INTERNAL PRACTICAL EXAM(CP)

QUESION-1:

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:

```
import heapq
def mostBooked(n, meetings):
    meetings.sort()
    available = [i for i in range(n)]
    heapq.heapify(available)

    ongoing = []
    count = [0] * n
    for start, end in meetings:
        while ongoing and ongoing[0][0] <= start:
            finish_time, room = heapq.heappop(ongoing)
            heapq.heappush(available, room)
        if available:
            room = heapq.heappop(available)
            heapq.heappush(ongoing, (end, room))
            count[room] += 1
        else:
            finish_time, room = heapq.heappop(ongoing)
            new_end = finish_time + (end - start)  # same duration
            heapq.heappush(ongoing, (new_end, room))
            count[room] += 1

    max_meetings = max(count)
    for i in range(n):
```

```
        if count[i] == max_meetings:
            return i
n = int(input("Enter number of rooms: "))
m = int(input("Enter number of meetings: "))

meetings = []
for i in range(m):
    start, end = map(int, input(f"Enter meeting {i+1} start and end time: ").split())
    meetings.append([start, end])
print("Room with most meetings:", mostBooked(n, meetings))
```

OUTPUT:

```
PS D:\SEM-5\CP> & C:\Python313\python.exe d:/SEM-5/CP/internal_water.py
Enter number of rooms: 2
Enter number of meetings: 4
Enter meeting 1 start and end time: 0 2
Enter meeting 2 start and end time: 1 5
Enter meeting 3 start and end time: 3 6
Enter meeting 4 start and end time: 5 8
Room with most meetings: 0
```

```
PS D:\SEM-5\CP> & C:\Python313\python.exe d:/SEM-5/CP/internal_water.py
Enter number of rooms: 3
Enter number of meetings: 5
Enter meeting 1 start and end time: 1 20
Enter meeting 2 start and end time: 2 10
Enter meeting 3 start and end time: 3 5
Enter meeting 4 start and end time: 4 9
Enter meeting 5 start and end time: 6 8
Room with most meetings: 1
```

QUESION-2

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:

```
def trap(height):
    left, right = 0, len(height) - 1
    left_max, right_max = 0, 0
    water_trapped = 0

    while left < right:
        if height[left] < height[right]:
            if height[left] >= left_max:
                left_max = height[left]
            else:
                water_trapped += left_max - height[left]
            left += 1
        else:
            if height[right] >= right_max:
                right_max = height[right]
            else:
                water_trapped += right_max - height[right]
            right -= 1
    return water_trapped

arr = list(map(int,input("Enter heights separated by space: ").split()))
print("Total water trapped:",trap(arr))
```

OUTPUT:

```
PS D:\SEM-5\CP> & C:\Python313\python.exe d:/SEM-5/CP/internal_water.py
Enter heights separated by space: 1 0 2 1 0 4 1 3 0 2 1 0 2 3
Total water trapped: 16
```

```
PS D:\SEM-5\CP> & C:\Python313\python.exe d:/SEM-5/CP/internal_water.py
Enter heights separated by space: 4 2 0 3 2 5
Total water trapped: 9
PS D:\SEM-5\CP>
```