Amazon Interviewer:
Hi Can, welcome back! We'll do a collaborative DSA round in an Amazon ops setting. Please think aloud, summarize the problem in your structured format, ask clarifying questions, then propose and implement. Ready?

Candidate (Excellent):
Ready. I'll summarize right after you present it.

Interviewer — Problem Statement (Amazon context)

At an Amazon sort center, pickers book **packing stations** for time intervals. Each booking is an interval **`[start, end)`** (half-open: starts at `start`, frees exactly at `end`). A single station cannot run **overlapping** bookings.

Given:

- `intervals` — a list of bookings, **already sorted by start ascending**.

Task: Return the **minimum number of stations** needed to run **all** bookings without overlap.

Examples

1. `intervals = [[0,30],[5,10],[15,20]]` → **2**
   Reason: `[0,30]` overlaps both others; `[5,10]` and `[15,20]` do **not** overlap each other (end is exclusive).

2. `intervals = [[7,10],[10,12]]` → **1**
   Reason: half-open means reuse at the boundary `10`.

Constraints

- `1 ≤ n ≤ 2 * 10^5`
- `0 ≤ start < end ≤ 10^9`
- Intervals are **already sorted by start**

Candidate — Problem Summary (explicit & structured)

Problem Title: Minimum Stations for Overlapping Bookings
Inputs:

- `intervals`: sorted list of `[start, end)` (half-open) bookings
  Must satisfy:
- Same station cannot host overlapping bookings
- End is **exclusive** → if one ends at `t` and another starts at `t`, reuse is allowed
  Output:
- Smallest integer = stations required so all bookings can run
  Constraints & Implications:
- Up to `2e5` intervals, already sorted → solution should be **O(n log n)** or better; heap-based scan is ideal
  Core idea (pattern): Greedy + Min-Heap (Pattern #13) tracking **peak** concurrent usage; free **all** finished stations before placing the next booking

Is this accurate?

Interviewer:
Yes—that's precise. Please proceed.

Candidate — Clarifying Questions

1. Are intervals guaranteed valid (`start < end`)? → **Yes.**
2. If `intervals` is empty, return `0`? → **Yes.**
3. Confirm: input is **already sorted by start**, so I should **not** sort again? → **Correct.**

Candidate — Approach Exploration

Brute force (reject): Checking each booking against all active bookings can hit **O(n²)** in dense cases.

Chosen Pattern: Greedy + Min-Heap of end times (Top-K / Heap)

- Walk bookings in **start** order (given).
- Keep a **min-heap** of current **end times** (one per busy station).

- For each booking `(start, end)`:
    - **While** the earliest end `≤ start`, **pop** (those stations are free).
    - **Push** current `end` (occupy a station).
    - Track **peak heap size** → that's the **minimum stations required**.
- **Why it's correct:** Always reusing the station that frees earliest is optimal; the **peak concurrency** equals the number of stations needed.

Alternative (not implementing): Sweep-line with two arrays (starts/ends) + two pointers also yields the same peak.

Candidate — Plan (with Example integrated)

Example 1: `[[0,30],[5,10],[15,20]]` (already sorted)

- heap `[]`, `max_stations = 0`

1. `(0,30)` → free none → push `30` → heap `[30]` → `max_stations = 1`
2. `(5,10)` → `30 > 5` → push `10` → heap `[10,30]` → `max_stations = 2`
3. `(15,20)` → pop `10` (≤ 15) → heap `[30]` → push `20` → heap `[20,30]` → `max_stations = 2`
   Answer = 2

Adversarial (tests "while"): `[[0,5],[0,6],[0,7],[8,9]]`

- After first three, heap size = 3 → `max_stations = 3`
- At `(8,9)`, **while** pops `5,6,7` (all ≤ 8) → heap `[]`, then push `9`
- Final heap is 1 but **peak** was 3 → answer **3** (shows we must track the peak, not final size)

Candidate — Pseudocode

```
if intervals empty: return 0
heap = []                       # min-heap of end times
max_stations = 0
```

```
for (start, end) in intervals: # already sorted by start
    while heap and heap[0] <= start:
        pop heap                # free all finished stations
    push end into heap          # occupy a station
    max_stations = max(max_stations, len(heap))
return max_stations
```

Interviewer:
Please implement with concise, English-like naming and right-side comments.

Candidate — Implementation (Python, concise + right-side comments)

```
import heapq
from typing import List

def min_stations(intervals: List[List[int]]) -> int:
    if not intervals: return 0                              # No bookings → no
stations

    heap: List[int] = []                                    # Min-heap of end
times
    max_stations = 0                                        # Peak concurrent
usage

    for start, end in intervals:                            # Process bookings
(already sorted)
        while heap and heap[0] <= start:                    # Free all stations
that ended
            heapq.heappop(heap)
        heapq.heappush(heap, end)                           # Occupy a station
with this booking
        max_stations = max(max_stations, len(heap))         # Track peak usage

    return max_stations                                     # Minimum stations
required
```

Interviewer:
Great. Let’s dry-run your function on a few cases.

Candidate — Dry Runs

1. `[[0,30],[5,10],[15,20]]` → **2**
   - push 30 → `[30]` (1)
   - push 10 → `[10,30]` (2)
   - pop 10 → `[30]`, push 20 → `[20,30]` (2 peak) → **2**
2. `[[7,10],[10,12]]` → **1** (half-open reuse)
   - push 10 → `[10]` (1)
   - pop 10 (≤ 10) → `[]`, push 12 → `[12]` (1) → **1**
3. `[[0,5],[0,6],[0,7],[8,9]]` → **3**
   - push 5 → `[5]` (1)
   - push 6 → `[5,6]` (2)
   - push 7 → `[5,6,7]` (3)
   - pop 5, pop 6, pop 7 (all ≤ 8), push 9 → `[9]` (peak remained 3) → **3**
4. `[]` → **0**

Interviewer:
Add a tiny test harness with expected outputs.

Candidate — Quick Tests (expected in comments)

```
def _test():
    print(min_stations([[0,30],[5,10],[15,20]]))        # 2
    print(min_stations([[7,10],[10,12]]))               # 1
    print(min_stations([[0,5],[0,6],[0,7],[8,9]]))      # 3
    print(min_stations([[0,10],[10,20],[20,30]]))       # 1
    print(min_stations([[1,3],[2,4],[3,5],[4,6]]))      # 2
    print(min_stations([]))                              # 0
```

```
# _test()
```

Interviewer:
Explain the time and space complexity with reasoning.

Candidate — Complexity (with reasoning)

- **Time:**
  We process each of the `n` intervals once. Each interval is **pushed** once; each is **popped** once total across the run. Heap operations are **O(log n)** each, so total heap work is **O(n log n)**. (We didn't sort because input is already sorted.)

- **Space:**
  The heap holds at most the **peak overlap** count (≤ `n`) end times → **O(n)** worst-case extra space.

Interviewer (targeted follow-ups)

1. Why use **`while heap and heap[0] <= start`** instead of an `if`?
2. If the intervals were **closed** `[start, end]`, how would the freeing condition change?
3. Could you solve this without a heap?

Candidate — Answers

1. Multiple stations may have freed before `start`. We must pop **all** of them to avoid overcounting. An `if` would pop only one and give a wrong peak.
2. With closed intervals, `[a,b]` and `[b,c]` **overlap** at `b`, so the condition would be `heap[0] < start` (strict).
3. Yes—**sweep line**: sort arrays of starts and ends (if not already sorted), walk with two pointers, increment on each start, decrement on each end `≤ start` (half-open semantics), track the peak. Same **O(n log n)**.

Interviewer — Wrap-up Feedback

Excellent work:

- Clear structured summary with title, must-satisfy, output, and implications
- Strong clarifications (already sorted, half-open)
- Correct **Greedy + Min-Heap** approach with **while-freeing** and **peak tracking**
- Concise, English-like code with right-side comments and meaningful names
- Thorough dry runs, tests, and complexity reasoning

Ratings

- Coding: **4/4**
- Problem Solving: **4/4**
- Communication: **4/4**