LeetCode Problems

[The Number of the Smallest Unoccupied Chair](https://leetcode.com/problems/the-number-of-the-smallest-unoccupied-chair/)

class Solution {

public int smallestChair(int[][] times, int targetFriend) {

int n = times.length;

Integer[] order = new Integer[n];

for (int i = 0; i < n; i++) order[i] = i;

Arrays.sort(order, (a, b) -> Integer.compare(times[a][0], times[b][0]));

PriorityQueue<Integer> emptySeats = new PriorityQueue<>();

PriorityQueue<int[]> takenSeats = new PriorityQueue<>(Comparator.comparingInt(a -> a[0]));

for (int i = 0; i < n; i++) emptySeats.offer(i);

for (int i : order) {

int arrival = times[i][0], leave = times[i][1];

while (!takenSeats.isEmpty() && takenSeats.peek()[0] <= arrival) {

emptySeats.offer(takenSeats.poll()[1]);

}

int seat = emptySeats.poll();

if (i == targetFriend) return seat;

takenSeats.offer(new int[]{leave, seat});

}

return -1;

}

}

[Divide Intervals Into Minimum Number of Groups](https://leetcode.com/problems/divide-intervals-into-minimum-number-of-groups/)

class Solution {

public int minGroups(int[][] intervals) {

int n = intervals.length;

int[] start\_times = new int[n];

int[] end\_times = new int[n];

for (int i = 0; i < n; i++) {

start\_times[i] = intervals[i][0];

end\_times[i] = intervals[i][1];

}

Arrays.sort(start\_times);

Arrays.sort(end\_times);

int end\_ptr = 0, group\_count = 0;

for (int start : start\_times) {

if (start > end\_times[end\_ptr]) {

end\_ptr++;

} else {

group\_count++;

}

}

return group\_count;

}

}