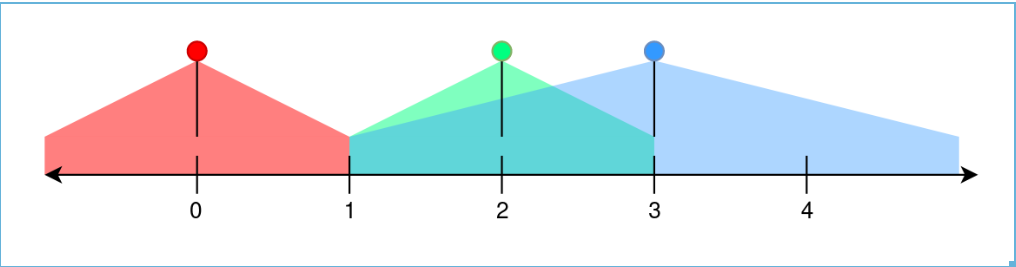
<https://leetcode.ca/2022-05-09-2237-Count-Positions-on-Street-With-Required-Brightness/>

You are given an integer n. A perfectly straight street is represented by a number line ranging from 0 to n - 1. You are given a 2D integer array lights representing the street lamp(s) on the street. Each lights[i] = [positioni, rangei] indicates that there is a street lamp at position positioni that lights up the area from [max(0, positioni - rangei), min(n - 1, positioni + rangei)] (**inclusive**).

The **brightness** of a position p is defined as the number of street lamps that light up the position p. You are given a **0-indexed** integer array requirement of size n where requirement[i] is the minimum **brightness** of the ith position on the street.

Return *the number of positions* i *on the street between* 0 *and* n - 1 *that have a* ***brightness*** *of* ***at least***requirement[i]*.*

**Example 1:**



Input: n = 5, lights = [[0,1],[2,1],[3,2]], requirement = [0,2,1,4,1]

Output: 4

Explanation:

- The first street lamp lights up the area from [max(0, 0 - 1), min(n - 1, 0 + 1)] = [0, 1] (inclusive).

- The second street lamp lights up the area from [max(0, 2 - 1), min(n - 1, 2 + 1)] = [1, 3] (inclusive).

- The third street lamp lights up the area from [max(0, 3 - 2), min(n - 1, 3 + 2)] = [1, 4] (inclusive).

- Position 0 is covered by the first street lamp. It is covered by 1 street lamp which is greater than requirement[0].

- Position 1 is covered by the first, second, and third street lamps. It is covered by 3 street lamps which is greater than requirement[1].

- Position 2 is covered by the second and third street lamps. It is covered by 2 street lamps which is greater than requirement[2].

- Position 3 is covered by the second and third street lamps. It is covered by 2 street lamps which is less than requirement[3].

- Position 4 is covered by the third street lamp. It is covered by 1 street lamp which is equal to requirement[4].

Positions 0, 1, 2, and 4 meet the requirement so we return 4.

**Example 2:**

Input: n = 1, lights = [[0,1]], requirement = [2]

Output: 0

Explanation:

- The first street lamp lights up the area from [max(0, 0 - 1), min(n - 1, 0 + 1)] = [0, 0] (inclusive).

- Position 0 is covered by the first street lamp. It is covered by 1 street lamp which is less than requirement[0].

- We return 0 because no position meets their brightness requirement.

**Constraints:**

* 1 <= n <= 10^5
* 1 <= lights.length <= 10^5
* 0 <= positioni < n
* 0 <= rangei <= 10^5
* requirement.length == n
* 0 <= requirement[i] <= 10^5

**Attempt 1: 2023-12-03**

**Solution 1: Sweep Line (10 min)**

**The same way for 'end + 1' when handling L2848.Points That Intersect With Cars**

class Solution {

public int meetRequirement(int n, int[][] lights, int[] requirement) {

// Additional '+1' to make sure includes corner case for light range

// includes index n - 1 and delta array 'timeline' should decrease on

// index n

int[] timeline = new int[n + 1];

for(int i = 0; i < lights.length; i++) {

int start = Math.max(0, lights[i][0] - lights[i][1]);

int end = Math.min(n - 1, lights[i][0] + lights[i][1]);

timeline[start]++;

timeline[end + 1]--;

}

int count = 0;

int result = 0;

for(int i = 0; i < n; i++) {

count += timeline[i];

if(count >= requirement[i]) {

result++;

}

}

return result;

}

}

Time Complexity: O(N)

Space Complexity: O(N)

**Refer to**

<https://algo.monster/liteproblems/2237>

## Problem Description

You are provided with an integer n which represents the length of a street on a number line from 0 to n - 1. There are street lamps along this street, given by a 2D integer array named lights. Each element of lights is a pair [position\_i, range\_i], where position\_i denotes the location of a street lamp and range\_i is the range of its light. This range means the lamp can illuminate the street from [max(0, position\_i - range\_i), min(n - 1, position\_i + range\_i)] (both ends included).

The brightness at any position p on the street is defined as the count of street lamps that cover the position p. A separate 0-indexed integer array named requirement specifies the minimum brightness required at each position i of the street.

The goal is to find and return the count of positions i on the street (between 0 to n - 1) where the brightness is at least the specified requirement at i.

## Intuition

The intuition of the solution involves understanding that we need to calculate the brightness at each position on the street and then check if it meets the requirement for that position. A key observation here is that by turning a lamp on, it increases the brightness of all positions within its range; similarly, turning it off would decrease the brightness. This is similar to an interval update in a range query problem.

The challenge then is how to efficiently calculate the brightness at all positions, given that each lamp can affect a potentially broad range and there can be multiple overlaps from different lamps. Directly updating each range for each lamp would lead to a solution that is too slow, as each lamp could affect up to n positions.

To solve this efficiently, one can use a technique known as prefix sum or cumulative sum. We can increment the brightness at the starting position of a lamp's range and decrement it just after the ending position of the lamp's range. By doing this for all lamps, we would have an array where the value at each position indicates how much the total brightness changes at that point on the street. Accumulating these changes from the beginning to the end will give us the actual brightness at each position.

Once we have the brightness at each position, it's straightforward to compare it with the requirement at each position and count the positions where the brightness is adequate.

The given solution code performs exactly this:

1. Initializes an array d to store the differences in brightness.
2. Loops through each lamp and updates the d array to include the brightness change at the start and just after the end of each lamp's range.
3. Utilizes Python's accumulate function to calculate the prefix sum of brightness changes, giving the actual brightness at each position.
4. Counts the number of positions where the actual brightness meets or exceeds the required brightness.

The solution is elegant and efficient, taking advantage of subtle changes in brightness rather than brute-force updating the entire range for every lamp, resulting in a more time-efficient approach.

## Solution Approach

The solution uses a difference array to efficiently manage brightness updates across the range that each lamp covers. Here's a walkthrough of the steps involved in the solution:

1. Initialize a difference array d, which is an auxiliary array that allows us to apply updates to the original array in constant time. In our case, the difference array is oversized to avoid boundary checks later.
2. Loop through each lamp's properties given in the lights array. For every lamp with parameters [position\_i, range\_i], calculate the starting index i and ending index j for its illumination. This is done by ensuring that the range does not exceed the boundaries of the street, i.e., between 0 and n - 1.
3. Update the difference array d by incrementing the value at index i by 1 and decrementing the value at index j + 1 by 1. The increment at index i signifies that from this point onwards, the brightness level has increased due to the lamp, while the decrement at j + 1 marks that beyond this point, the influence of the current lamp does not extend, effectively lowering the brightness level.
4. Now that we have set up the difference array, we use the accumulate function in Python to compute the prefix sum, which essentially applies all the increments and decrements we've added in the difference array and yields the actual brightness level at each index. The accumulate function will perform this operation in O(n) time across the difference array, thereby generating the brightness levels for the entire street.
5. Finally, we loop over the prefix sum result alongside the requirement array, comparing values at each position. For every position, we check if the brightness level at that index (obtained from the prefix sum) is greater than or equal to the required brightness level. We count all such occurrences where the condition is satisfied.

The overall time complexity is O(n + m), where n is the length of the street and m is the number of lamps. This is because we process each lamp to update the difference array once and then scan through the array of length n to compute the prefix sum and compare it against requirements.

In summary, the code efficiently uses a difference array technique and the power of prefix sum to obtain brightness levels across the street, followed by a simple tally against the specified brightness requirements.

### Example Walkthrough

Let's consider an example to illustrate the solution approach.

Suppose we are given:

* A street of length n = 5, which is from position 0 to 4.
* An array lights = [[1, 2], [3, 1]], indicating there are two street lamps. The first lamp is at position 1 with a range of 2 and can illuminate from position 0 to 3. The second lamp is at position 3 with a range of 1 and can illuminate from position 2 to 4.
* An array requirement = [1, 2, 1, 1, 2] specifying the minimum brightness required at each position on the street.

We want to find the count of positions where the brightness is at least the specified requirement.

Let's proceed with the steps:

1. Initialize a difference array d of size n + 1 to handle brightness changes, thus d = [0, 0, 0, 0, 0, 0].
2. Process the first lamp [1, 2]. The starting index i is max(0, 1 - 2) = 0 and the ending index j is min(4, 1 + 2) = 3. Update d by incrementing d[i] and decrementing d[j + 1], leading to d = [1, 0, 0, 0, -1, 0].
3. Process the second lamp [3, 1]. The starting index i is max(0, 3 - 1) = 2 and the ending index j is min(4, 3 + 1) = 4. Update d by incrementing d[i] and decrementing d[j + 1], which after the update gives d = [1, 0, 1, 0, -1, -1].
4. Compute the prefix sum with the accumulate function to determine the actual brightness at each position. After using accumulate, the brightness levels array becomes [1, 1, 2, 2, 1, 0].
5. Compare these brightness levels with requirement. At positions 0, 1, 2, and 3, the brightness equals or exceeds the requirement, which gives us 4 positions meeting the requirement. Position 4 has a brightness of 1, which does not meet the requirement of 2.

Hence, the final answer is 4 positions with adequate brightness.

class Solution {

// Method to calculate the number of positions that meet the required brightness

public int meetRequirement(int n, int[][] lights, int[] requirement) {

// Array 'brightnessChanges' holds the net change in brightness at each position

int[] brightnessChanges = new int[100010];

// Loop through the array of lights to populate the 'brightnessChanges' array

for (int[] light : lights) {

// Calculate the effective range of light for each light bulb

// Make sure the range does not go below 0 or above n-1

int start = Math.max(0, light[0] - light[1]);

int end = Math.min(n - 1, light[0] + light[1]);

// Increment brightness at the start position

++brightnessChanges[start];

// Decrement brightness just after the end position

--brightnessChanges[end + 1];

}

int currentBrightness = 0; // Holds the cumulative brightness at each position

int positionsMeetingReq = 0; // Number of positions meeting the requirement

// Iterate over positions from 0 to n-1

for (int i = 0; i < n; ++i) {

// Calculate the current brightness by adding the net brightness change at position i

currentBrightness += brightnessChanges[i];

// If current brightness meets or exceeds the requirement at position i, increase count

if (currentBrightness >= requirement[i]) {

++positionsMeetingReq;

}

}

// Return the total number of positions meeting the brightness requirement

return positionsMeetingReq;

}

}