Problem 2021: Brightest Position on Street

Problem Information

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

Problem Description

A perfectly straight street is represented by a number line. The street has street lamp(s) on it and is represented by a 2D integer array

lights

. Each

lights[i] = [position

i

, range

i

]

indicates that there is a street lamp at position

position

i

that lights up the area from

[position

i

- range

i

, position

i

+ range

i

]

(

inclusive

).

The

brightness

of a position

p

is defined as the number of street lamp that light up the position

p

.

Given

lights

, return

the

brightest

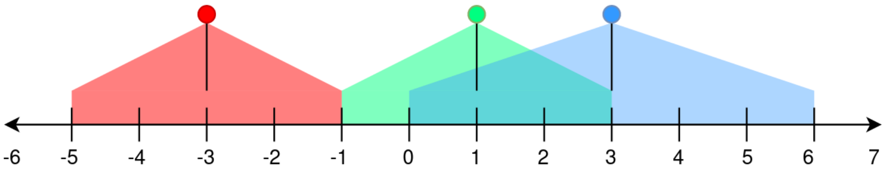
position on the

street. If there are multiple brightest positions, return the

smallest

one.

Example 1:



Input:

lights = [[-3,2],[1,2],[3,3]]

Output:

-1

Explanation:

The first street lamp lights up the area from [(-3) - 2, (-3) + 2] = [-5, -1]. The second street lamp lights up the area from [1 - 2, 1 + 2] = [-1, 3]. The third street lamp lights up the area from [3 - 3, 3 + 3] = [0, 6].

Position -1 has a brightness of 2, illuminated by the first and second street light. Positions 0, 1, 2, and 3 have a brightness of 2, illuminated by the second and third street light. Out of all these positions, -1 is the smallest, so return it.

Example 2:

Input:

lights = [[1,0],[0,1]]

Output:

1

Explanation:

The first street lamp lights up the area from [1 - 0, 1 + 0] = [1, 1]. The second street lamp lights up the area from [0 - 1, 0 + 1] = [-1, 1].

Position 1 has a brightness of 2, illuminated by the first and second street light. Return 1 because it is the brightest position on the street.

Example 3:

Input:

lights = [[1,2]]

Output:

-1

Explanation:

The first street lamp lights up the area from [1 - 2, 1 + 2] = [-1, 3].

Positions -1, 0, 1, 2, and 3 have a brightness of 1, illuminated by the first street light. Out of all these positions, -1 is the smallest, so return it.

Constraints:

1 <= lights.length <= 10

5

lights[i].length == 2

-10

8

<= position

i

<= 10

8

0 <= range

i

<= 10

8

Code Snippets

C++:

```
class Solution {
public:
int brightestPosition(vector<vector<int>>& lights) {

}
};
```

Java:

```
class Solution {
public int brightestPosition(int[][] lights) {

}
}
```

Python3:

```
class Solution:
def brightestPosition(self, lights: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function brightestPosition(lights: number[][]): number {

};
```

C#:

```
public class Solution {
public int BrightestPosition(int[][] lights) {

}
}
```

C:

```
int brightestPosition(int** lights, int lightsSize, int* lightsColSize) {

}
```

Go:

```
func brightestPosition(lights [][]int) int {

}
```

Kotlin:

```
class Solution {
fun brightestPosition(lights: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func brightestPosition(_ lights: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn brightest_position(lights: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} lights
# @return {Integer}
def brightest_position(lights)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $lights
* @return Integer
*/
function brightestPosition($lights) {

}
}
```

Dart:

```
class Solution {
int brightestPosition(List<List<int>> lights) {

}
}
```

Scala:

```
object Solution {
def brightestPosition(lights: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec brightest_position(lights :: [[integer]]) :: integer
def brightest_position(lights) do

end
end
```

Erlang:

```
-spec brightest_position(Lights :: [[integer()]]) -> integer().
brightest_position(Lights) ->
.
```

Racket:

```
(define/contract (brightest-position lights)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Brightest Position on Street
* Difficulty: Medium
* Tags: array, tree, sort
*
* 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 brightestPosition(vector<vector<int>>& lights) {

}
};
```

Java Solution:

```
/**
* Problem: Brightest Position on Street
* Difficulty: Medium
* Tags: array, tree, sort
*
* 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 brightestPosition(int[][] lights) {

}
}
```

Python3 Solution:

```
"""
Problem: Brightest Position on Street
Difficulty: Medium
Tags: array, tree, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Brightest Position on Street
* Difficulty: Medium
* Tags: array, tree, sort
*
* 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[][]} lights
* @return {number}
*/
var brightestPosition = function(lights) {

};
```

TypeScript Solution:

```
/**
* Problem: Brightest Position on Street
* Difficulty: Medium
* Tags: array, tree, sort
*
* 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 brightestPosition(lights: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Brightest Position on Street
* Difficulty: Medium
* Tags: array, tree, sort
*
* 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 BrightestPosition(int[][] lights) {

}
}
```

C Solution:

```
/*
* Problem: Brightest Position on Street
* Difficulty: Medium
* Tags: array, tree, sort
*
* 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
*/

int brightestPosition(int** lights, int lightsSize, int* lightsColSize) {

}
```

Go Solution:

```
// Problem: Brightest Position on Street
// Difficulty: Medium
// Tags: array, tree, sort
//
// 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 brightestPosition(lights [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun brightestPosition(lights: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func brightestPosition(_ lights: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Brightest Position on Street
// Difficulty: Medium
// Tags: array, tree, sort
//
// 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 brightest_position(lights: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} lights
# @return {Integer}
def brightest_position(lights)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[][] $lights
* @return Integer
*/
function brightestPosition($lights) {

}
}
```

Dart Solution:

```
class Solution {
int brightestPosition(List<List<int>> lights) {

}
}
```

Scala Solution:

```
object Solution {
def brightestPosition(lights: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec brightest_position(lights :: [[integer]]) :: integer
def brightest_position(lights) do

end
end
```

Erlang Solution:

```
-spec brightest_position(Lights :: [[integer()]]) -> integer().
brightest_position(Lights) ->
.
```

Racket Solution:

```
(define/contract (brightest-position lights)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```