<https://leetcode.com/problems/maximum-population-year/description/>

You are given a 2D integer array logs where each logs[i] = [birthi, deathi] indicates the birth and death years of the ith person.

The **population** of some year x is the number of people alive during that year. The ith person is counted in year x's population if x is in the **inclusive** range [birthi, deathi - 1]. Note that the person is **not** counted in the year that they die.

Return *the* ***earliest*** *year with the* ***maximum population***.

**Example 1:**

Input: logs = [[1993,1999],[2000,2010]]

Output: 1993

Explanation: The maximum population is 1, and 1993 is the earliest year with this population.

**Example 2:**

Input: logs = [[1950,1961],[1960,1971],[1970,1981]]

Output: 1960

Explanation:

The maximum population is 2, and it had happened in years 1960 and 1970.

The earlier year between them is 1960.

**Constraints:**

* 1 <= logs.length <= 100
* 1950 <= birthi < deathi <= 2050

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

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

**Same as L253/P5.5.Meeting Rooms II Sweep Line strategy, and a bit different than L2848.Points That Intersect With Cars for 'end' handling, because here we define:**

The **population** of some year x is the number of people alive during that year. The ith person is counted in year x's population if x is in the **inclusive** range [birthi, deathi - 1]. Note that the person is **not** counted in the year that they die.

**In L2848 the range is inclusive till 'end' index itself, mapping to here means [birthi, deathi] not [birthi, deathi - 1], the person is counted in the year that they die.**

class Solution {

public int maximumPopulation(int[][] logs) {

// 2050 - 1950 + 1 = 101

int[] timeline = new int[101];

for(int[] log : logs) {

timeline[log[0] - 1950]++;

// No need L2848 way to handle by log[1] + 1 to include

// the 'end' index value, since definition saying:

// The person is not counted in the year that they die.

timeline[log[1] - 1950]--;

}

int count = 0;

int max = 0;

int year = 0;

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

count += timeline[i];

if(count > max) {

max = count;

year = i;

}

}

return year + 1950;

}

}

Time Complexity: O(N)

Space Compleixty: O(N)

**Refer to**

<https://leetcode.com/problems/maximum-population-year/solutions/1198978/java-o-n-solution-with-explanation-range-addition/>

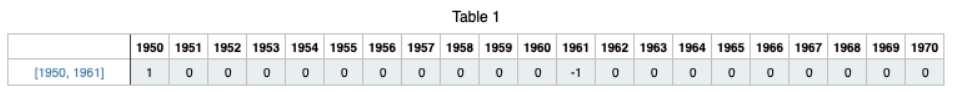
Let's understand with an exampleSuppose we are given the log [1950, 1961]This means birth year is 1950 and death year is 1961Therefore, years from 1950 to 1960 will be incremented by 1.

We can do it by iterating from 1950 to 1960 but that will increase time complexity if we have to do it for every query in logs array.

To do this in O(1), we increment year[1950] by 1 and decrement year[1961] by 1.We can reapeat this for every query in logs array.

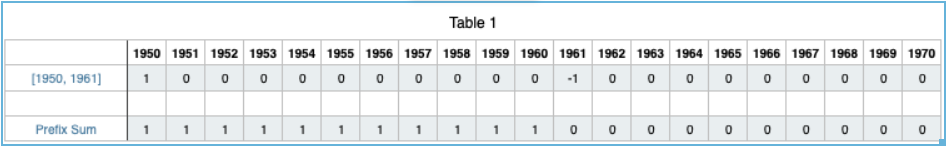
What will this do ?

For the case [1950, 1961], let's look at how the array will look like



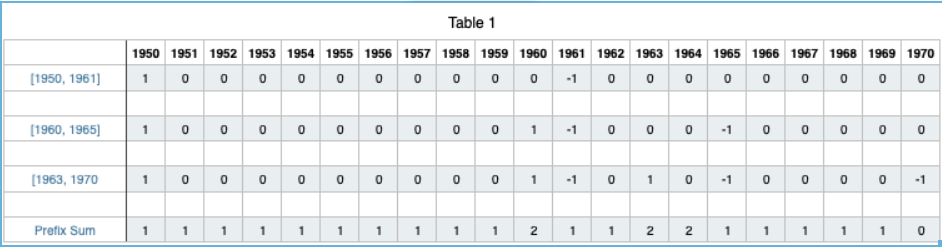
But this is not the desired result ?

To get the answer, After iterating through all the queries, take prefix sum of the array(year)This is how the array will look like



You can see that the Prefix Sum row will give the desired result as we have incremented the values of array from index 1950 to 1960.

Let's try for the test case, logs = [[1950,1961],[1960,1965],[1963,1970]] for a better understanding



Looking at the Prefix Sum, we can clearly see that the maximum value is 2 and its first occurence is at 1960. Hence, 1960 is the answer.

class Solution {

public int maximumPopulation(int[][] logs) {

int[] year = new int[2051];

// O(n) -> n is log.length

for(int[] log : logs){

year[log[0]] += 1;

year[log[1]] -= 1;

}

int maxNum = year[1950], maxYear = 1950;

// O(100) -> 2050 - 1950 = 100

for(int i = 1951; i < year.length; i++){

year[i] += year[i - 1]; // Generating Prefix Sum

if(year[i] > maxNum){

maxNum = year[i];

maxYear = i;

}

}

return maxYear;

}

}