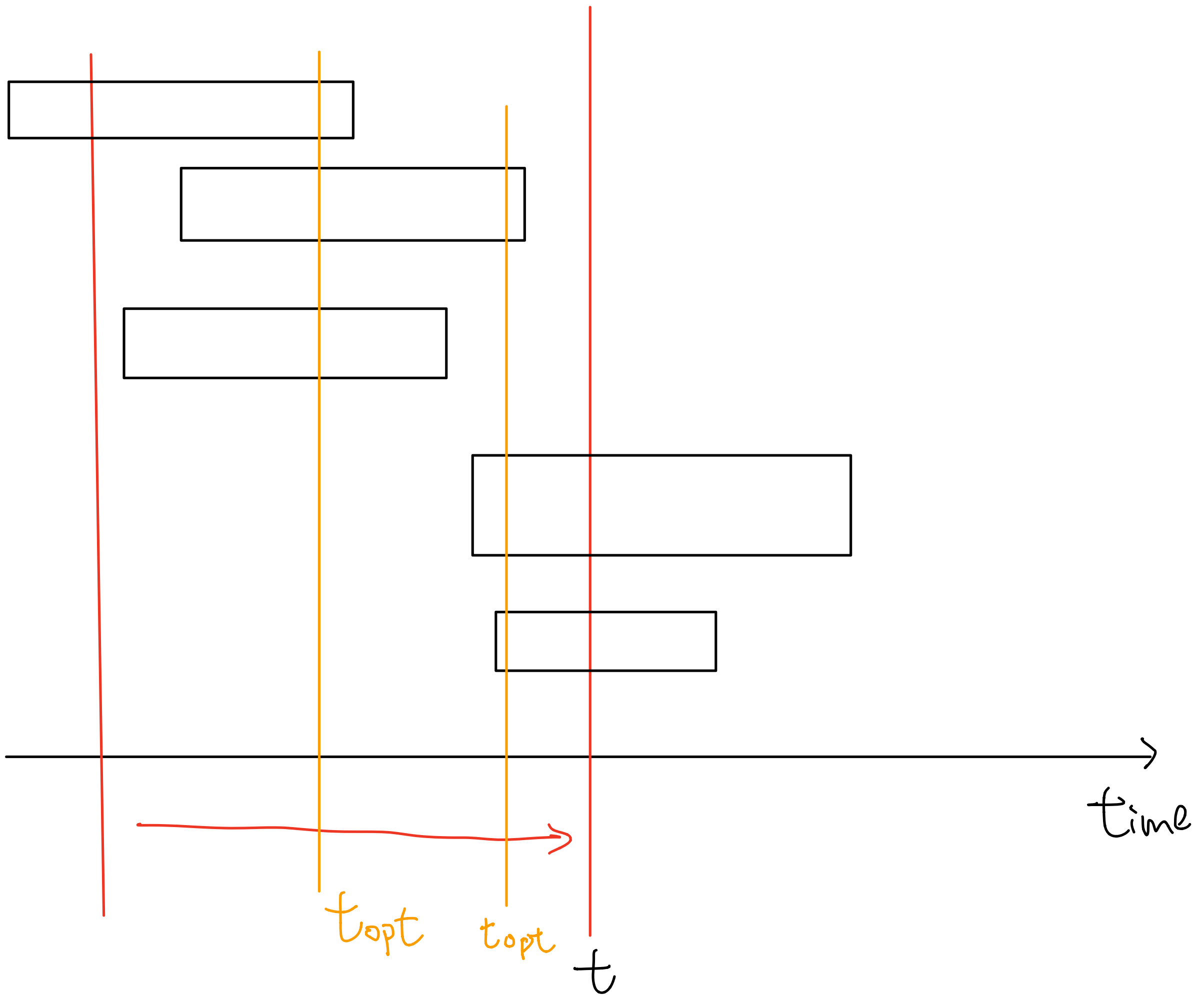
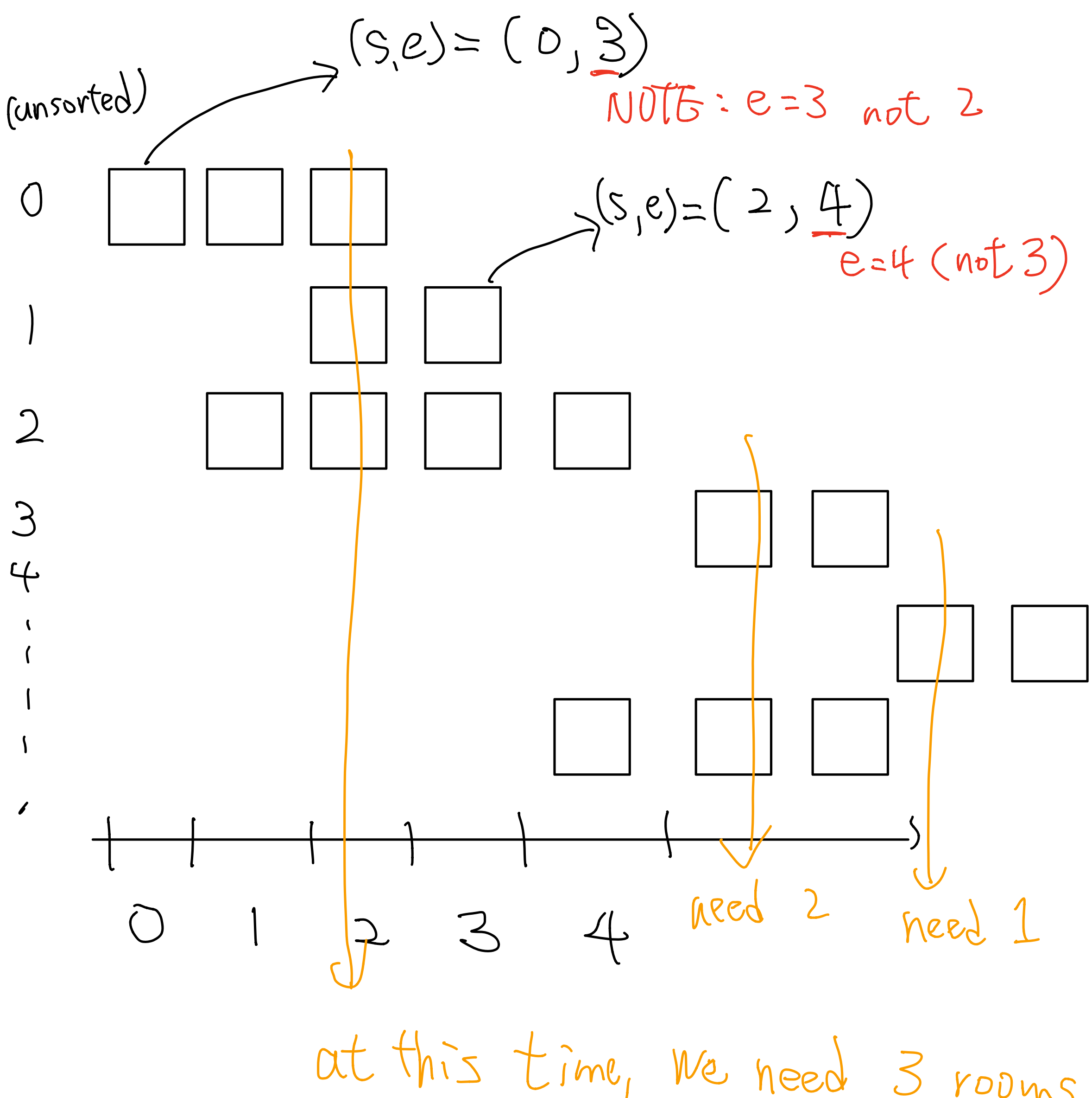
Proof: Why using a PQ will works?



$$ans = \max_{t}(\text{\# of conflicts at each } t)$$

At t_{opt}, we need at least 3 rooms

Or more correctly, we can visualize as slots



Proof: if $k = \max_t (\#\text{ of conflicts at } t)$

If we only have $k' < k$ rooms, apparently we will not able to provide enough rooms at $t = \text{argmax}(\cdots)$

If we have exactly k rooms, because at every t, there are at most k slots occupying rooms.

This proves why k rooms are enough

So the rest of proof is:

Why a sorted intervals (w.r.t s)
+ PQ can compute $\Rightarrow$ $\boxed{\max_t(\text{\# of conflicts at each } t)}$

A naive solution will be:

```
for slot in all slots:
    check the number of slot collide this slot
        (including itself)
```

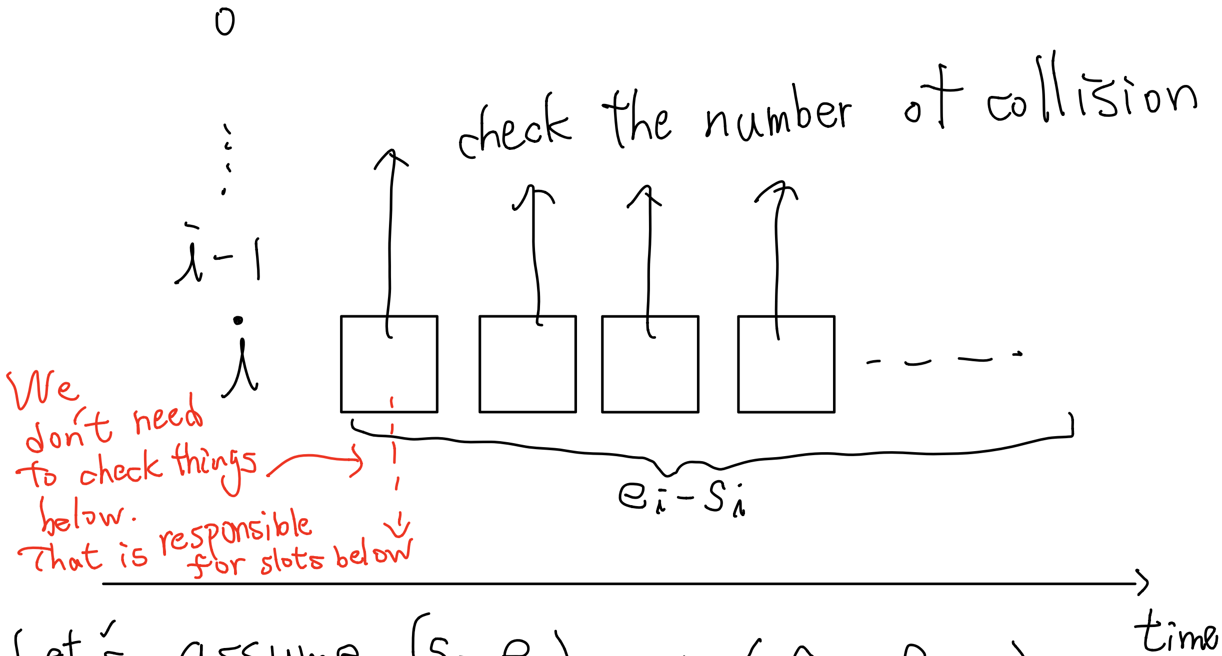
A slight improvement will be

```
slots.sort()
for i in range(len(slots)):
    for j in range(0, i):
        check if slot_i collide slot_j
```

($\because$ Ex. if $slot_0$ collide $slot_1$, we only need to consider (0, 1) and don't need to consider (1, 0))

Using the idea above and observe that the number of slots are huge.

$\Rightarrow$ Try using (s, e) pair to compute it.



Let's assume $(s_0, e_0) \cdots (s_{n-1}, e_{n-1})$ is sorted w.r.t. `s`

When our current index is i, and we have seen $[0, i)$

Because $[0, i]$ are sorted w.r.t. s,

all $j < i$, $s_j \le s_i$

if we open a new room, that must be caused by $[s_i, s_i+1]$ grid collide other grids in the existing rooms.

* If $[s_i, s_{i+1}]$ does not collide all existing rooms,
  we will need to remove the smallest end times from the queue because it is useless!
  We have a new meeting added and rooms are the same. We must update the largest end times in the rooms!
* If $[s_i, s_{i+1}]$ collide all existing rooms,
  we will need to open a room.

Finally, because our intervals $\max_t$ (# of conflicts at t) is fixed

and greedy will ONLY open a room when a meeting collides ALL rooms

$\Rightarrow$ This greedy algorithm is correct

More about the proof:

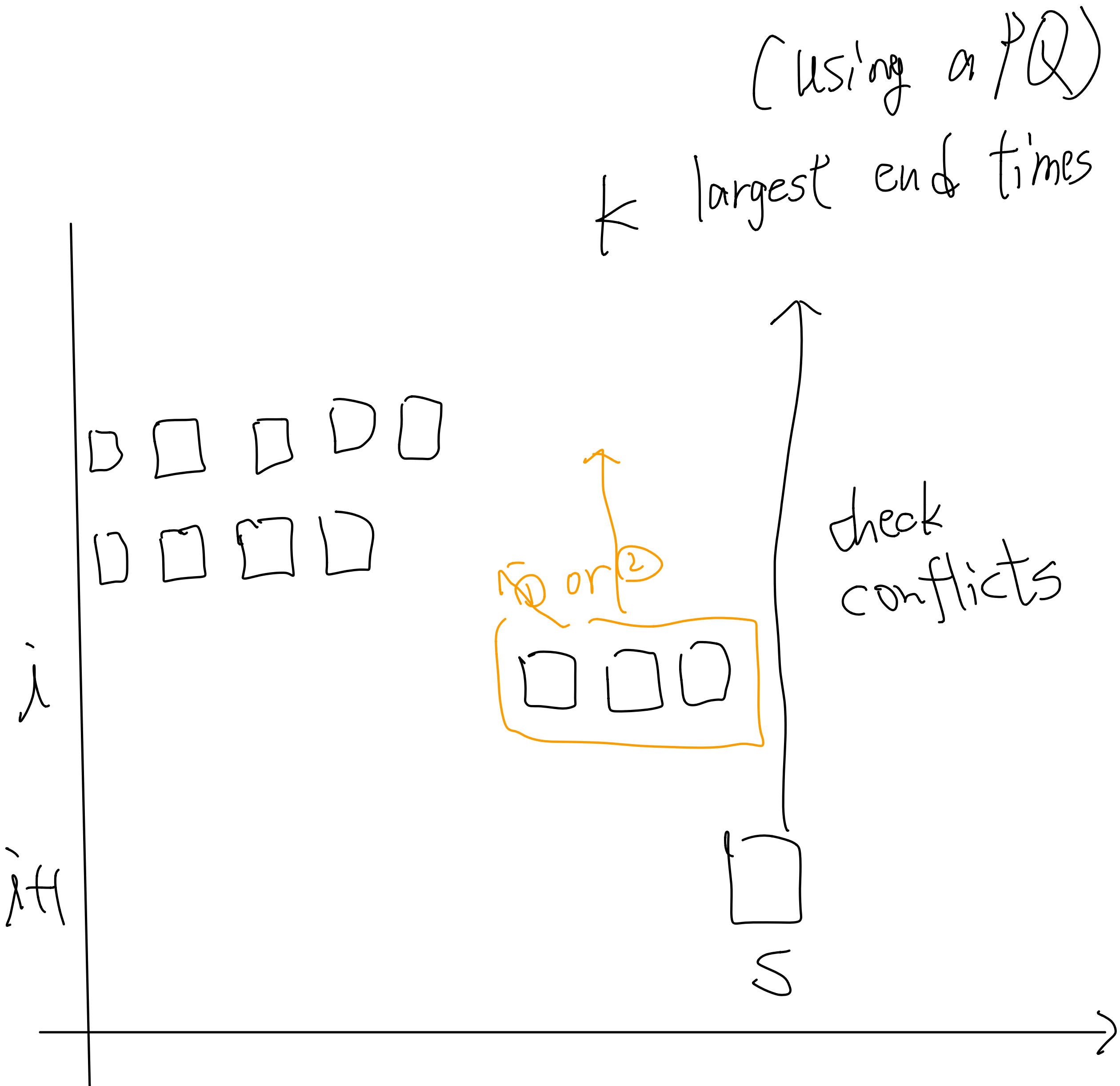
① Because the greedy algorithm never arrange two conflicting meeting in the same room.

$\Rightarrow$ # rooms found by greedy $\geq$ $\max_t$ (# of conflicts at t)

② Greedy only open a room when one meeting collides meetings "above" ($[0, i)$ for i)

$\Rightarrow$ # rooms found by greedy $\leq$ $\max_t$ (# of conflicts of t)

$\Rightarrow$ # rooms found by greedy $= \max_t$ (- - - -)



I think the reason why we choose ① over ② is that:
Even ② can produce an optimal sol, put ① will not be worse!