2251. Number of Flowers in Full Bloom

Binary Search

Problem Description

Array

Hash Table

In this problem, we are dealing with a scenario related to flowers blooming. The input includes two arrays:

1. A 2D array flowers, where each sub-array contains two elements indicating the start and end of the full bloom period for a particular flower. The

Ordered Set Prefix Sum

flower blooms inclusively from start_i to end_i. An array persons, where each element represents the time a person arrives to see the flowers.

Sorting

The goal is to determine how many flowers are in full bloom for each person when they a

The goal is to determine how many flowers are in full bloom for each person when they arrive. The output should be an array answer, where answer[i] corresponds to the number of flowers in full bloom at the time the ith person arrives.

1. **Sorting:** We separate the start and end times of the bloom periods into two lists and sort them. The sorted start times help us

Intuition

Hard

determine how many flowers have started blooming at a given point, and the sorted end times indicate how many flowers have finished blooming.

bloom for each person's arrival time. Here's the implementation explained step by step:

count of all flowers that have started blooming up to time p (including p).

comprehends the count of flowers in bloom for each person, as per their arrival times:

return [bisect_right(start, p) - bisect_left(end, p) for p in persons]

problem in a time-efficient manner, taking advantage of the ordered datasets for quick lookups.

Now let's walk through the steps to get the number of flowers in bloom for each person:

○ The difference 1 (started) - 0 (ended) tells us that exactly one flower is in full bloom.

bisect_right([1, 2, 4], 3) results in index 2, as two flowers have bloomed by time 3.

• The difference 2 (started) - 1 (ended) is 1, so one flower is blooming for this person.

Calculate the number of flowers in full bloom for each person's visit

The total number of flowers that have started blooming by person p's visit time

Subtracting the number of flowers that have finished blooming by person p's visit time

• The difference 3 (started) - 2 (ended) is 1, indicating that one flower is in bloom when this person arrives.

bisect_left([3, 5, 7], 3) gives us index 1, as one flower has stopped blooming.

To solve this problem, we can use a two-step strategy involving sorting and binary search:

- have finished blooming.
 Binary Search: When a person arrives, we want to count the flowers that have begun blooming but haven't finished. We use the binary search algorithm to find:
- The index of the *first* end time that is *strictly greater* than the arrival time of the person, which indicates how many flowers have finished blooming. We get this number using bisect_left on the sorted end times.
 The index of the *first* start time that is *greater than or equal* to the arrival time, which tells us how many flowers have started to bloom. We use bisect_right for this on the sorted start times.
- By subtracting the number of flowers that have finished blooming from those that have started, we get the count of flowers in full bloom when a person arrives. We repeat this process for each person and compile the results into the final answer array.
- Solution Approach

 The solution approach uses a combination of <u>sorting</u> and <u>binary search</u> to efficiently determine how many flowers are in full

Sort Starting and Ending Times: First, we extract all the start times and end times from the flowers array into separate lists

start = sorted(a for a, _ in flowers)

and sort them:

end = sorted(b for _, b in flowers)

Sorting these lists allows us to use binary search later on. The start list will be used to determine how many flowers have

started blooming by a certain time, and the end list will help determine how many flowers have ended their bloom.

the count of flowers in bloom at that particular time. For each p:
bisect_right(start, p) - bisect_left(end, p)

Binary Search for Bloom Count: The next step is to iterate over each person's arrival time p in the persons list and find out

bisect_left(end, p) finds the index in the sorted end list where p could be inserted to maintain the order. This index signifies the count of

```
    bisect_right(start, p) - bisect_left(end, p)
    bisect_right(start, p) finds the index in the sorted start list where p would be inserted to maintain the order. This index represents the
```

flowers that have not finished blooming by time p.

number of flowers in bloom at the arrival time of p.

3. **Compile Results**: The above operation is repeated for each person's arrival time, and the results are compiled into the answer list. This list

By subtracting the numbers obtained from bisect_left on the end list from bisect_right on the start list, we obtain the total

In the end, the answer list is returned, which provides the solution, i.e., the number of flowers in full bloom at the time of each person's arrival.

The algorithms and data structures used here, like sorting and binary search (bisect module in Python), enable us to solve the

Imagine we have an array of flowers where the blooms are represented as flowers = [[1,3], [2,5], [4,7]] and an array of persons with arrival times as persons = [1, 3, 5]. We want to find out how many flowers are in full bloom each person sees when they arrive.

First, we need to process the flowers' bloom times. We sort the start times [1, 2, 4] and the end times [3, 5, 7] of the

Person at time 1:
 Using bisect_right for the sorted start times: bisect_right([1, 2, 4], 1) gives us index 1, indicating one flower has started blooming.

Using bisect_left for the end times: bisect_left([3, 5, 7], 1) gives us index 0, indicating no flowers have finished blooming.

2. Person at time 3:

of their arrival times.

Python

Solution Implementation

bloom_counts = [

from bisect import bisect_right, bisect_left

end_times = sorted(end for _, end in flowers)

public int[] fullBloomFlowers(int[][] flowers, int[] people) {

int[] bloomStart = new int[flowerCount];

int[] bloomEnd = new int[flowerCount];

int flowerCount = flowers.length; // Number of flowers

// Extract the start and end bloom times for each flower into separate arrays

vector<int> fullBloomFlowers(vector<vector<int>>& flowers, vector<int>& people) {

// Separate vectors to hold the start and end times for each flower

// Loop over all flowers to populate the start and end vectors

// Sort the start and end vectors to prepare for binary search

bloomCount.push_back(flowersStarted - flowersEnded);

function fullBloomFlowers(flowers: number[][], people: number[]): number[] {

// Arrays to store the start and end times of each flower's bloom.

const bloomStarts = new Array(flowerCount).fill(0);

// Vector to hold the number of flowers in full bloom for each person

// Loop through each person to determine how many flowers are in full bloom

// This gives us the number of flowers that have started blooming

// This gives us the number of flowers that have already ceased blooming

auto flowersEnded = lower_bound(ends.begin(), ends.end(), person) - ends.begin();

// Find the position of the first flower that starts after the person's time (exclusive)

auto flowersStarted = upper_bound(starts.begin(), starts.end(), person) - starts.begin();

// Subtract flowersEnded from flowersStarted to get the number of flowers in full bloom

// Find the position of the first flower that ends at or before the person's time (inclusive)

bisect_right(start_times, p) -

bisect_left(end_times, p)

for p in persons

return bloom_counts

blooming periods.

Example Walkthrough

Person at time 5:

• bisect_right([1, 2, 4], 5) gives an index of 3 - all three flowers have started blooming by time 5.

• bisect_left([3, 5, 7], 5) yields index 2, as two flowers have finished blooming strictly before time 5.

Thus, for the persons arriving at times 1, 3, and 5, the function will return [1, 1, 1] as the number of flowers in full bloom at each

class Solution:
 def fullBloomFlowers(self, flowers: List[List[int]], persons: List[int]) -> List[int]:
 # Sort the start times and end times of the flowers' blooming periods
 start_times = sorted(start for start, _ in flowers)

sol = Solution() # print(sol.fullBloomFlowers([[1, 10], [3, 3]], [4, 5])) Java

import java.util.Arrays;

public class Solution {

Example usage:

```
for (int i = 0; i < flowerCount; ++i) {</pre>
            bloomStart[i] = flowers[i][0];
            bloomEnd[i] = flowers[i][1];
       // Sort the start and end bloom times arrays
       Arrays.sort(bloomStart);
       Arrays.sort(bloomEnd);
        int peopleCount = people.length; // Number of people
        int[] answer = new int[peopleCount]; // Array to store the answers
        // For each person, calculate the number of flowers in full bloom
        for (int i = 0; i < peopleCount; ++i) {</pre>
           // Number of flowers that have started blooming minus
           // the number of flowers that have already ended blooming
            answer[i] = findInsertionPoint(bloomStart, people[i] + 1) - findInsertionPoint(bloomEnd, people[i]);
        return answer; // Return the array containing the number of flowers in full bloom for each person
    private int findInsertionPoint(int[] times, int value) {
        int left = 0; // Start of the search range
        int right = times.length; // End of the search range
       // Binary search to find the insertion point of 'value'
       while (left < right) {</pre>
            int mid = (left + right) / 2; // Midpoint of the current search range
            if (times[mid] >= value) {
                right = mid; // Adjust the search range to the left half
            } else {
                left = mid + 1; // Adjust the search range to the right half
        return left; // The insertion point is where we would add 'value' to keep the array sorted
C++
#include <vector>
#include <algorithm>
using namespace std;
```

```
// Return the counts of flowers in full bloom for each person
return bloomCount;
}
};

TypeScript
```

class Solution {

// Number of flower intervals

for (auto& flower: flowers) {

starts.push_back(flower[0]);

ends.push_back(flower[1]);

sort(starts.begin(), starts.end());

sort(ends.begin(), ends.end());

for (auto& person : people) {

const flowerCount = flowers.length;

vector<int> bloomCount;

int n = flowers.size();

vector<int> starts;

vector<int> ends;

public:

```
const bloomEnds = new Array(flowerCount).fill(0);
      // Split the flowers' bloom times into start and end times.
      for (let i = 0; i < flowerCount; ++i) {</pre>
          bloomStarts[i] = flowers[i][0];
          bloomEnds[i] = flowers[i][1];
      // Sort the start and end times.
      bloomStarts.sort((a, b) => a - b);
      bloomEnds.sort((a, b) => a - b);
      // Array to store the result for each person.
      const results: number[] = [];
      for (const person of people) {
          // Find the number of flowers blooming at the time person visits.
          const flowersBloomingStart = search(bloomStarts, person + 1); // Start of blooms after person
          const flowersBloomingEnd = search(bloomEnds, person); // End of blooms by the time person visits
          results.push(flowersBloomingStart - flowersBloomingEnd); // Number of flowers currently in bloom
      return results;
  // Binary search helper function to find the index at which a flower's start or end time is greater than or equal to x.
  function search(nums: number[], x: number): number {
      let left = 0;
      let right = nums.length;
      while (left < right) {</pre>
          const mid = left + ((right - left) >> 1); // Prevents potential overflow
          if (nums[mid] >= x) {
              right = mid; // Look in the left half
          } else {
              left = mid + 1; // Look in the right half
      return left; // Left is the index where nums[left] is >= x
from bisect import bisect_right, bisect_left
class Solution:
   def fullBloomFlowers(self, flowers: List[List[int]], persons: List[int]) -> List[int]:
       # Sort the start times and end times of the flowers' blooming periods
        start_times = sorted(start for start, _ in flowers)
        end_times = sorted(end for _, end in flowers)
       # Calculate the number of flowers in full bloom for each person's visit
       bloom_counts = |
           # The total number of flowers that have started blooming by person p's visit time
            bisect_right(start_times, p) -
            # Subtracting the number of flowers that have finished blooming by person p's visit time
            bisect_left(end_times, p)
            for p in persons
       return bloom_counts
# Example usage:
 sol = Solution()
# print(sol.fullBloomFlowers([[1, 10], [3, 3]], [4, 5]))
Time and Space Complexity
```

Sorting the start times of the flowers: sorted(a for a, _ in flowers) Sorting the end times of the flowers: sorted(b for _, b in flowers)

The given code consists of three main parts:

- 3. Iterating through each person and using binary search to find the count of bloomed flowers: [bisect_right(start, p) bisect_left(end, p) for p in persons]
- Let's consider n as the number of flowers and m as the number of persons. Here's a breakdown of the time complexity:

 Sorting the start and end times: Sorting takes $0(n \log n)$ time for both the start and end lists. Hence the combined sorting time is $2 * 0(n \log n)$

Therefore, the overall space complexity of the code is O(n).

- Binary search for each person: For each person, bisect_right and bisect_left are performed once. These operations have a time complexity of O(log n). Since these operations are performed for m persons, the total time for this part is O(m log n).
- Space Complexity

Adding these up, the overall time complexity of the code is $0(n \log n + m \log n)$.

The space complexity comes from the additional lists used to store start and end times:

• Start and end lists: Two lists are created to store start and end times, each of size n. Hence, the space taken by these lists is 2 * 0(n).

n).

Time Complexity