go Solution:

```
// Problem: Maximize the Total Height of Unique Towers
// Difficulty: Medium
// Tags: array, greedy, 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 maximumTotalSum(maximumHeight []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maximumTotalSum(maximumHeight: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func maximumTotalSum(_ maximumHeight: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximize the Total Height of Unique Towers
// Difficulty: Medium
// Tags: array, greedy, 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 maximum_total_sum(maximum_height: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} maximum_height
# @return {Integer}
def maximum_total_sum(maximum_height)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $maximumHeight
* @return Integer
*/
function maximumTotalSum($maximumHeight) {

}
}
```

Dart Solution:

```
class Solution {
int maximumTotalSum(List<int> maximumHeight) {

}
}
```

Scala Solution:

```
object Solution {
def maximumTotalSum(maximumHeight: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_total_sum(maximum_height :: [integer]) :: integer
def maximum_total_sum(maximum_height) do

end
end
```

Erlang Solution:

```
-spec maximum_total_sum(MaximumHeight :: [integer()]) -> integer().
maximum_total_sum(MaximumHeight) ->
.
```

Racket Solution:

```
(define/contract (maximum-total-sum maximumHeight)
(-> (listof exact-integer?) exact-integer?)
)
```