Georgia Tech

CSE 6140/ CX 4140:

Computational Science and Engineering ALGORITHMS

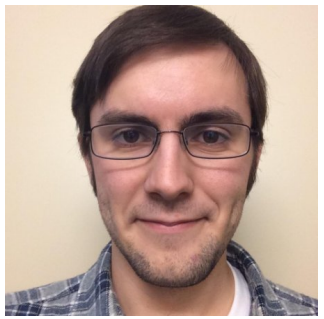
Instructor: Anne Benoit

Visiting Associate Professor, CSE

Based on slides by Bistra Dilkina

Course Staff

- Instructor: Prof. Anne Benoit (abenoit6)
- Office: Klaus 1341
- Office hour: Tue 1:20-2:20pm in Klaus 1335
- TA: Amrita (agupta375)
  - T 11, R 11
- TA: Patrick (plavin3)
  - M 2, W 1
- TA: Eisha (enathan3)
  - T 3, R 3
- TA: Caleb (dcrobins)
  - M 8.30, F 8.30
- TA: Harsh (harshs27)
  - W 11, F 2
- TA: Ruomeng (ruomengxu)
  - M 12.30, F 12.30



Emails:
@gatech.edu

Course Details

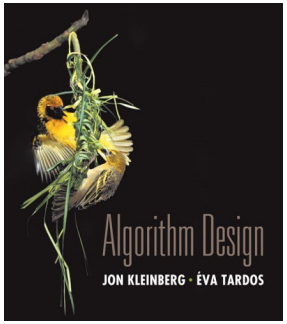
- Please send emails to all TAs
- T-square:
  - Syllabus
  - Course schedule
  - Slides
  - Assignments
  - Emails
  - Grades
- Piazza:
  - http://piazza.com/gatech/fall2017/cse6140cx4140
  - By students for students - No HW answers!

Greedy Algorithms

Greedy-choice property: we can assemble a globally optimal solution by making locally optimal (greedy) choices

i.e. we make the choice that looks best given the current partial solution

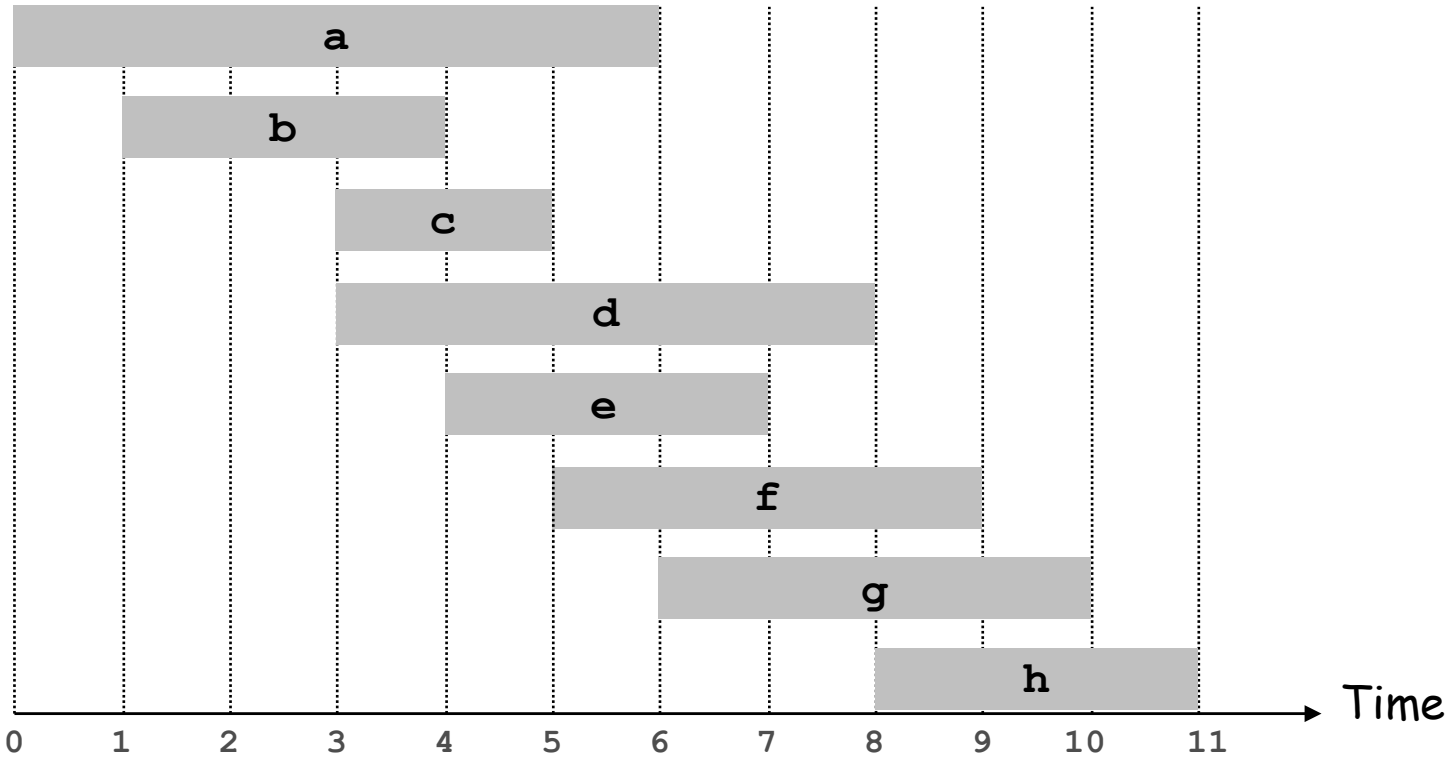
Interval Scheduling [KT 4]



Interval Scheduling

Interval scheduling.

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



Interval Scheduling: Greedy Algorithm

Greedy algorithm. Choose next job to add to solution as the one with earliest finish time that it is compatible with the ones already taken.

O(n log n) Sort jobs by finish times so that $f_1 \le f_2 \le \ldots \le f_n$.

set of jobs selected

$A \leftarrow \phi$

$f_{j^*} = 0$

O(n) for $j = 1$ to n {

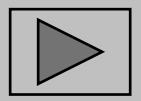
  if (job j compatible with A: $s_j \ge f_{j^*}$)

    $A \leftarrow A \cup \{j\}$

    $f_{j^*} = f_j$

}

return A



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

- Remember job j^* that was added last to A.
- Job j is compatible with A if $s_j \ge f_{j^*}$.

Greedy is optimal (greedy stays ahead argument)

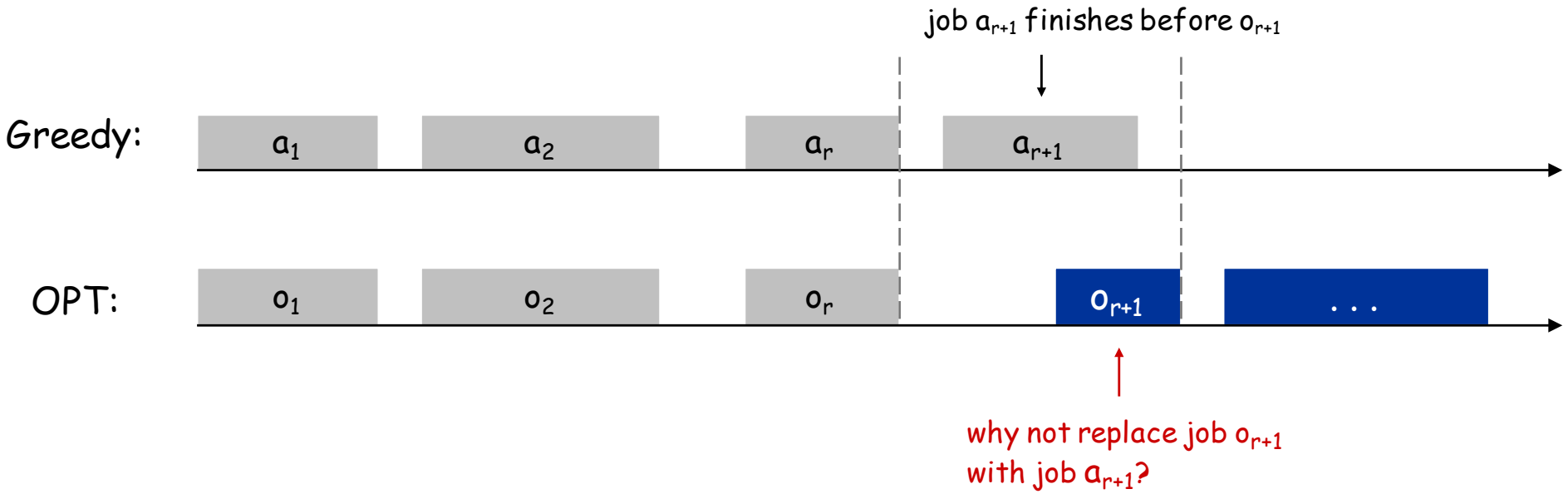
Proof by induction & contradiction in class on paper

Interval Scheduling: Analysis (alternative proof)

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $a_1, a_2, \ldots a_k$ denote set of jobs in A selected by greedy.
- Let $o_1, o_2, \ldots o_m$ denote set of jobs in the optimal solution O with $a_1 = o_1, a_2 = o_2, \ldots, a_r = o_r$ for the largest possible value of r (not any optimal solution).

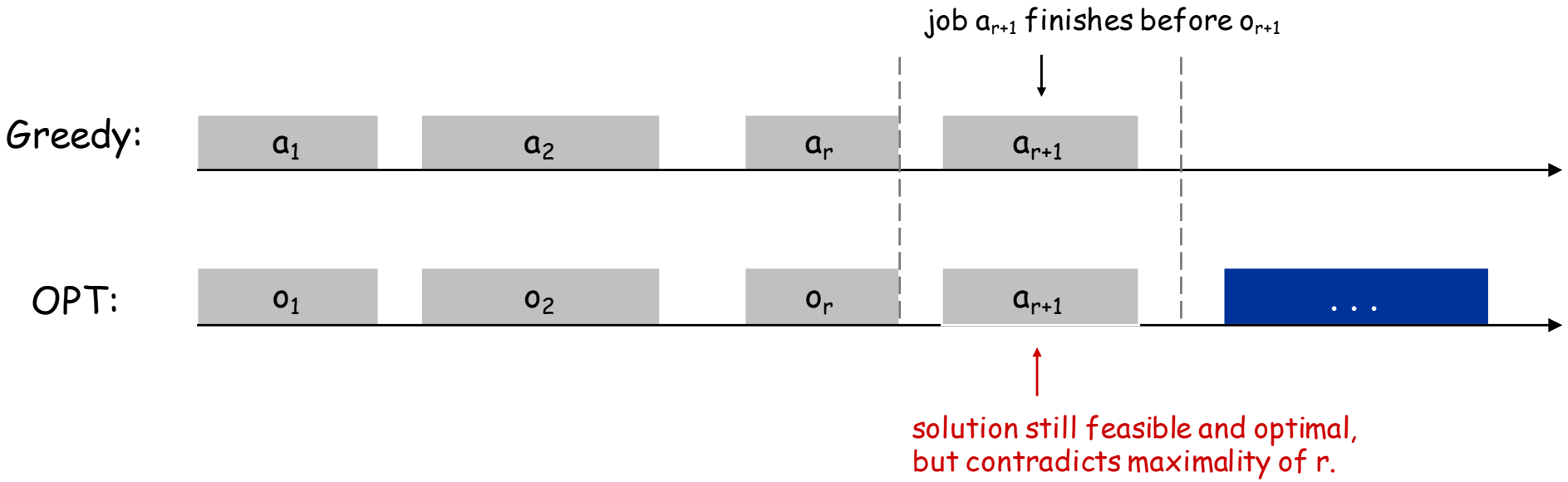


Interval Scheduling: Analysis (alternative proof)

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $a_1, a_2, \ldots a_k$ denote set of jobs in A selected by greedy.
- Let $o_1, o_2, \ldots o_m$ denote set of jobs in the optimal solution O with $a_1 = o_1, a_2 = o_2, \ldots, a_r = o_r$ for the largest possible value of r (not any optimal solution).



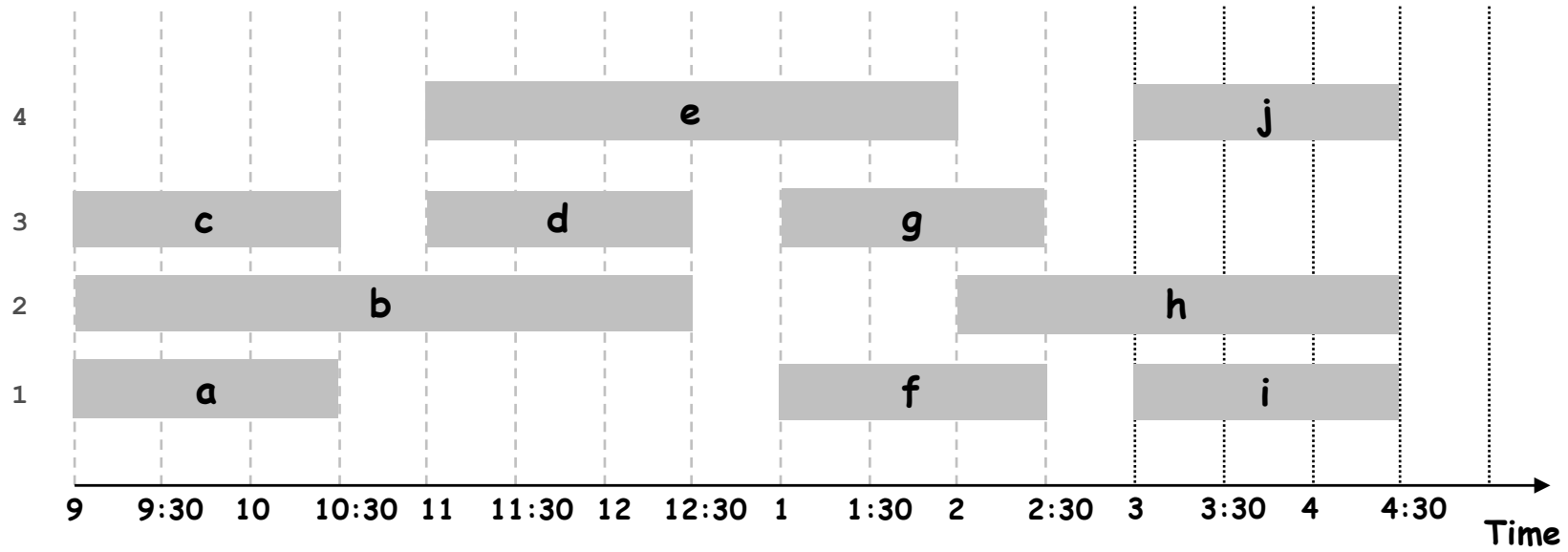
4.1 Interval Partitioning

Interval Partitioning

Interval partitioning.

- Lecture j starts at s_j and finishes at f_j.
- Goal: find 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.

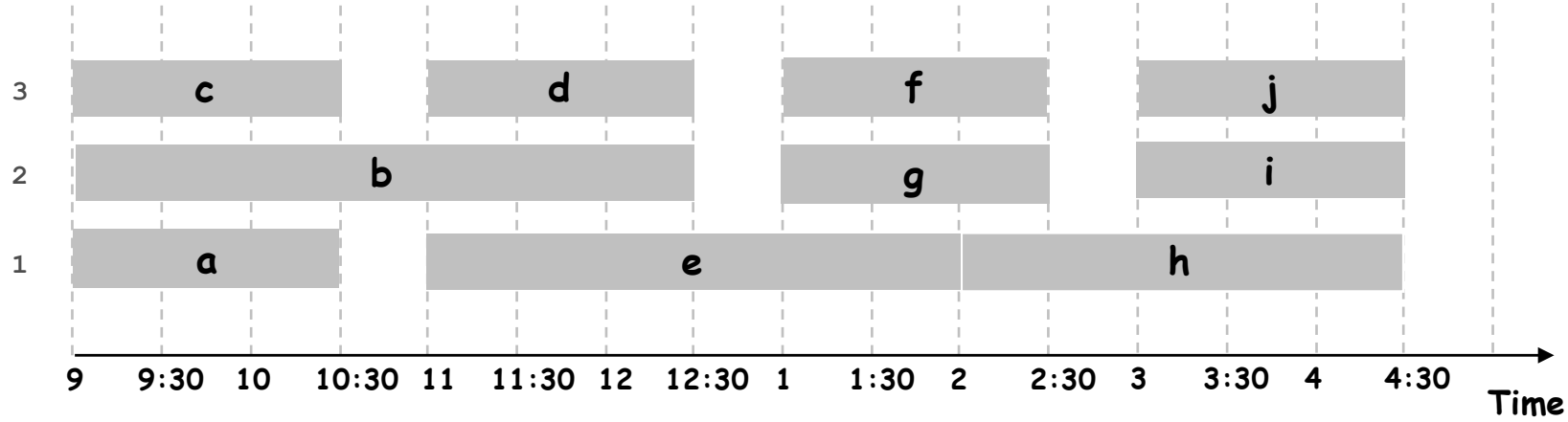


Interval Partitioning

Interval partitioning.

- Lecture j starts at s_j and finishes at f_j.
- Goal: find 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: Lower Bound on Optimal Solution

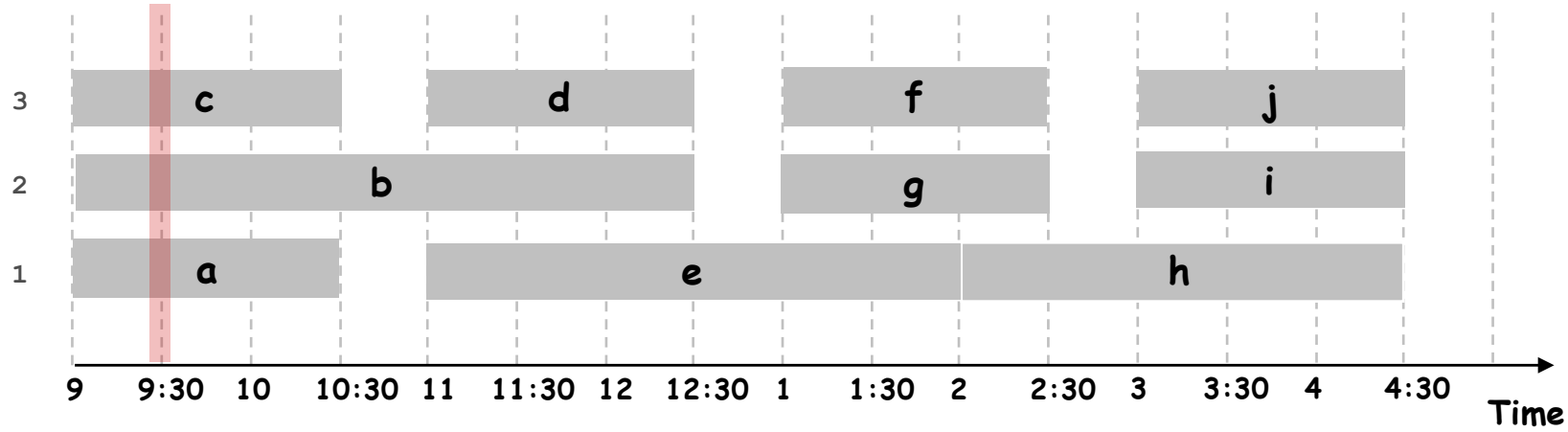
Def. The depth of a set of open intervals is the maximum number that contain any given time.

Key observation. Number of classrooms needed ≥ depth.

Ex: Depth of schedule below = 3 ⇒ schedule below is optimal.

a, b, c all contain 9:30

Q. Does there always exist a schedule equal to depth of intervals?



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 s1 ≤ s2 ≤ ... ≤ sn.
d ← 0   ← number of allocated classrooms

for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

Interval Partitioning: Greedy Analysis

Observation. Greedy algorithm never schedules two incompatible lectures in the same classroom.

Theorem. Greedy algorithm is optimal.

Pf.

- Let d = number of classrooms that the greedy algorithm allocates.
- The last classroom d is opened because we needed to schedule a job, say j, that is incompatible with all d-1 other classrooms.
- Since we sorted by start time, all these incompatibilities are caused by scheduled lectures that start no later than s_j.
- These d-1 jobs each end after s_j.
- Thus, we have d lectures overlapping at time $s_j + \varepsilon$.
- That means that the depth of the given intervals >= d,
- We argued any schedule requires #classrooms >= depth (Key Observation), hence greedy solution must have d >= depth
- Hence greedy solution with d=depth classrooms must be optimal.

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 s1 ≤ s2 ≤ ... ≤ sn.
d ← 0   ← number of allocated classrooms

for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

Implementation.

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 s1 ≤ s2 ≤ ... ≤ sn.


for j = 1 to n {
   k = 1 (resource/classroom)
   while (request j not scheduled) {
        lastk = finish time of the last request currently
scheduled on resource k
        if sj ≥ lastk then schedule request j on resource k
        k = k+1
    }
}
```

Time: May be slow O(nd), which may be $O(n^2)$

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 s1 ≤ s2 ≤ ... ≤ sn.

d ← 1
Schedule request 1 on resource 1
Last1 = f1
Insert 1 into priority queue Q with key = last1
For i=2 to n {
   j = findmin(Q)
   if si≥lastj
      schedule request i on resource j
      lastj = fi
      Increasekey(j,Q)  to lastj
   else
      d = d+1
      schedule request i on resource d
      lastd = fi
      Insert d into priority queue Q with key = lastd
}
```

Time: n calls to Increasekey and findmin, hence O(*n*log*n*)

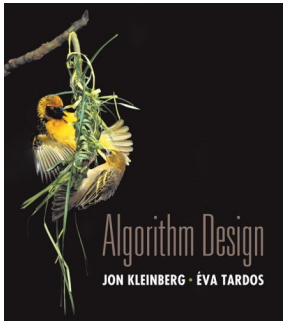
Greedy Analysis Strategies

Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.

Structural. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

Scheduling to Minimize Lateness



Slides by Kevin Wayne.

Scheduling to Minimize Lateness

Minimizing lateness problem.

- Single resource processes one job at a time.
- Job j requires t_j units of processing time and is due at time d_j.
- If j starts at time s_j, it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max \{ 0, f_j - d_j \}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max \ell_j$.

Ex:

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

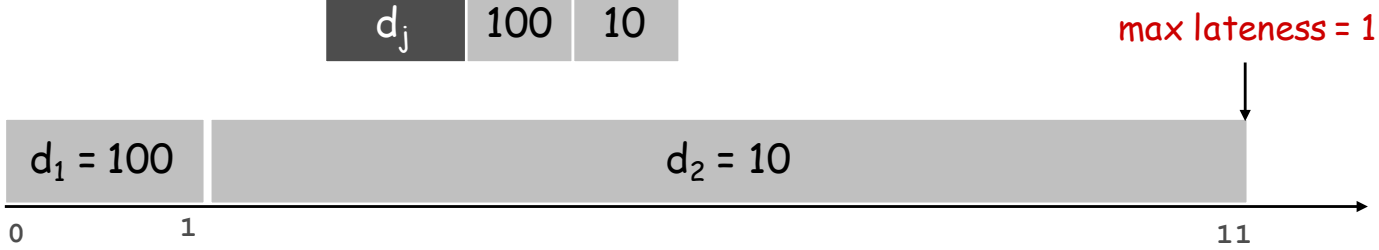
Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time t_j.

	1	2
t_j	1	10
d_j	100	10

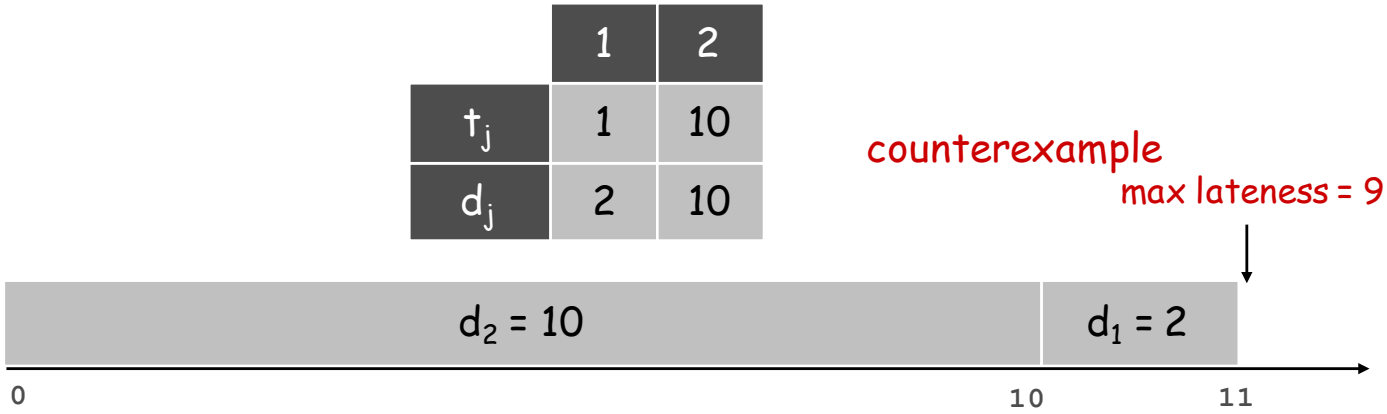
counterexample



- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

	1	2
t_j	1	10
d_j	2	10

counterexample

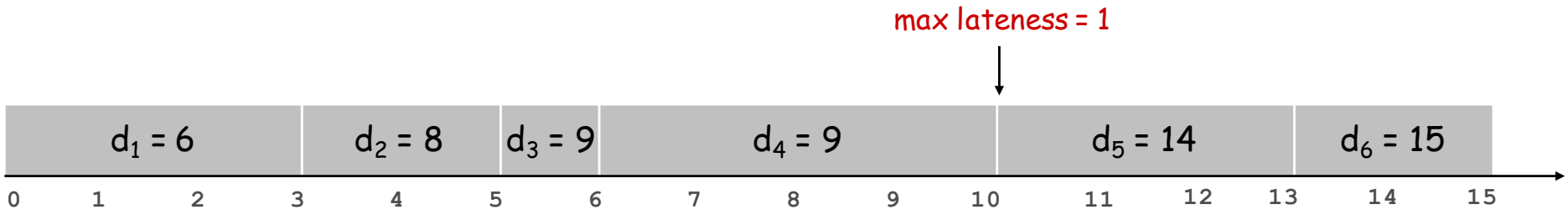


Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

```
Sort n jobs by deadline so that d_1 ≤ d_2 ≤ … ≤ d_n

t ← 0
for j = 1 to n
   Assign job j to interval [t, t + t_j]
   s_j ← t, f_j ← t + t_j
   t ← t + t_j
output intervals [s_j, f_j]
```



Is this algorithm optimal or can you find a counter-example?