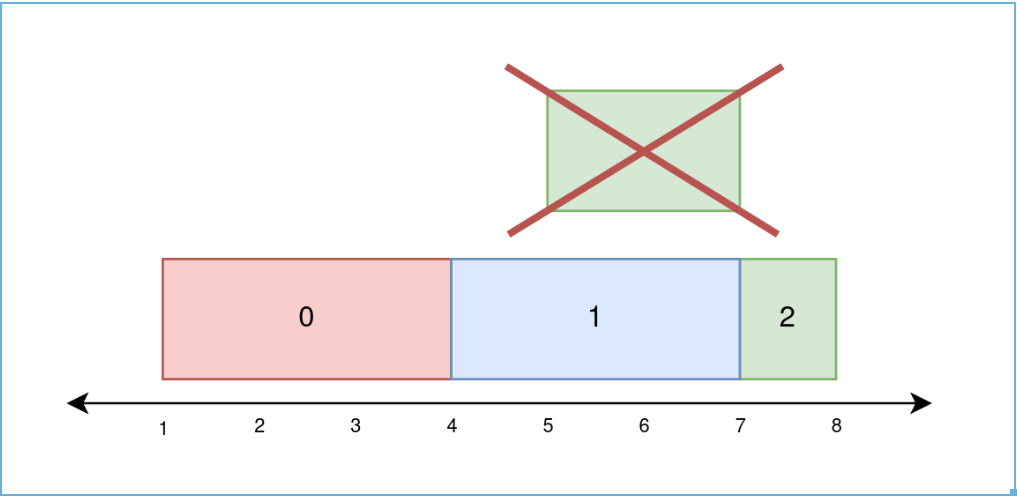
<https://algo.monster/liteproblems/2158>

There is a long and thin painting that can be represented by a number line. You are given a **0-indexed** 2D integer array paint of length n, where paint[i] = [starti, endi]. This means that on the ith day you need to paint the area **between** starti and endi.

Painting the same area multiple times will create an uneven painting so you only want to paint each area of the painting at most **once**.

Return *an integer array* worklog *of length* n *, where* worklog[i] *is the amount of* ***new*** *area that you painted on the* ith*day.*

**Example 1:**



Input: paint = [[1,4],[4,7],[5,8]]

Output: [3,3,1]

Explanation:

On day 0, paint everything between 1 and 4.

The amount of new area painted on day 0 is 4 - 1 = 3.

On day 1, paint everything between 4 and 7.

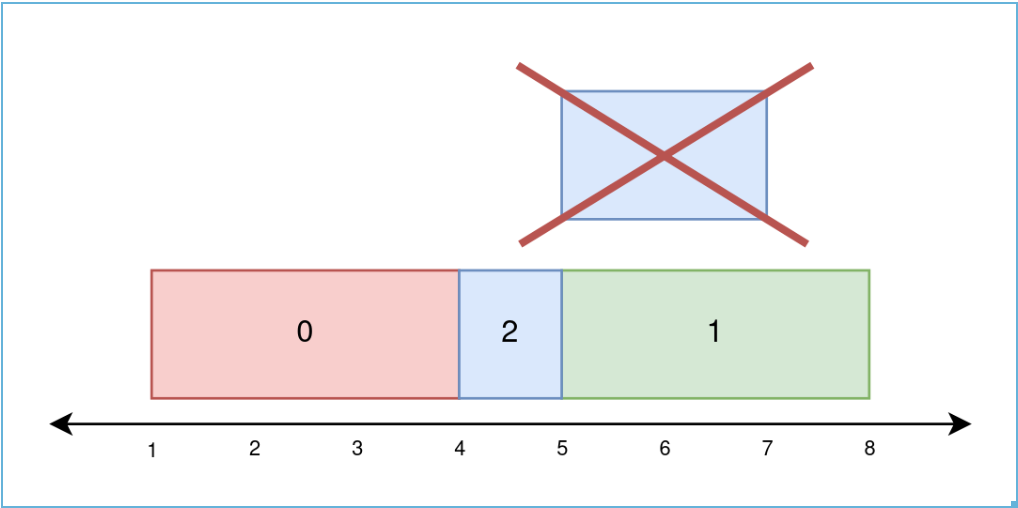
The amount of new area painted on day 1 is 7 - 4 = 3.

On day 2, paint everything between 7 and 8.

Everything between 5 and 7 was already painted on day 1.

The amount of new area painted on day 2 is 8 - 7 = 1.

**Example 2:**



Input: paint = [[1,4],[5,8],[4,7]]

Output: [3,3,1]

Explanation:

On day 0, paint everything between 1 and 4.

The amount of new area painted on day 0 is 4 - 1 = 3.

On day 1, paint everything between 5 and 8.

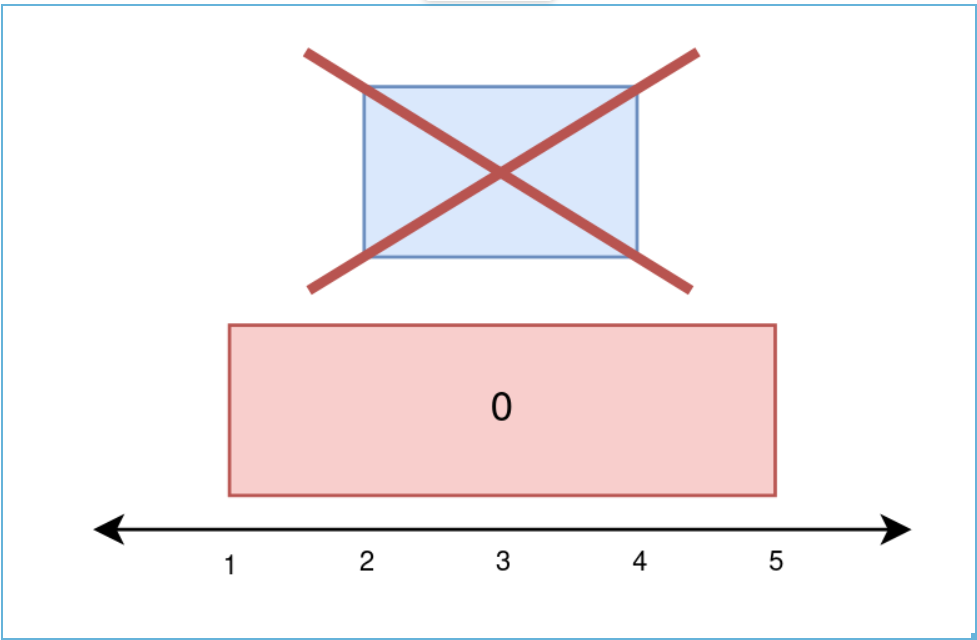
The amount of new area painted on day 1 is 8 - 5 = 3.

On day 2, paint everything between 4 and 5.

Everything between 5 and 7 was already painted on day 1.

The amount of new area painted on day 2 is 5 - 4 = 1.

**Example 3:**



Input: paint = [[1,5],[2,4]]

Output: [4,0]

Explanation:

On day 0, paint everything between 1 and 5.

The amount of new area painted on day 0 is 5 - 1 = 4.

On day 1, paint nothing because everything between 2 and 4 was already painted on day 0.

The amount of new area painted on day 1 is 0.

**Constraints:**

* 1 <= paint.length <= 10^5
* paint[i].length == 2
* 0 <= starti < endi <= 5 \* 10^4

**Attempt 1: 2023-12-09**

**One detail very important, before paint ith day, we have to check coverage for all paint 0th day to (i - 1)th day**

**Solution 1: Brute Force (10 min)**

**Style 1:**

class Solution {

public int[] amountPainted(int[][] paint) {

boolean[] slots = new boolean[100000];

int n = paint.length;

int[] result = new int[n];

// e.g day 0 -> {1,4}, day 1 -> {5,8}, day 2 -> {4,7}

// P0 P0 P0 P1 P1 P1 P2

// slot 0 1 2 3 4 5 6 7 8

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

// Each paint[day] represent by slot[start, end)

// 'end' of that day exclusive

// day = 0 is a little special, no previous day coverage on

// slot array need to check, all later days(if any) need to

// check previous day coverage on slot array, if any slot

// mark as 'true', we cannot add into count

if(day == 0) {

for(int i = paint[day][0]; i < paint[day][1]; i++) {

result[day]++;

slots[i] = true;

}

} else {

for(int i = paint[day][0]; i < paint[day][1]; i++) {

if(!slots[i]) {

result[day]++;

}

slots[i] = true;

}

}

}

return result;

}

}

Let N be the number of days (up to 100000) and let M be the largest number that appears in the input (up to 50000)

Time Complexity: O(M\*N)

Space Compleixty: O(M\*N)

**Style 2:**

class Solution {

public int[] amountPainted(int[][] paint) {

boolean[] slots = new boolean[100000];

int n = paint.length;

int[] result = new int[n];

// e.g day 0 -> {1,4}, day 1 -> {5,8}, day 2 -> {4,7}

// P0 P0 P0 P1 P1 P1 P2

// slot 0 1 2 3 4 5 6 7 8

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

// Each paint[day] represent by slot[start, end)

// 'end' of that day exclusive

// After improve, actually day = 0 has no difference than any

// other day, because initially all slots are 'false', we can

// merge two cases into single check on if 'slots[i]' is true,

// only when 'false' we can add into paint count that day

for(int i = paint[day][0]; i < paint[day][1]; i++) {

if(!slots[i]) {

result[day]++;

}

slots[i] = true;

}

}

return result;

}

}

Let N be the number of days (up to 100000) and let M be the largest number that appears in the input (up to 50000)

Time Complexity: O(M\*N)

Space Compleixty: O(M\*N)

**Refer to**

<https://algo.monster/liteproblems/2158>

Let's split the number line into blocks such that for the ith block covers the interval [i, i+1]. Create a boolean array to store whether each block has been painted.

On day i, we are tasked with painting blocks start i to end i - 1. We can check each of these blocks, painting the unpainted ones (we also keep count of how many blocks we paint because that's what the question asks for). In the worst case, we have to check every block on every day. Let n be the number of days (up to 100000) and let m be the largest number that appears in the input (up to 50000). The time complexity is O(*nm*). This is not fast enough.

**Solution 2: Sweep Line + TreeSet + Sorting (180 min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

class Node {

// 'start' day or 'end' day coming from one paint event

// in paint array

int day;

// Record Node create from which paint event in paint array,

// used for mapping to corresponding index of result array

int eventIndex;

// 'start' assign 1, 'end' assign -1

int startOrEnd;

public Node(int day, int eventIndex, int startOrEnd) {

this.day = day;

this.eventIndex = eventIndex;

this.startOrEnd = startOrEnd;

}

}

public int[] amountPainted(int[][] paint) {

int n = paint.length;

int[] result = new int[n];

int minDay = 100001;

int maxDay = 0;

List<Node> list = new ArrayList<>();

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

int start = paint[i][0];

int end = paint[i][1];

minDay = Math.min(minDay, start);

maxDay = Math.max(maxDay, end);

list.add(new Node(start, i, 1));

list.add(new Node(end, i, -1));

}

Collections.sort(list, (a, b) -> a.day - b.day);

// 用TreeSet做Sweep Line中的delta变动数组非常巧妙

TreeSet<Integer> delta = new TreeSet<>();

int i = 0;

for(int day = minDay; day <= maxDay; day++) {

while(i < list.size() && list.get(i).day == day) {

if(list.get(i).startOrEnd == 1) {

delta.add(list.get(i).eventIndex);

} else {

delta.remove(list.get(i).eventIndex);

}

i++;

}

if(!delta.isEmpty()) {

result[delta.first()]++;

}

}

return result;

}

}

Time Complexity: O(max(paint[i][0])+sort)

Space Complexity: O(n)

**Refer to**

<https://walkccc.me/LeetCode/problems/2158/>

**这里对treeset的使用非常关键，生成对象runningIndices利用了它的自动排序特性保证了无论存入多少个event的坐标，永远是最先存入的还没有结束的event的坐标被调用，使用treeset.first()方法获取最先存入的还没有结束的event的坐标**

class Solution {

enum Type { kEntering, kLeaving }

class Event {

public int day;

public int index;

public Type type;

public Event(int day, int index, Type type) {

this.day = day;

this.index = index;

this.type = type;

}

}

public int[] amountPainted(int[][] paint) {

final int n = paint.length;

final int minDay = Arrays.stream(paint).mapToInt(x -> x[0]).min().getAsInt();

final int maxDay = Arrays.stream(paint).mapToInt(x -> x[1]).max().getAsInt();

int[] ans = new int[n];

// Stores the indices of paints that are available now.

TreeSet<Integer> runningIndices = new TreeSet<>();

List<Event> events = new ArrayList<>();

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

final int start = paint[i][0];

final int end = paint[i][1];

events.add(new Event(start, i, Type.kEntering)); // 1 := entering

events.add(new Event(end, i, Type.kLeaving)); // -1 := leaving

}

Collections.sort(events, (a, b) -> a.day - b.day);

int i = 0; // events' index

for (int day = minDay; day < maxDay; ++day) {

while (i < events.size() && events.get(i).day == day) {

if (events.get(i).type == Type.kEntering)

runningIndices.add(events.get(i).index);

else

runningIndices.remove(events.get(i).index);

++i;

}

if (!runningIndices.isEmpty())

++ans[runningIndices.first()];

}

return ans;

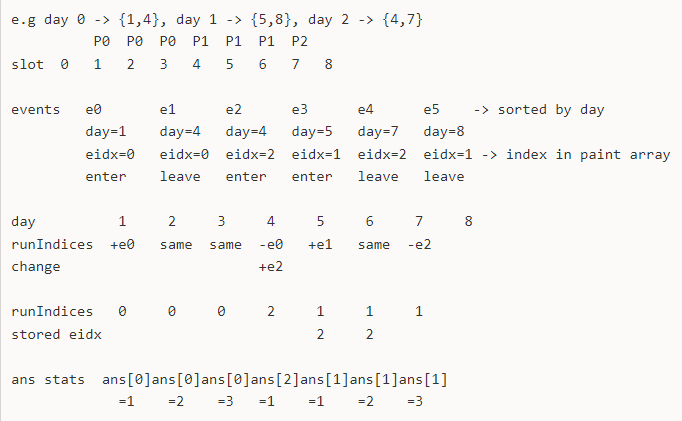
}

}

**Time Complexity:** *O*(max(paint[i][0])+sort)

**Space Complexity:** *O*(*n*)

**Step by Step**



e.g day 0 -> {1,4}, day 1 -> {5,8}, day 2 -> {4,7}

P0 P0 P0 P1 P1 P1 P2

slot 0 1 2 3 4 5 6 7 8

events e0 e1 e2 e3 e4 e5 -> sorted by day

day=1 day=4 day=4 day=5 day=7 day=8

eidx=0 eidx=0 eidx=2 eidx=1 eidx=2 eidx=1 -> index in paint array

enter leave enter enter leave leave

day 1 2 3 4 5 6 7 8

runIndices +e0 same same -e0 +e1 same -e2

change +e2

runIndices 0 0 0 2 1 1 1

stored eidx 2 2

ans stats ans[0]ans[0]ans[0]ans[2]ans[1]ans[1]ans[1]

=1 =2 =3 =1 =1 =2 =3

day 1

ans runIndices.first()=0

ans[0]++ -> 1

day 2

ans runIndices.first()=0

ans[0]++ -> 2

day 3

ans runIndices.first()=0

ans[0]++ -> 3

day 4

ans runIndices.first()=2

ans[2]++ -> 1

day 5

ans runIndices.first()=1

ans[1]++ -> 1

day 6

ans runIndices.first()=1

ans[1]++ -> 2

day 7

ans runIndices.first()=1

ans[1]++ -> 3