CSC373

Week 2: Greedy Algorithms

Announcements

- First tutorial tomorrow!
- Details on Piazza
- First Assignment to be posted tomorrow (May 19) after tutorial
- Due June 1

Recap

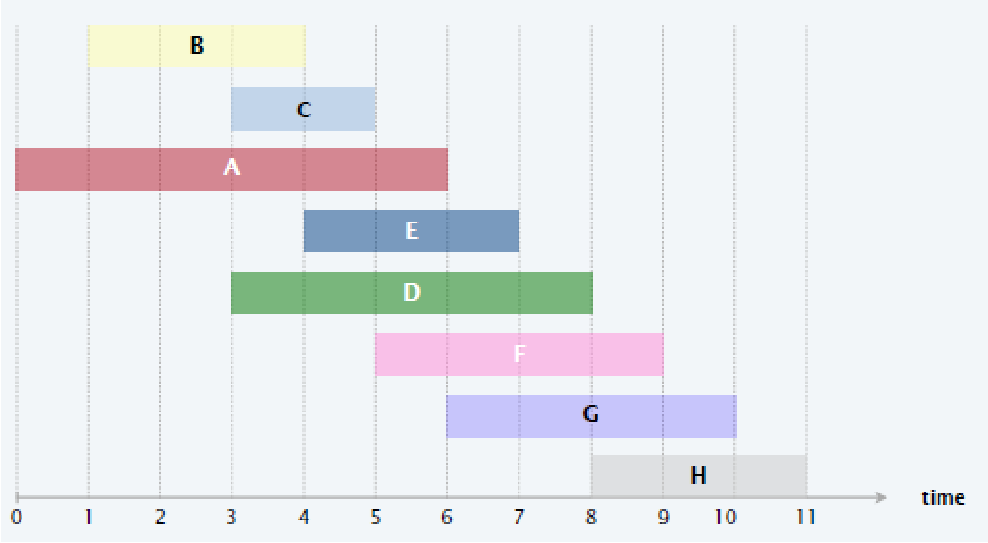
- Divide & Conquer
  - Master theorem
  - Counting inversions in $O(n \log n)$
  - Finding closest pair of points in $\mathbb{R}^2$ in $O(n \log n)$
  - Fast integer multiplication in $O\left(n^{\log_2 3}\right)$
  - Fast matrix multiplication in $O\left(n^{\log_2 7}\right)$
  - Finding k^{th} smallest element (in particular, median) in $O(n)$

Greedy Algorithms

- Greedy/myopic algorithm outline
  - Goal: find a solution x maximizing/minimizing objective function f
  - Challenge: space of possible solutions x is too large
  - Insight: x is composed of several parts (e.g., x is a set or a sequence)
  - Approach: Instead of computing x directly…
    - Compute it one part at a time
    - Select the next part “greedily” to get the most immediate “benefit” (this needs to be defined carefully for each problem)
    - Polynomial running time is typically guaranteed
    - Need to prove that this will always return an optimal solution despite having no foresight

Interval Scheduling

- Problem
  - Job j starts at time s_j and finishes at time f_j
  - Two jobs i and j are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ don't overlap
    - Note: we allow a job to start right when another finishes
  - Goal: find maximum-size subset of mutually compatible jobs



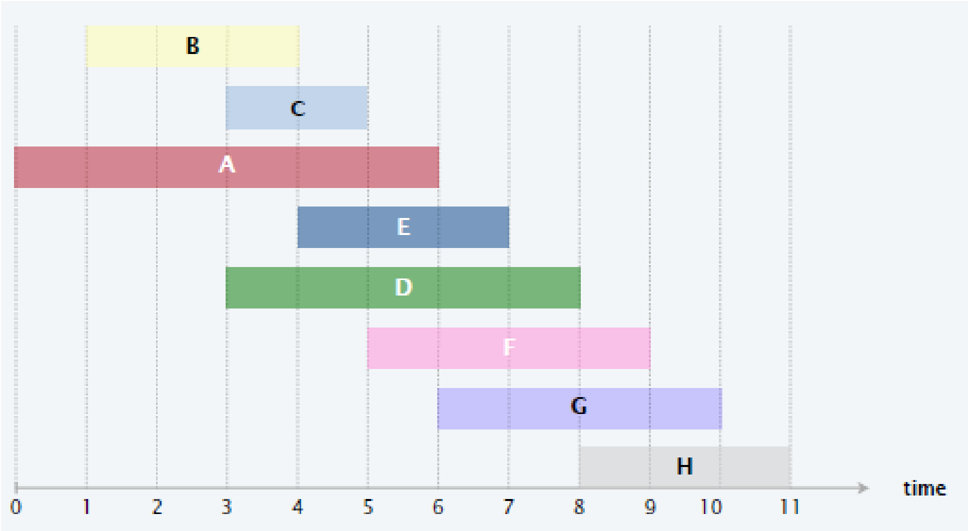
Interval Scheduling

- Greedy template
  - Consider jobs in some "natural" order
  - Take a job if it's compatible with the ones already chosen

- What order?
  - Earliest start time: ascending order of s_j
  - Earliest finish time: ascending order of f_j
  - Shortest interval: ascending order of $f_j - s_j$
  - Fewest conflicts: ascending order of c_j, where c_j is the number of remaining jobs that conflict with j

Example

- Earliest start time: ascending order of s_j
- Earliest finish time: ascending order of f_j
- Shortest interval: ascending order of $f_j - s_j$
- Fewest conflicts: ascending order of c_j, where c_j is the number of remaining jobs that conflict with j



Interval Scheduling

- Does it work?

Counterexamples for

earliest start time

shortest interval

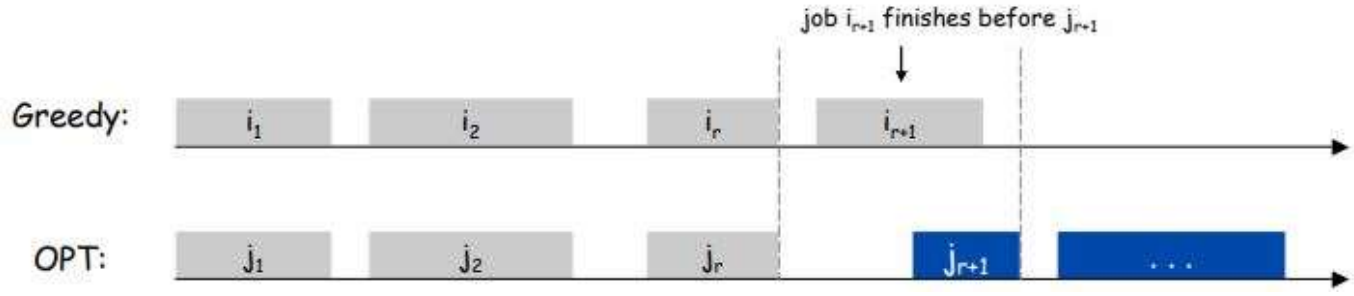
fewest conflicts

Interval Scheduling

- Implementing greedy with earliest finish time (EFT)
  - Sort jobs by finish time, say $f_1 \leq f_2 \leq \cdots \leq f_n$
    - $O(n \log n)$
  - For each job j, we need to check if it's compatible with *all* previously added jobs
    - Naively, this can take $O(n)$ time per job j, so $O(n^2)$ total time
    - We only need to check if $s_j \geq f_{i^*}$, where i^* is the *last added job*
      - For any jobs i added before i^*, $f_i \leq f_{i^*}$
      - By keeping track of f_{i^*}, we can check job j in $O(1)$ time
  - Running time: $O(n \log n)$

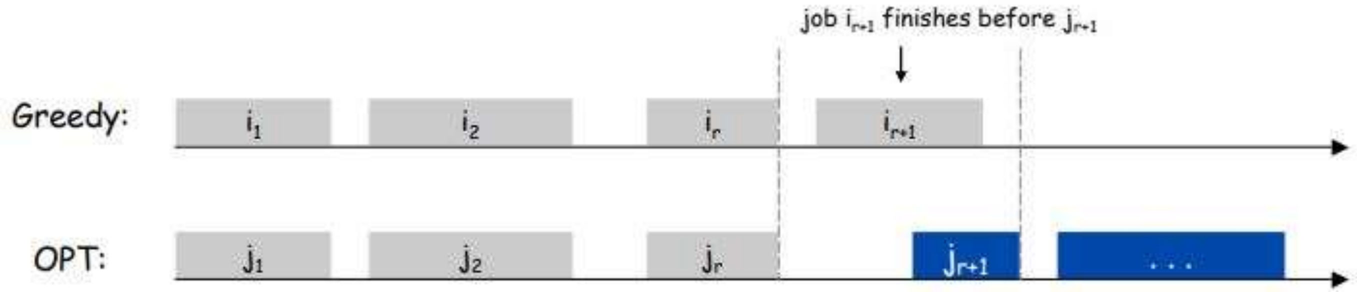
Interval Scheduling

- Proof of optimality by contradiction
  - Suppose for contradiction that greedy is not optimal
  - Say greedy selects jobs $i_1, i_2, \ldots, i_k$ sorted by finish time
  - Consider an optimal solution $j_1, j_2, \ldots, j_m$ (also sorted by finish time) which matches greedy for as many indices as possible
    - That is, we want $j_1 = i_1, \ldots, j_r = i_r$ for the greatest possible r
  - Both i_{r+1} and j_{r+1} must be compatible with the previous selection $(i_1 = j_1, \ldots, i_r = j_r)$



Interval Scheduling

- Proof of optimality by contradiction
  - Consider a new solution $i_1, i_2, \ldots, i_r, i_{r+1}, j_{r+2}, \ldots, j_m$
    - We have replaced j_{r+1} by i_{r+1} in our reference optimal solution
    - This is still feasible because $f_{i_{r+1}} \leq f_{j_{r+1}} \leq s_{j_t}$ for $t \geq r+2$
    - This is still optimal because m jobs are selected
    - But it matches the greedy solution in $r+1$ indices
      - This is the desired contradiction



Interval Scheduling

- Proof of optimality by induction
  - Let S_j be the subset of jobs picked by greedy after considering the first j jobs in the increasing order of finish time
    - Define $S_0 = \emptyset$
  - We call this partial solution *promising* if there is a way to extend it to an optimal solution by picking some subset of jobs $j + 1, \ldots, n$
    - $\exists T \subseteq \{j + 1, \ldots, n\}$ such that $O_j = S_j \cup T$ is optimal
  - Inductive claim: For all $t \in \{0,1, \ldots, n\}$, S_t is promising
  - If we prove this, then we are done!
    - For $t = n$, if S_n is promising, then it must be optimal (Why?)
    - We chose $t = 0$ as our base case since it is "trivial"

Interval Scheduling

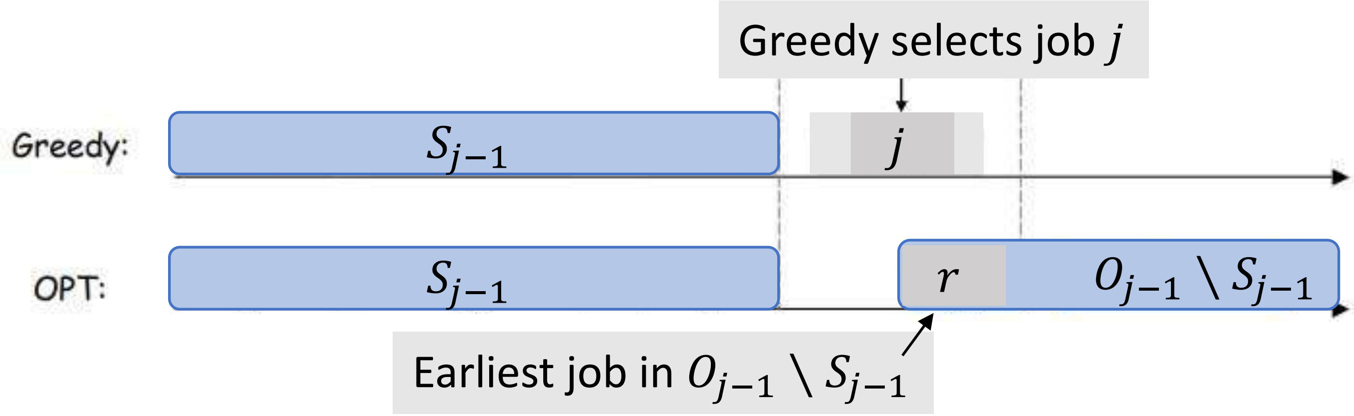
- Proof of optimality by induction
  - S_j is *promising* if $\exists T \subseteq \{j+1, \dots, n\}$ such that $O_j = S_j \cup T$ is optimal
  - Inductive claim: For all $t \in \{0,1,\dots,n\}$, S_t is promising
  - Base case: For $t = 0$, $S_0 = \emptyset$ is clearly promising
    - Any optimal solution extends it
  - Induction hypothesis: Suppose the claim holds for $t = j - 1$ and optimal solution O_{j-1} extends S_{j-1}
  - Induction step: At $t = j$, we have two possibilities:

    1) Greedy did not select job j, so $S_j = S_{j-1}$
       - Job j must conflict with some job in S_{j-1}
       - Since $S_{j-1} \subseteq O_{j-1}$, O_{j-1} also cannot include job j
       - $O_j = O_{j-1}$ also extends $S_j = S_{j-1}$

Interval Scheduling

- Proof of optimality by induction
  - Induction step: At $t = j$, we have two possibilities:

    2) Greedy selected job j, so $S_j = S_{j-1} \cup \{ j \}$
    - Consider the earliest job r in $O_{j-1} \setminus S_{j-1}$
    - Consider O_j obtained by replacing r with j in O_{j-1}
    - Prove that O_j is still feasible
    - O_j extends S_j, as desired!



Contradiction vs Induction

- Both methods make the same claim
  - "The greedy solution after j iterations can be extended to an optimal solution, $\forall j$"

- They also use the same key argument
  - "If the greedy solution after j iterations can be extended to an optimal solution, then the greedy solution after $j + 1$ iterations can be extended to an optimal solution as well"

  - For proof by induction, this is the key induction step
  - For proof by contradiction, we take the greatest j for which the greedy solution can be extended to an optimal solution, and derive a contradiction by extending the greedy solution after $j + 1$ iterations

Interval Partitioning

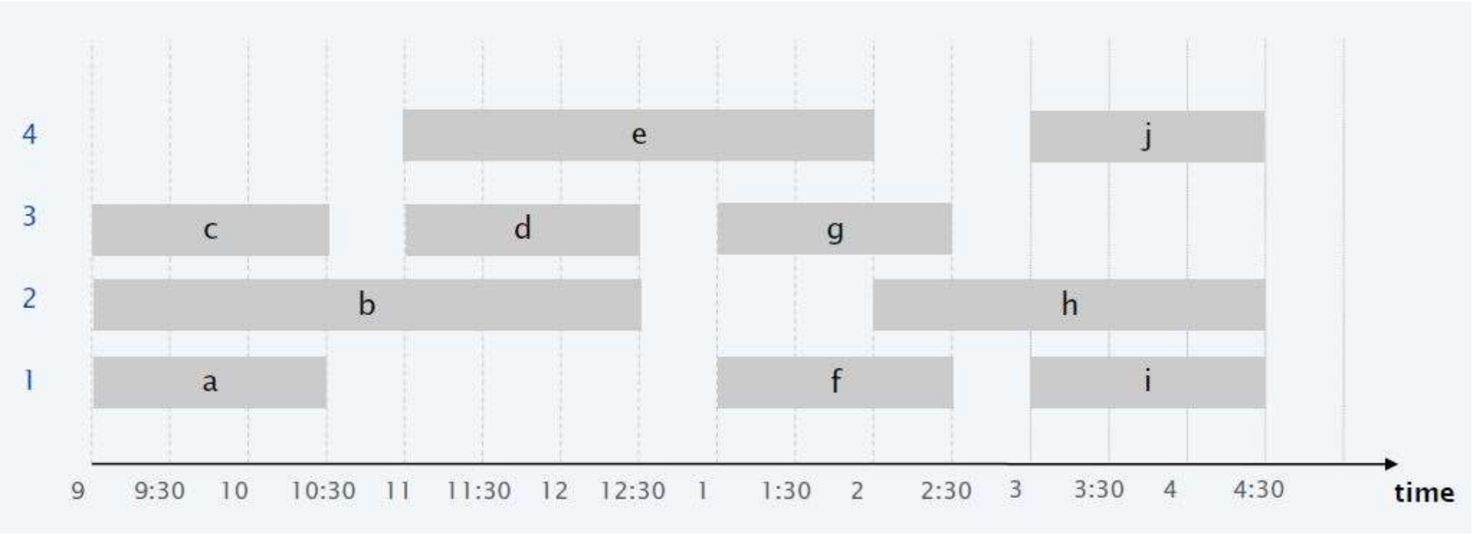
- Problem
  - Job j starts at time s_j and finishes at time f_j
  - Two jobs are compatible if they don't overlap
  - Goal: group jobs into fewest partitions such that jobs in the same partition are compatible

- One idea
  - Find the maximum compatible set using the previous greedy EFT algorithm, call it one partition, recurse on the remaining jobs.
  - Doesn't work (check by yourselves)

Interval Partitioning

- Think of scheduling lectures for various courses into as few classrooms as possible

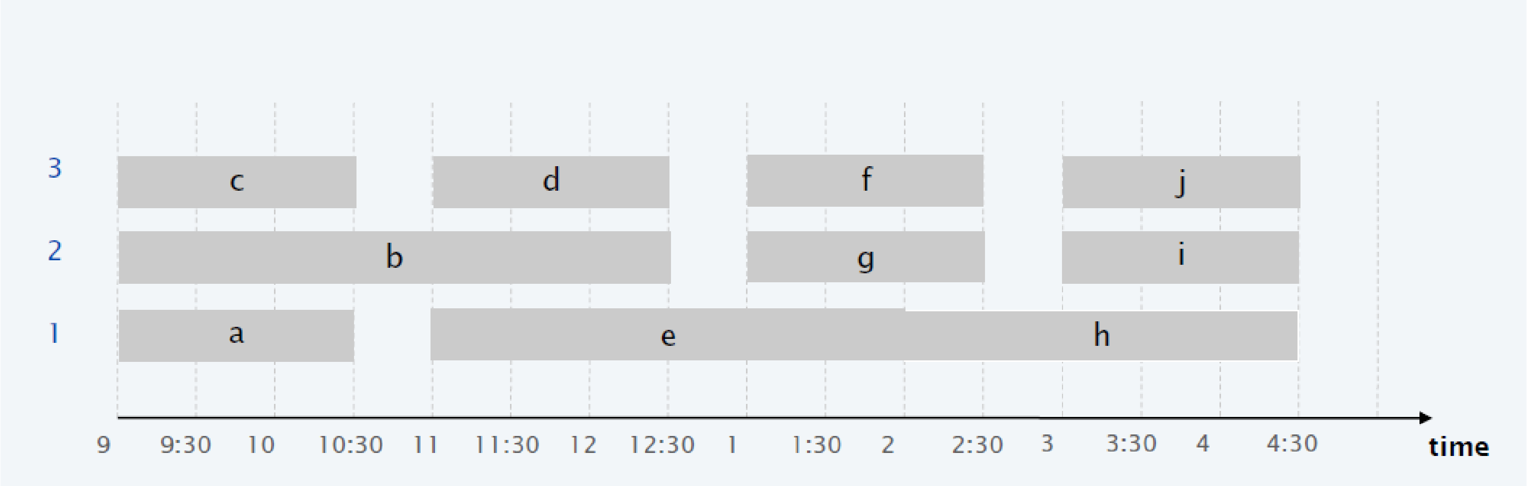
- This schedule uses 4 classrooms for scheduling 10 lectures



Interval Partitioning

- Think of scheduling lectures for various courses into as few classrooms as possible

- This schedule uses 3 classrooms for scheduling 10 lectures

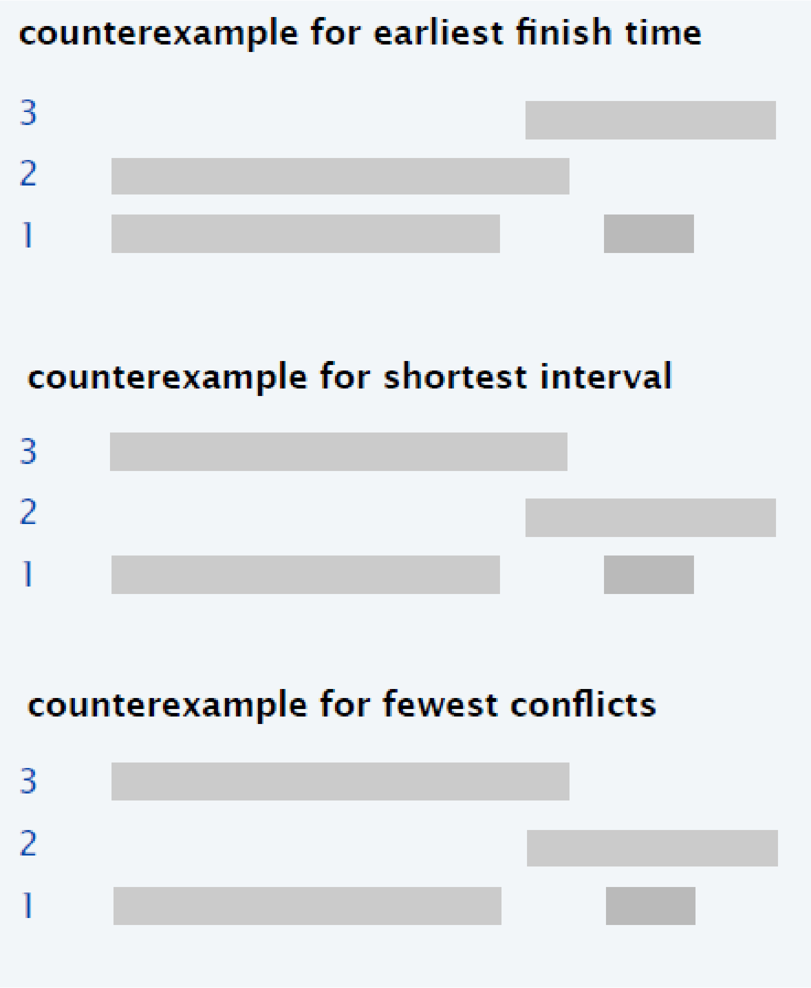


Interval Partitioning

- Let's go back to the greedy template!
  - Go through lectures in some "natural" order
  - Assign each lecture to an (arbitrary?) compatible classroom, and create a new classroom if the lecture conflicts with every existing classroom

- Order of lectures?
  - Earliest start time: ascending order of s_j
  - Earliest finish time: ascending order of f_j
  - Shortest interval: ascending order of $f_j - s_j$
  - Fewest conflicts: ascending order of c_j, where c_j is the number of remaining jobs that conflict with j

Interval Partitioning



- At least when you assign each lecture to an arbitrary compatible classroom, three of these heuristics do not work.
- The fourth one works! (next slide)

Interval Partitioning

EARLIESTSTARTTIMEFIRST(n, s_1, s_2, …, s_n, f_1, f_2, …, f_n)

SORT lectures by start time so that $s_1 \le s_2 \le \ldots \le s_n$.

$d \leftarrow 0$ ⟵ number of allocated classrooms

FOR $j = 1$ TO n

  IF lecture j is compatible with some classroom

    Schedule lecture j in any such classroom k.

  ELSE

    Allocate a new classroom $d + 1$.

    Schedule lecture j in classroom $d + 1$.

    $d \leftarrow d + 1$

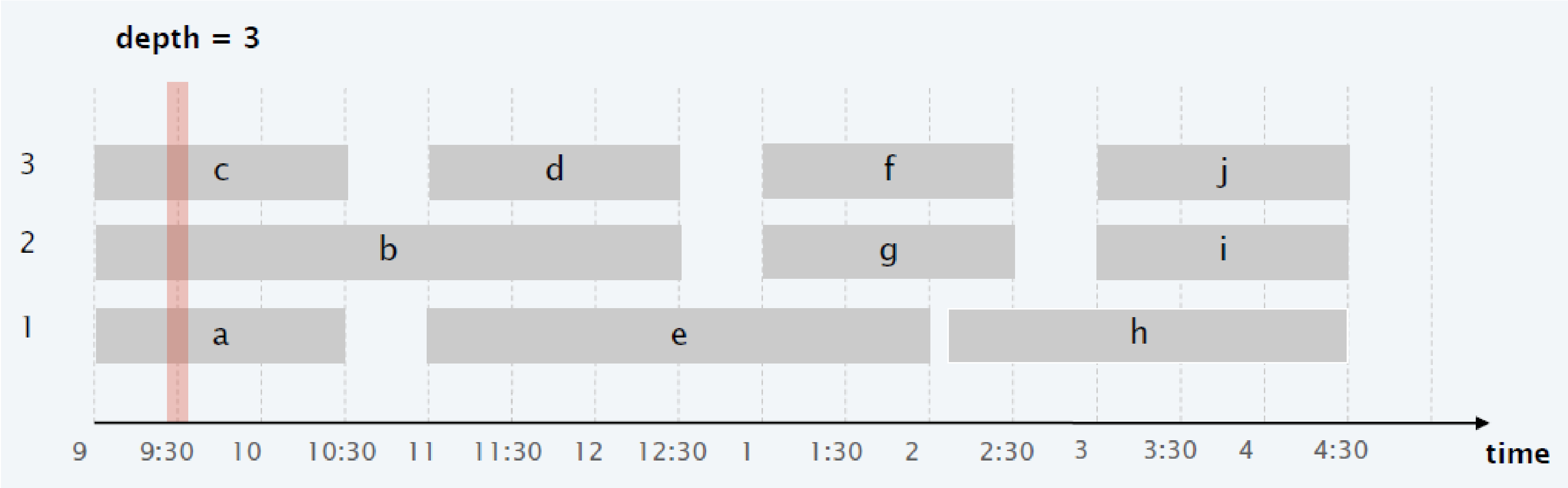
RETURN schedule.

Interval Partitioning

- Running time
  - Key step: check if the next lecture can be scheduled at some classroom
  - Store classrooms in a priority queue
    - key = latest finish time of any lecture in the classroom
  - Is lecture j compatible with some classroom?
    - Same as "Is s_j at least as large as the minimum key?"
    - If yes: add lecture j to classroom k with minimum key, and increase its key to f_j
    - Otherwise: create a new classroom, add lecture j, set key to f_j
  - $O(n)$ priority queue operations, $O(n \log n)$ time

Interval Partitioning

- Proof of optimality (lower bound)
  - # classrooms needed ≥ "depth"
    - depth = maximum number of lectures running at any time
    - Recall, as before, that job i runs in $[s_i, f_i)$
  - Claim: our greedy algorithm uses only these many classrooms!



Interval Partitioning

- Proof of optimality (upper bound)
  - Let d = # classrooms used by greedy
  - Classroom d was opened because there was a lecture j which was incompatible with some lectures already scheduled in each of $d - 1$ other classrooms
  - All these d lectures end after s_j
  - *<u>Since we sorted by start time</u>*, they all start at/before s_j
  - So, at time s_j, we have d mutually overlapping lectures
  - Hence, depth $\geq d =$ #classrooms used by greedy ■
  - Note: before we proved that #classrooms used by any algorithm (including greedy) ≥ depth, so greedy uses exactly as many classrooms as the depth.

Interval Graphs

NOT IN SYLLABUS

- Interval scheduling and interval partitioning can be seen as graph problems

- Input
  - Graph $G = (V, E)$
  - Vertices V = jobs/lectures
  - Edge $(i, j) \in E$ if jobs i and j are incompatible

- Interval scheduling = maximum independent set (MIS)
- Interval partitioning = graph coloring

Interval Graphs

NOT IN SYLLABUS

- MIS and graph coloring are NP-hard for general graphs
- But they're efficiently solvable for "interval graphs"
  - ➢ Graphs which can be obtained from incompatibility of intervals
  - ➢ In fact, this holds even when we are not given an interval representation of the graph
- Can we extend this result further?
  - ➢ Yes! Chordal graphs
    - o Every cycle with 4 or more vertices has a chord