1942. The Number of the Smallest Unoccupied Chair

There is a party where `n` friends numbered from `0` to `n - 1` are attending. There is an **infinite** number of chairs in this party that are numbered from `0` to `infinity`. When a friend arrives at the party, they sit on the unoccupied chair with the **smallest number**.

- For example, if chairs `0`, `1`, and `5` are occupied when a friend comes, they will sit on chair number `2`.

When a friend leaves the party, their chair becomes unoccupied at the moment they leave. If another friend arrives at that same moment, they can sit in that chair.

You are given a **0-indexed** 2D integer array `times` where `times[i] = [arrivali, leavingi]`, indicating the arrival and leaving times of the `i`th friend respectively, and an integer `targetFriend`. All arrival times are **distinct**.

Return *the **chair number** that the friend numbered* `targetFriend` *will sit on*.

Example 1:

```
Input: times = [[1,4],[2,3],[4,6]], targetFriend = 1
Output: 1
Explanation:
- Friend 0 arrives at time 1 and sits on chair 0.
- Friend 1 arrives at time 2 and sits on chair 1.
- Friend 1 leaves at time 3 and chair 1 becomes empty.
- Friend 0 leaves at time 4 and chair 0 becomes empty.
- Friend 2 arrives at time 4 and sits on chair 0.
Since friend 1 sat on chair 1, we return 1.
```

Example 2:

```
Input: times = [[3,10],[1,5],[2,6]], targetFriend = 0
Output: 2
Explanation:
- Friend 1 arrives at time 1 and sits on chair 0.
- Friend 2 arrives at time 2 and sits on chair 1.
- Friend 0 arrives at time 3 and sits on chair 2.
- Friend 1 leaves at time 5 and chair 0 becomes empty.
- Friend 2 leaves at time 6 and chair 1 becomes empty.
- Friend 0 leaves at time 10 and chair 2 becomes empty.
Since friend 0 sat on chair 2, we return 2.
```

Constraints:

- `n == times.length`
- `2 <= n <= 10^4`
- `times[i].length == 2`
- `1 <= arrivali < leavingi <= 10^5`
- `0 <= targetFriend <= n - 1`
- Each `arrivali` time is **distinct**.

1942. The Number of the Smallest Unoccupied Chair

```
/*
   Brute force
   Time complexity: O(nlogn+n^2)=O(n^2)
   Space complexity: O(n)
*/
class Solution {
   public:
      int smallestChair(std::vector<std::vector<int>>& times, int targetFriend){
         int n=times.size();

         int target_friend_arrival=times[targetFriend][0];

         std::sort(times.begin(),times.end());

         std::vector<int> chairs(n,-1);

         for(auto& time: times){
            int arrival=time[0];
            int leave=time[1];
            int i=0;
            bool is_occupied=false;
            while(i<n&&!is_occupied){
               if(chairs[i]<=arrival){
                  if(arrival==target_friend_arrival) return i;
                  chairs[i]=leave;
                  is_occupied=true;
               }
               else is_occupied=false;
               i++;
            }
         }
         return -1;
      }
};
```

1942. The Number of the Smallest Unoccupied Chair

```
/*
   Min heaps
   Time complexity: O(nlogn)
   Space complexity: O(2n)=O(n)
*/
typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;
class Solution {
   public:
      int smallestChair(std::vector<std::vector<int>>& times, int targetFriend){
         int n=times.size();
         int target_friend_arrival=times[targetFriend][0];
         std::priority_queue<ii,vii,std::greater<ii>> occupied_chairs;
         std::priority_queue<int,std::vector<int>,std::greater<int>> availabe_chairs;
         for(int i=0;i<n;++i) availabe_chairs.push(i);
         std::sort(times.begin(),times.end());
         for(int i=0;i<n;++i){
            int arrival=times[i][0];
            int leave=times[i][1];
            while(!occupied_chairs.empty() && occupied_chairs.top().first<=arrival){
               availabe_chairs.push(occupied_chairs.top().second);
               occupied_chairs.pop();
            }
            if(!availabe_chairs.empty()){
               int chair=availabe_chairs.top();
               availabe_chairs.pop();
               if(arrival==target_friend_arrival) return chair;
               occupied_chairs.push({leave,chair});
            }
         }
         return -1; // Never reached
      }
};
```