CP Internal Exam

You are given a 2D integer array meetings where meetings[i] = [$start_i$, end_i] means that a meeting will be held during the **half-closed** time interval [$start_i$, end_i). All the values of $start_i$ are **unique**.

Meetings are allocated to rooms in the following manner:

1. Each meeting will take place in the unused room with the **lowest** number.
2. If there are no available rooms, the meeting will be delayed until a room becomes free. The delayed meeting should have the **same** duration as the original meeting.
3. When a room becomes unused, meetings that have an earlier original **start** time should be given the room.

Return *the **number** of the room that held the most meetings.* If there are multiple rooms, return *the room with the **lowest** number.*

A **half-closed interval** [a, b) is the interval between a and b **including** a and **not including** b.

Example 1:
Input: n = 2, meetings = [[0,10],[1,5],[2,7],[3,4]]
Output: 0
Explanation:

- At time 0, both rooms are not being used. The first meeting starts in room 0.
- At time 1, only room 1 is not being used. The second meeting starts in room 1.
- At time 2, both rooms are being used. The third meeting is delayed.
- At time 3, both rooms are being used. The fourth meeting is delayed.
- At time 5, the meeting in room 1 finishes. The third meeting starts in room 1 for the time period [5,10).
- At time 10, the meetings in both rooms finish. The fourth meeting starts in room 0 for the time period [10,11).
- Both rooms 0 and 1 held 2 meetings, so we return 0.

Example 2:
Input: n = 3, meetings = [[1,20],[2,10],[3,5],[4,9],[6,8]]
Output: 1
Explanation:

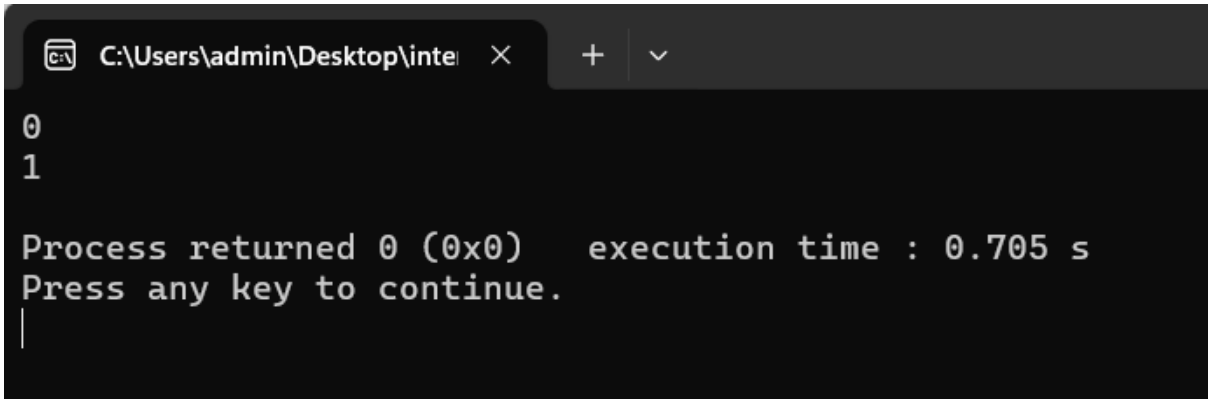
- At time 1, all three rooms are not being used. The first meeting starts in room 0.
- At time 2, rooms 1 and 2 are not being used. The second meeting starts in room 1.
- At time 3, only room 2 is not being used. The third meeting starts in room 2.
- At time 4, all three rooms are being used. The fourth meeting is delayed.

Code(C++)

```
#include <bits/stdc++.h>

using namespace std;
```

```
class Solution {
public:
    int mostBooked(int n, vector<vector<int>>& meetings) {
        sort(meetings.begin(), meetings.end()); // sort by start time

        vector<long long> endTime(n, 0); // when each room will be free
        vector<int> count(n, 0);        // number of meetings in each room

        for (auto &m : meetings) {
            long long start = m[0], end = m[1];
            long long duration = end - start;

            // find the first free room at "start"
            int room = -1;
            for (int i = 0; i < n; i++) {
                if (endTime[i] <= start) {
                    room = i;
                    break;
                }
            }

            if (room != -1) {
                // found a free room
                endTime[room] = end;
                count[room]++;
            } else {
                // all rooms busy → pick the one that gets free first
                long long earliest = LLONG_MAX;
                for (int i = 0; i < n; i++) {
                    if (endTime[i] < earliest) {
                        earliest = endTime[i];
                        room = i;
```

```
            }
          }
          // delay this meeting
          endTime[room] = earliest + duration;
          count[room]++;
        }
      }

      // find the room with max meetings
      int ans = 0;
      for (int i = 1; i < n; i++) {
        if (count[i] > count[ans]) ans = i;
      }
      return ans;
    }
};

int main() {
    Solution s;
    vector<vector<int>> m1 = {{0,10},{1,5},{2,7},{3,4}};
    cout << s.mostBooked(2, m1) << endl; // 0

    vector<vector<int>> m2 = {{1,20},{2,10},{3,5},{4,9},{6,8}};
    cout << s.mostBooked(3, m2) << endl; // 1
}
```



Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Examples:

Example 1:

- Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
- Output: 6
- Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:

- Input: height = [4,2,0,3,2,5]
- Output: 9

Constraints:

- n == height.length
- $1 \leq n \leq 2 \times 10^4$
- $0 \leq height[i] \leq 10^5$

Code-

```
#include <bits/stdc++.h>

using namespace std;


int trap(vector<int>& h) {

    int n = h.size();

    int l = 0, r = n - 1;

    int leftMax = 0, rightMax = 0;

    int water = 0;
```

```
    while (l < r) {
        if (h[l] < h[r]) {
            leftMax = max(leftMax, h[l]);
            water += leftMax - h[l];
            l++;
        } else {
            rightMax = max(rightMax, h[r]);
            water += rightMax - h[r];
            r--;
        }
    }
    return water;
}

int main() {
    int n;
    cin >> n;
    vector<int> h(n);
    for (int i = 0; i < n; i++) cin >> h[i];

    cout << trap(h) << endl;
    return 0;
}
```

Output-

```
C:\Users\admin\Desktop\inte

6
4
2
0
3
2
5
9

Process returned 0 (0x0)   execution time : 22.327 s
Press any key to continue.
```