Lecture 9: Greedy Algorithms

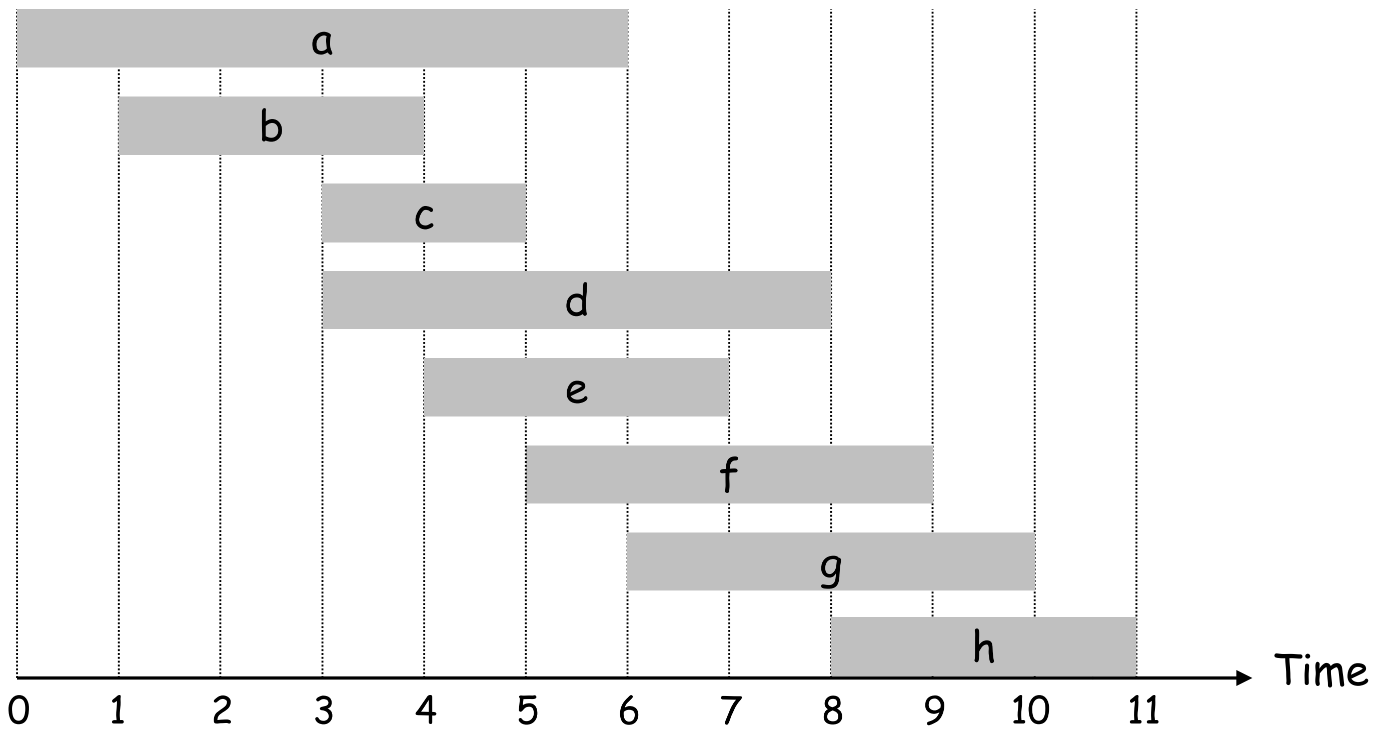
A greedy algorithm always makes the choice that looks best at the moment and adds it to the current partial solution.

Greedy algorithms don't always yield optimal solutions, but when they do, they're usually the simplest and most efficient algorithms available.

Interval Scheduling

Interval scheduling.

- Job j starts at s_j and finishes at f_j.
- Two jobs are compatible if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



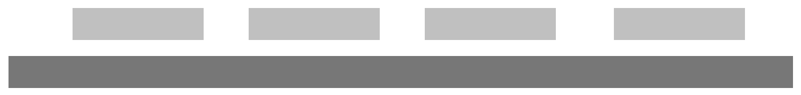
Interval Scheduling: Greedy Algorithms

Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

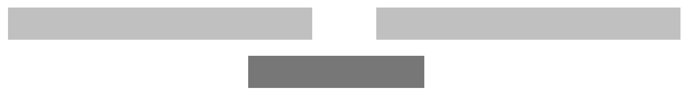
- [Earliest start time] Consider jobs in ascending order of start time s_j.
- [Earliest finish time] Consider jobs in ascending order of finish time f_j.
- [Shortest interval] Consider jobs in ascending order of interval length $f_j - s_j$.
- [Fewest conflicts] For each job, count the number of conflicting jobs c_j. Schedule in ascending order of conflicts c_j.

Interval Scheduling: Greedy Algorithms

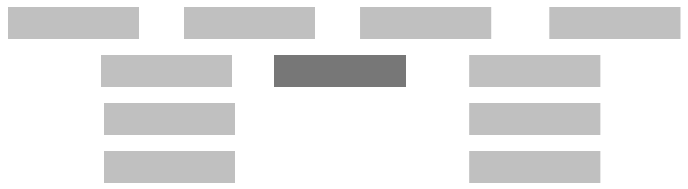
Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.



breaks earliest start time



breaks shortest interval



breaks fewest conflicts

Interval Scheduling: Greedy Algorithm

Greedy algorithm. Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

```
Sort jobs by finish times so that $f_1 \le f_2 \le \ldots \le f_n$
$A \leftarrow \emptyset$, $last \leftarrow 0$
for $j \leftarrow 1$ to $n$
    if $s_j \ge last$ then $A \leftarrow A \cup \{j\}$, $last \leftarrow f_j$
return $A$
```

Running time: $\Theta(n \log n)$.

- Remember the finish time of the last job added to A.
- Job j is compatible with A if $s_j \ge last$.

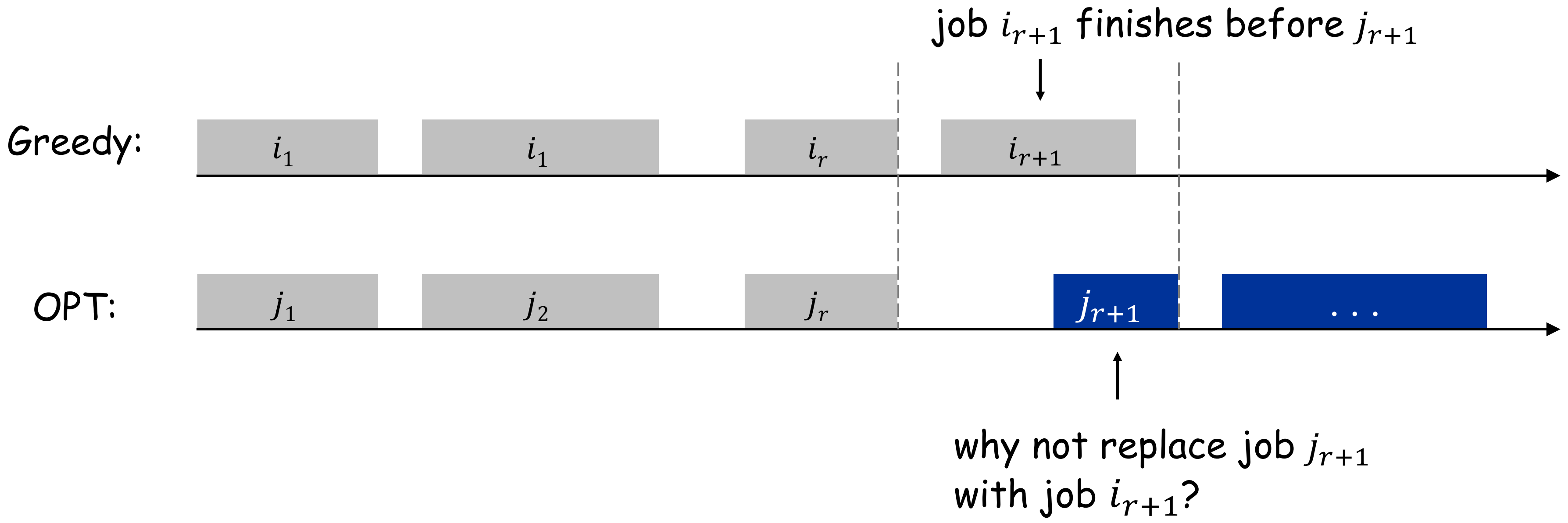
Remember: Correctness (optimality) of greedy algorithms is usually not obvious. Need to prove!

Interval Scheduling: Correctness

Theorem. Greedy algorithm is optimal.

Proof.

- Assume greedy is different from OPT. Let's see what's different.
- Let $i_1, i_2, \ldots\ i_k$ denote the set of jobs selected by greedy.
- Let $j_1, j_2, \ldots\ j_m$ denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.

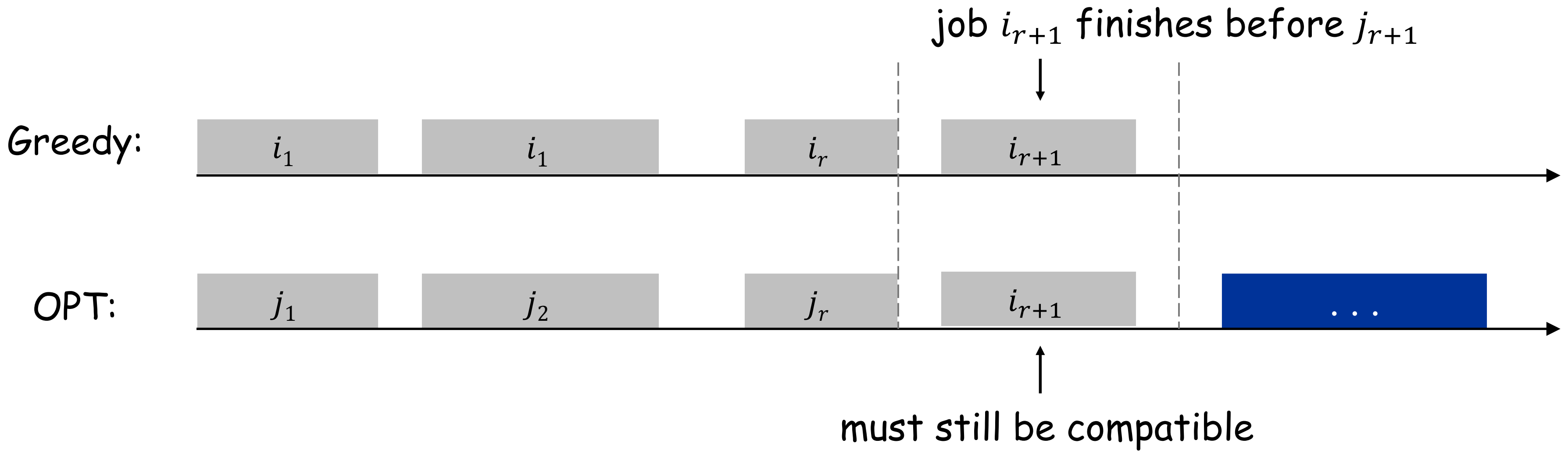

job i_{r+1} finishes before j_{r+1}
Greedy:
i_1
i_1
i_r
i_{r+1}
OPT:
j_1
j_2
j_r
j_{r+1}
. . .
why not replace job j_{r+1} with job i_{r+1}?

Interval Scheduling: Correctness

Theorem. Greedy algorithm is optimal.

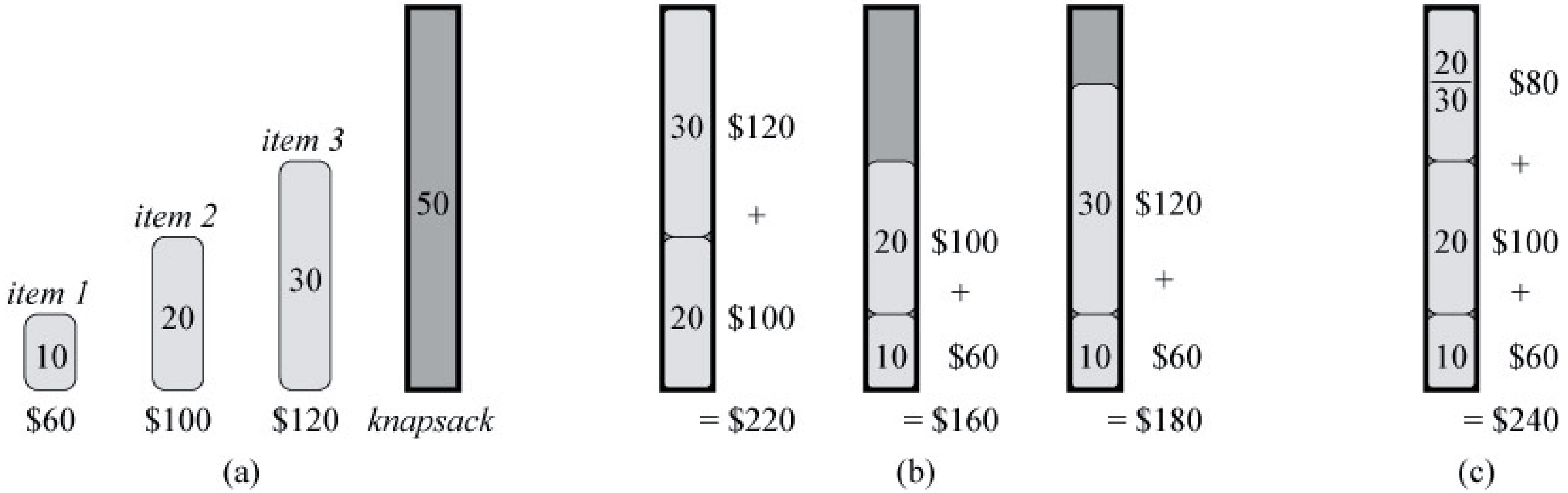
Proof.

- Assume greedy is different from OPT. Let's see what's different.
- Let $i_1, i_2, \ldots\ i_k$ denote the set of jobs selected by greedy.
- Let $j_1, j_2, \ldots\ j_m$ denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.



- Do this repeatedly until OPT is the same as greedy
  - Important: Quality of OPT doesn't change

The Fractional Knapsack Problem



Input: A set of n items, where item i has weight w_i and value v_i, and a knapsack with capacity W.

Goal: Find $0 \leq x_1, \dots, x_n \leq 1$ such that $\sum_{i=1}^{n} x_i w_i \leq W$ and $\sum_{i=1}^{n} x_i v_i$ is maximized.

- The x_i's must be 0 or 1: The 0/1 knapsack problem.
- The x_i's can take fractional values: The fractional knapsack problem

The Greedy Algorithm for Fractional Knapsack

```
Sort items so that $\frac{v_1}{w_1} > \frac{v_2}{w_2} > \cdots > \frac{v_n}{w_n}$
$w \leftarrow W$
for $i \leftarrow 1$ to $n$
    if $w_i \leq w$ then
        $x_i \leftarrow 1$
        $w \leftarrow w - w_i$
    else
        $x_i \leftarrow w/w_i$
        return
return
```

Idea:

- Sort all items by value-per-pound
- For each item, take as much as possible

Running time: $\Theta(n \log n)$

Note: This algorithm cannot solve the 0/1 version optimally.

Greedy Algorithm: Correctness

Theorem: The greedy algorithm is optimal.

Proof: We will assume that $\sum_{i=1}^{n} w_i \geq W$. Otherwise the algorithm is trivially optimal.

Let the greedy solution be $G = (x_1, x_2, \dots, x_k, 0, \dots, 0)$

- Note: All x_i's must be equal to 1, except possibly for $i = k$.

Consider any optimal solution $O = (y_1, y_2, \dots, y_n)$

- Note: Both G and O must fully pack the knapsack.

Look at the first item i where the two solutions differ.

- By greedy nature, $x_i > y_i$
- Let $x = x_i - y_i$

Greedy Algorithm: Correctness (continued)

We will modify O as follows:

- Set $y_i \leftarrow x_i$ and remove part of any items $i+1$ to item n of total weight xw_i
- This is always doable because in O, the total weight of item i to n is the same as that in G

After the modification:

- The total value cannot decrease, since all the subsequent items have lesser or equal value-per-pound.
- Since O is already an optimal solution, the value cannot increase.
- So the value must stay the same.

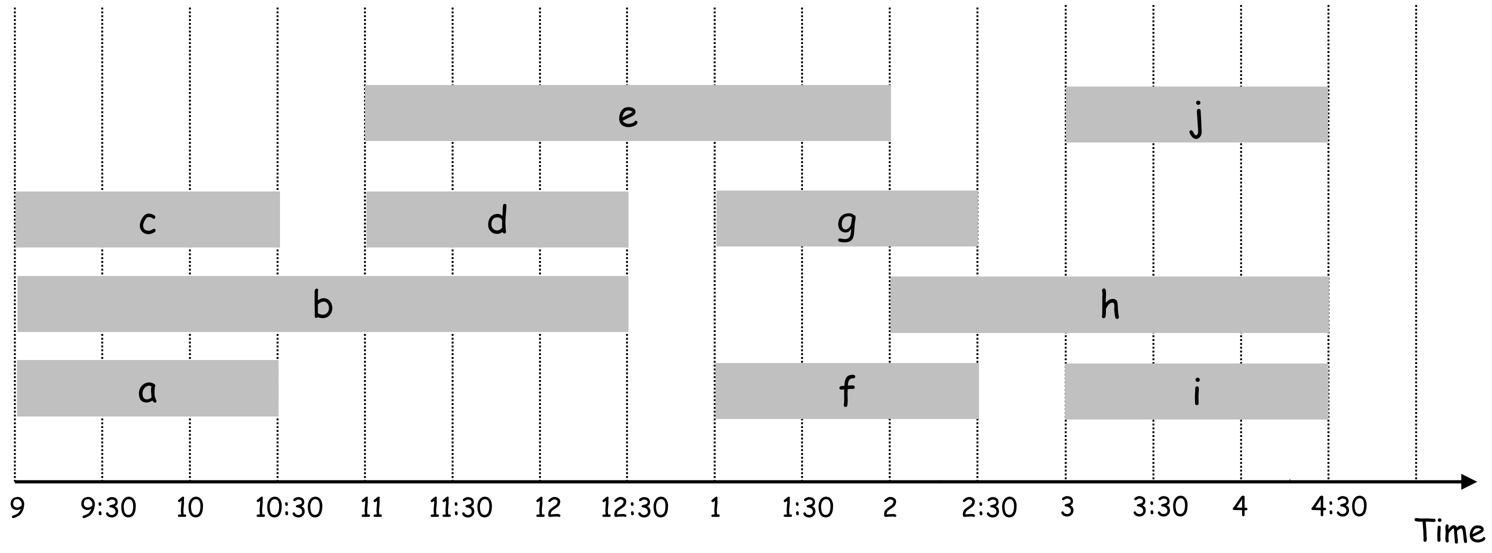
By repeating this process, we will eventually convert O into G, without changing the total value of the selection. Therefore G is also optimal.

Interval Partitioning

Interval partitioning.

- Lecture j starts at s_j and finishes at f_j.
- Goal: find the minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Ex: This schedule uses 4 classrooms to schedule 10 lectures.

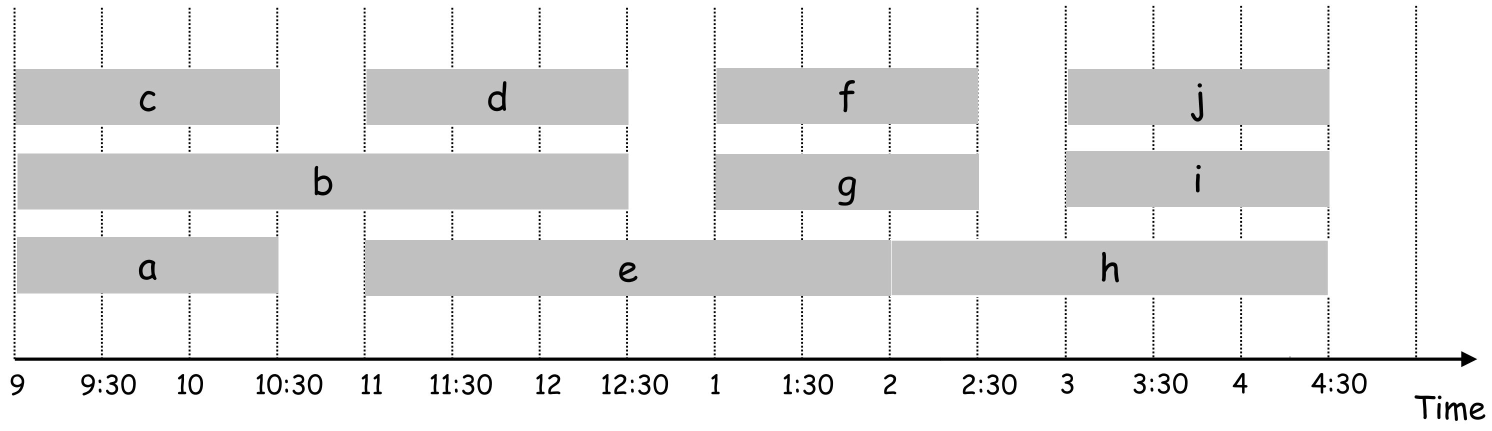


Interval Partitioning

Interval partitioning.

- Lecture j starts at s_j and finishes at f_j.
- Goal: find the minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Ex: This schedule uses only 3.



Interval Partitioning: Greedy Algorithm

Greedy algorithm. Consider lectures in increasing order of start time: assign lecture to any compatible classroom.

```
Sort intervals by starting time so that $s_1 \le s_2 \le \ldots \le s_n$.
$d \leftarrow 0$  // # classrooms used so far
for $j \leftarrow 1$ to $n$
    if lecture $j$ is compatible with some classroom $k$ then
        schedule lecture $j$ in classroom $k$
    else
        allocate a new classroom $d + 1$
        schedule lecture $j$ in classroom $d + 1$
        $d \leftarrow d + 1$
```

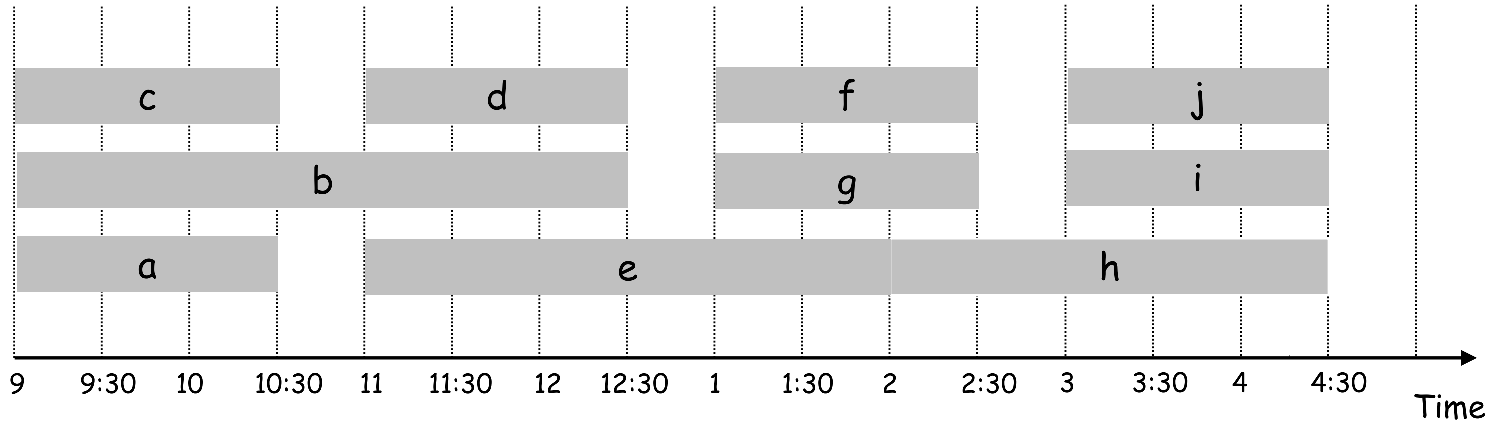
Interval Partitioning: Lower Bound on Optimal Solution

Def. The depth of a set of open intervals is the maximum number that contain any time instance.

Key observation. Number of classrooms needed $\geq$ depth.

Ex: Depth of schedule below = 3 $\Rightarrow$ this schedule is optimal.

We will show: The # classrooms used by the greedy algorithm = depth.



Interval Partitioning: Correctness

Theorem. Greedy algorithm is optimal.

Pf.

- Let d = number of classrooms that the greedy algorithm uses.
- Classroom d is opened because we needed to schedule a lecture, say j, that is incompatible with all $d-1$ other classrooms.
- Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than s_j.
- Thus, we have d lectures overlapping at time $s_j + \varepsilon$.
- Key observation $\Rightarrow$ depth $\geq d$.

Interval Partitioning: Running Time

```
Sort intervals by starting time so that $s_1 \leq s_2 \leq \ldots \leq s_n$.
$d \leftarrow 0$  // # classrooms used so far
for $j \leftarrow 1$ to $n$
    if lecture $j$ is compatible with some classroom $k$ then (*)
        schedule lecture $j$ in classroom $k$  (**)
    else
        allocate a new classroom $d+1$
        schedule lecture $j$ in classroom $d + 1$
        $d \leftarrow d+1$
```

Running time: $O(n \log n)$

- Brute-force implementation of line `(*)` takes $O(n)$ time $\Rightarrow O(n^2)$ in total
- Observation: If j is not compatible with the classroom with the earliest finish time, then j is not compatible with any other classroom
- Keep the classrooms in a minimum priority queue, with finish time of the last job being the key
  - Line `(**)` is an "increase-key" operation: $O(\log n)$ time
  - It always increases the key of the minimum element of the heap

Note: Not easy to show $\Theta(n \log n)$. Just write big-Oh whenever in doubt.