Chapter 4. Greedy Algorithms

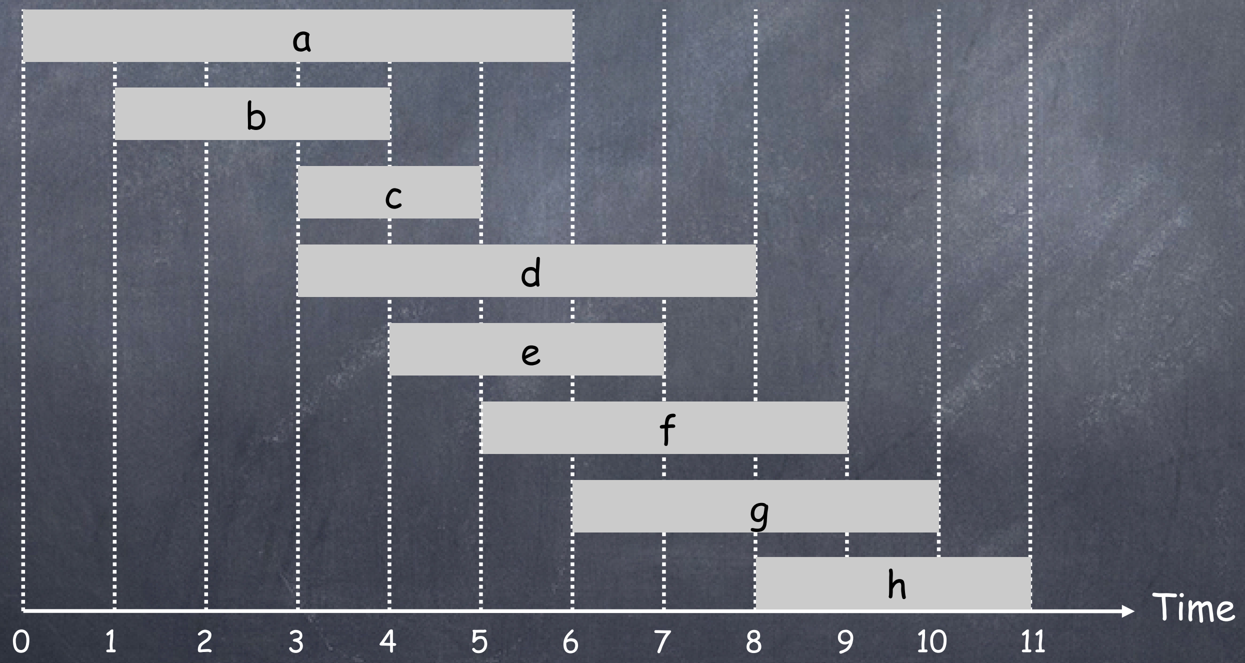
- Interval scheduling
- Greedy overview
- Shortest paths
- Minimum spanning trees

What is a greedy algorithm?

- Hard to describe, but I know it when I see it!

Interval Scheduling

- Schedule n jobs: j^{th} job has start time s_j, finish time f_j.
- Two jobs **compatible** if they don't overlap.
- Goal: find maximum size subset of mutually compatible jobs.



Greedy Template

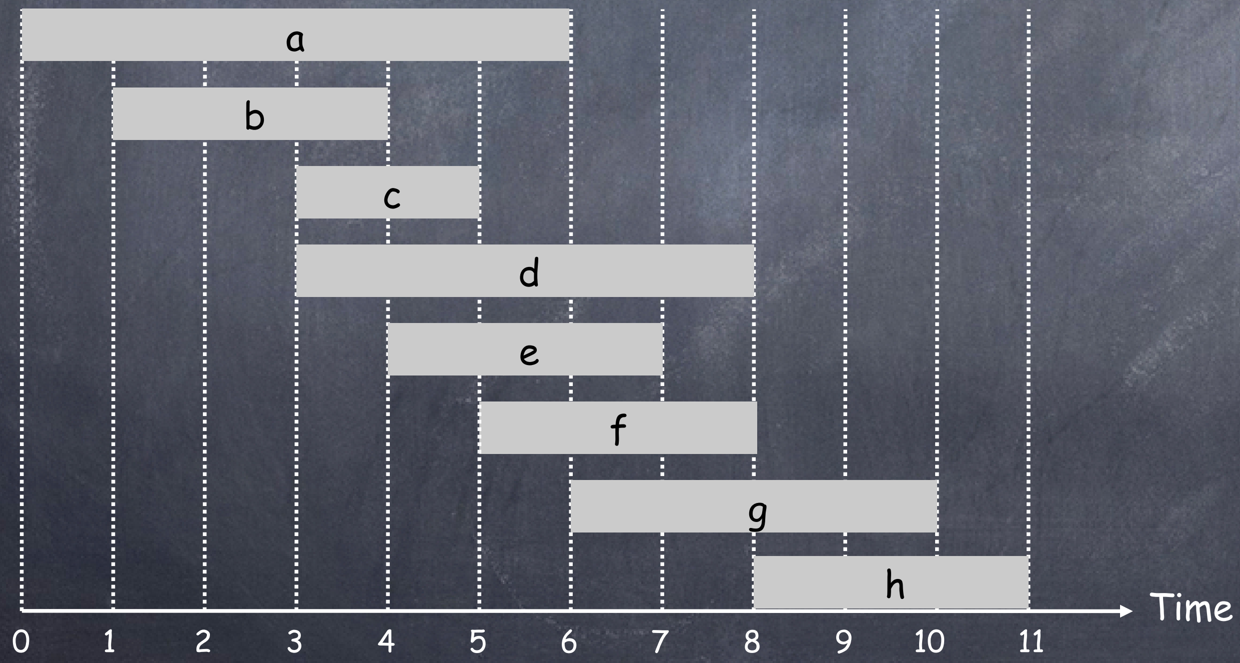
```
A ← {}
while (there are jobs compatible with A)
   pick "best" compatible job j
      A = A ∪ {j}
}
return A
```

Greedy: pick j and never look back

What rule to use?

Interval Scheduling: Greedy Solution

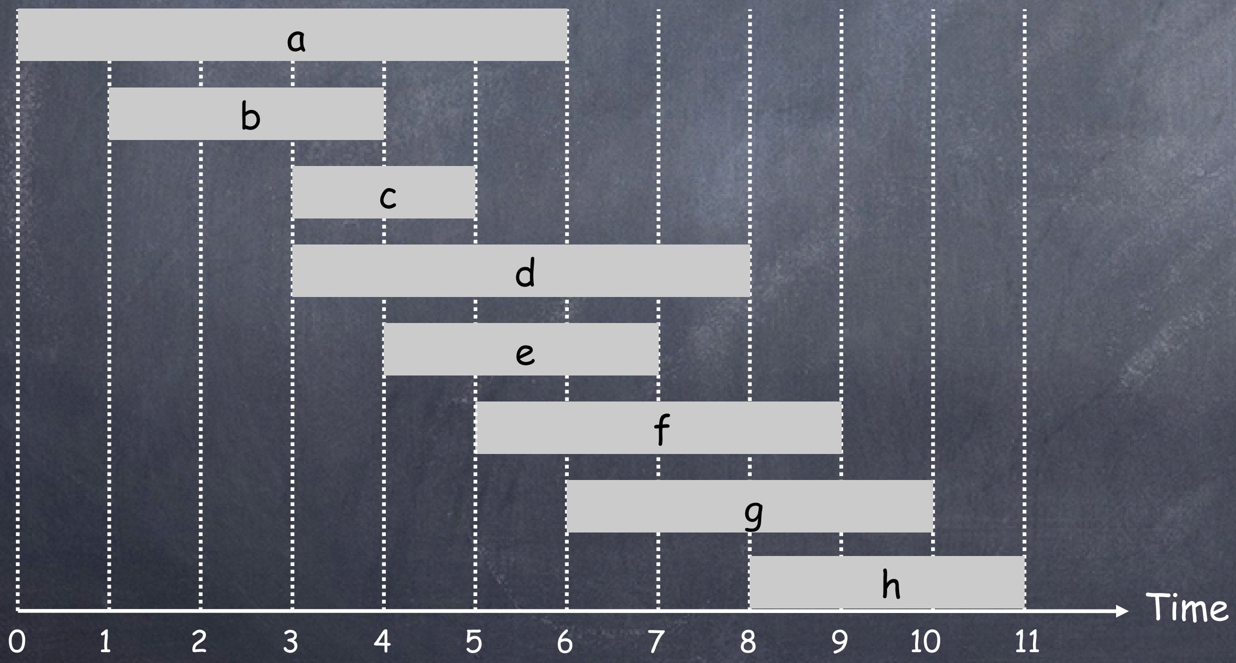
- Idea 1: Earliest start time. Consider jobs in ascending order of start time s_j.



a, g

Interval Scheduling: Greedy Solution

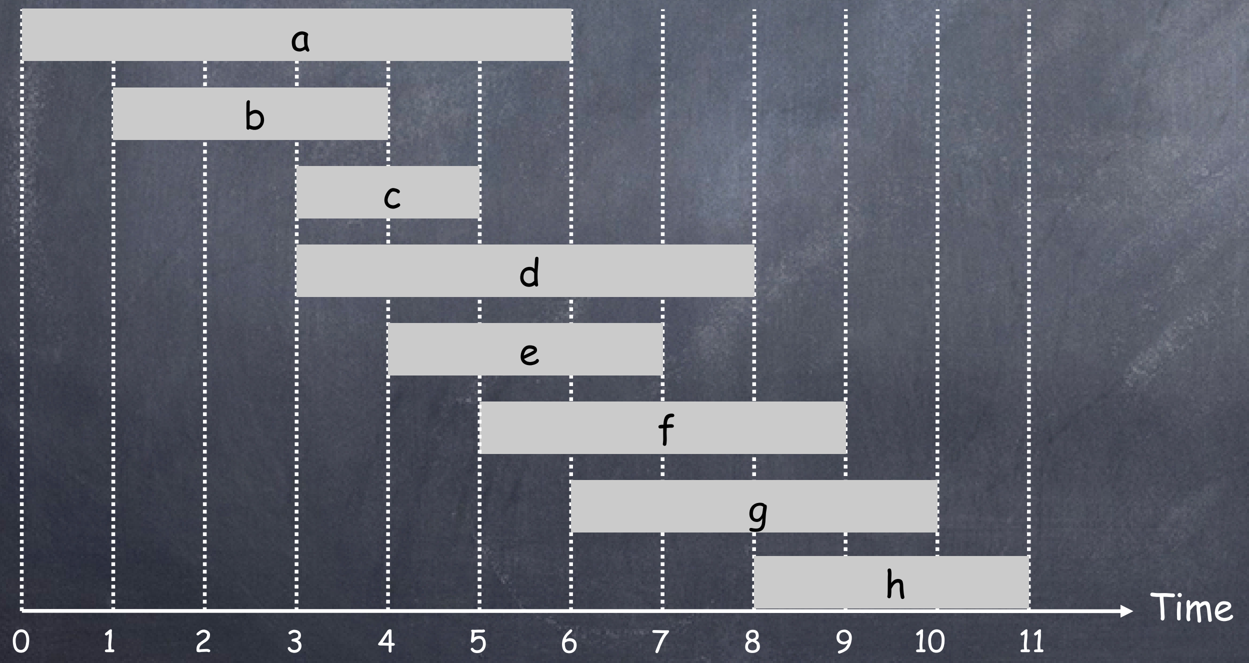
- Idea 2: Shortest interval. Consider jobs in ascending order of interval length $f_j - s_j$.



c, h

Interval Scheduling: Greedy Solution

- Idea 3: Fewest conflicts. For each job, count the number of conflicting jobs c_j. Schedule in ascending order of conflicts c_j.



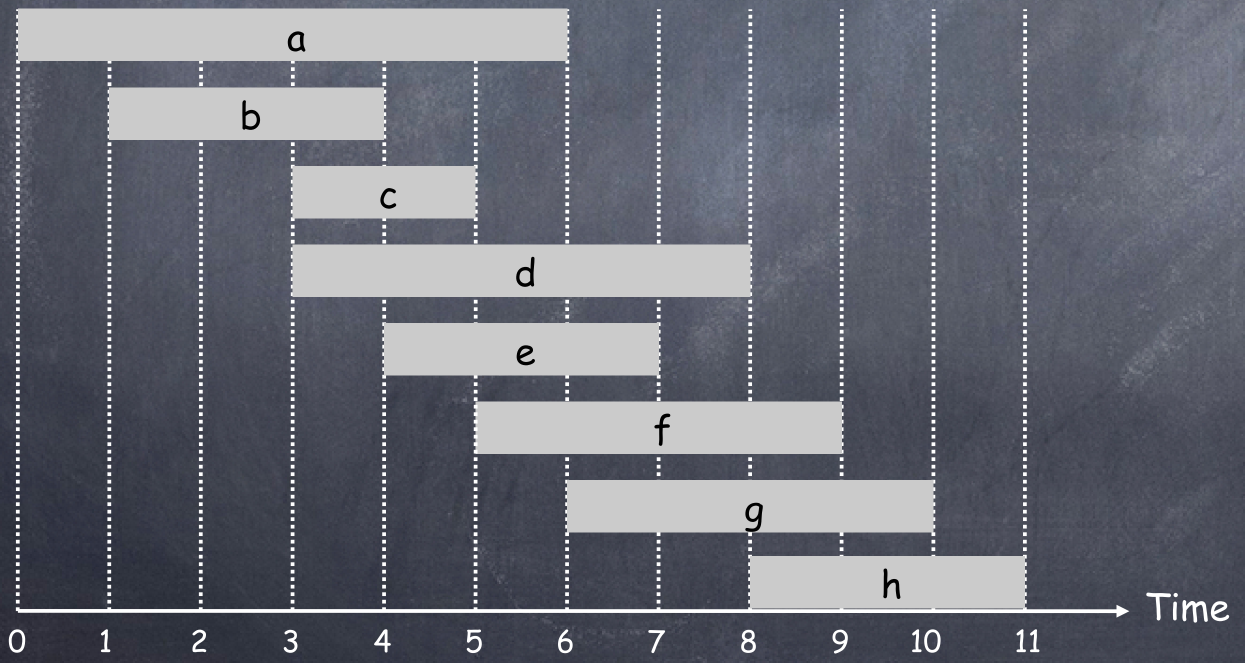
Interval Scheduling: Greedy Solution

- Idea 3: Fewest conflicts. For each job, count the number of conflicting jobs c_j. Schedule in ascending order of conflicts c_j.

h, b, e

Interval Scheduling: Greedy Solution

- Idea 4: Earliest finish time. Consider jobs in ascending order of finish time f_j.



b, e, h

Earliest Finish Time - Optimal Solution

Sort jobs by finish times so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

```
A ← {}
for j = 1 to n {
   if (job j compatible with A)
      A = A ∪ {j}
}
return A
```

Proof and running time on board

Greedy Overview

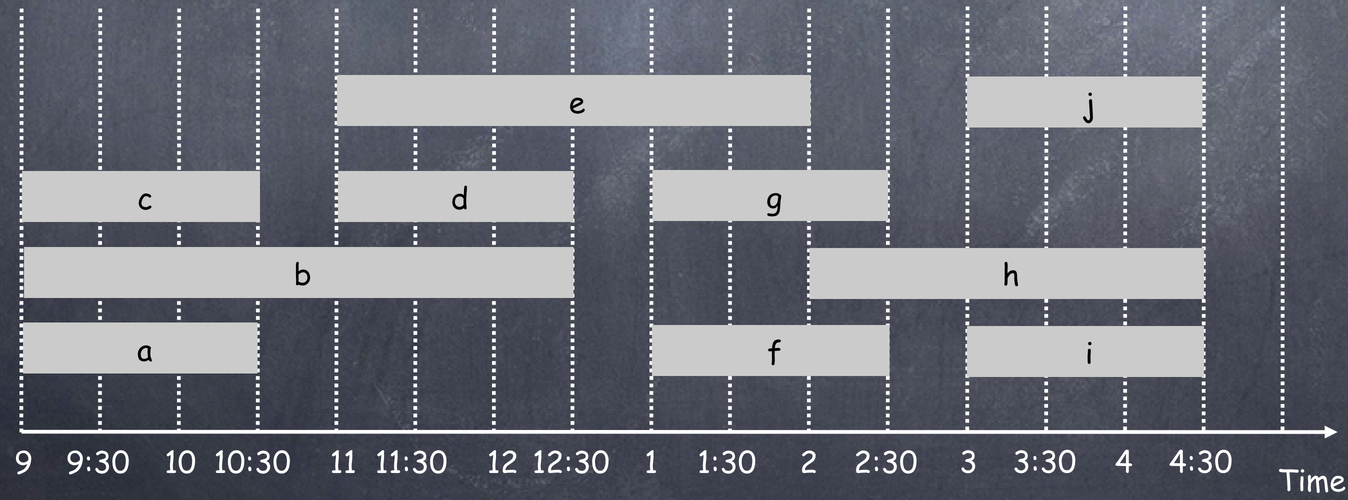
- Build up solution by adding items one at a time
- Choose next item by simple heuristic, never remove items
- Prove that the result is optimal!

- Simple algorithm -> hard part is proving it correct
- Running time usually n log n or worse: need to sort items

Interval Partitioning

Lecture j starts at s_j and finishes at f_j.

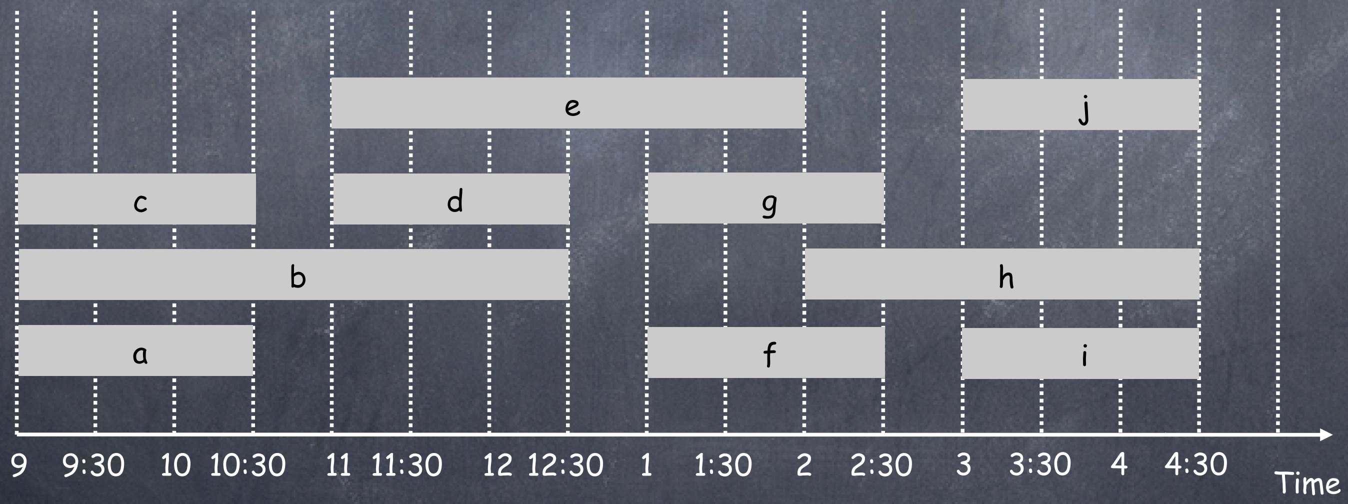
Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.



Interval Partitioning Lower Bound

The depth of a set of intervals is the maximum number that contain any point in time-line.

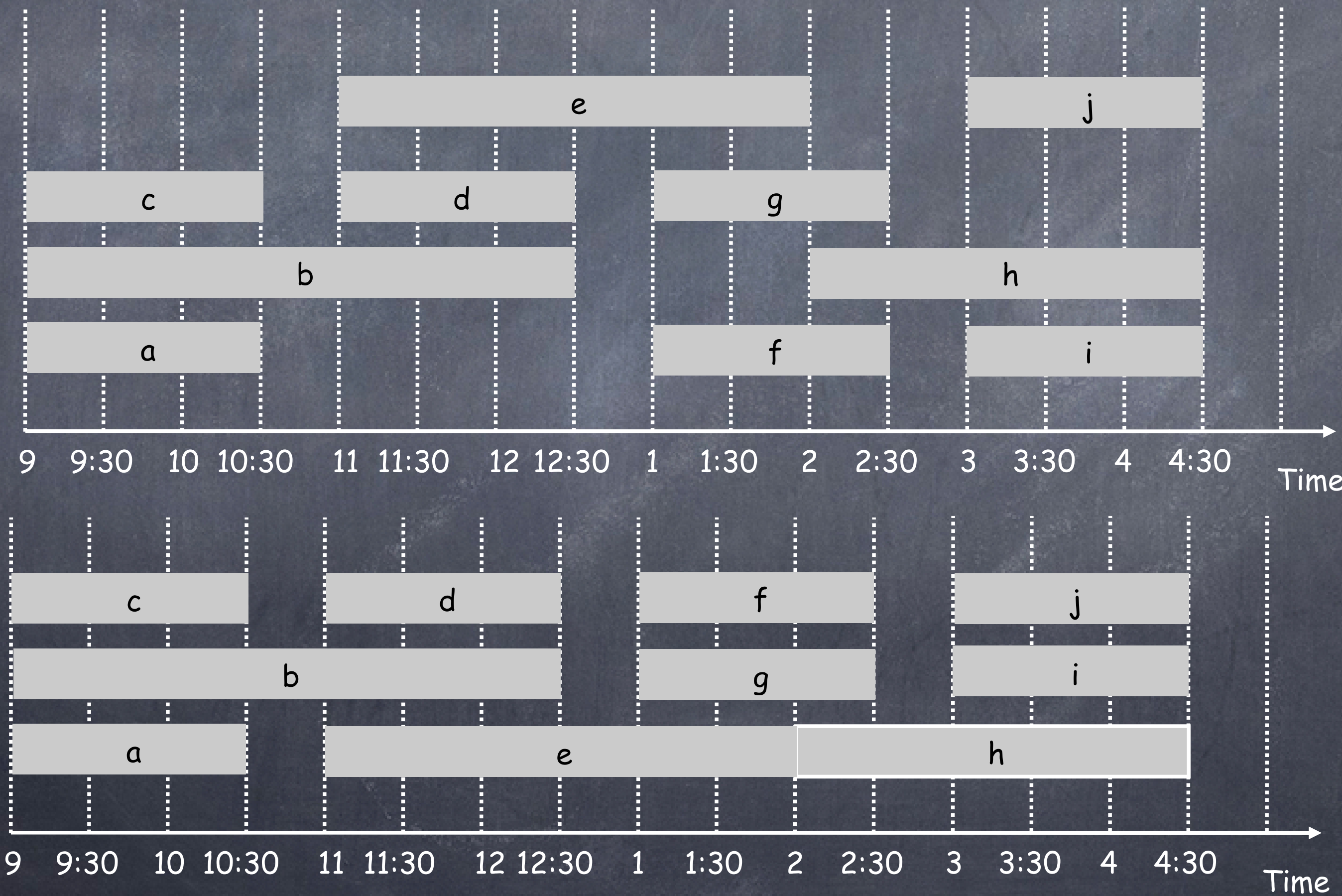
Key observation. Number of classrooms needed $\geq$ depth.



Interval Partitioning Lower Bound

Example: Depth of schedule below = 3

Question: Does there always exist a schedule equal to depth of intervals?



Idea

- Number classrooms 1, 2, 3, …
- Sort intervals in some order: for each interval, assign it to first available classroom
- What order?

Interval Partitioning: Greedy Solution

Sort intervals by starting time so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

```
k ← 0    // Number of classrooms

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

Complexity?

Scheduling to Minimize Lateness

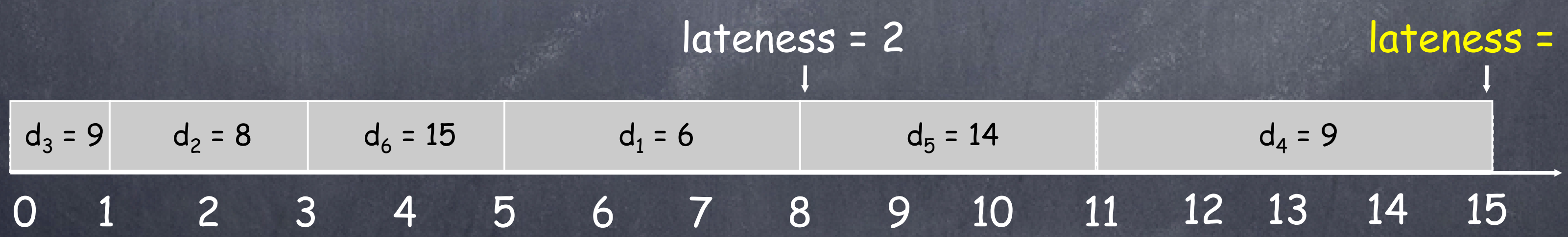
- Single computer processes one job at a time.
- Jobs $i = 1,2,\ldots,n$:
  - Processing time t_i
  - Deadline d_i
  - Start time s_i -> finish time $f_i = s_i + t_i$.
- Lateness: $l_i = \max_{1 \le i \le n} \{ 0, f_i - d_i \}$.

- Goal: schedule start times of all jobs to minimize maximum lateness $L = \max l_i$.

Scheduling Example

		Job 1	2	3	4	5	6
Processing time	t_i	3	2	1	4	3	2
Deadline	d_i	6	8	9	9	14	15

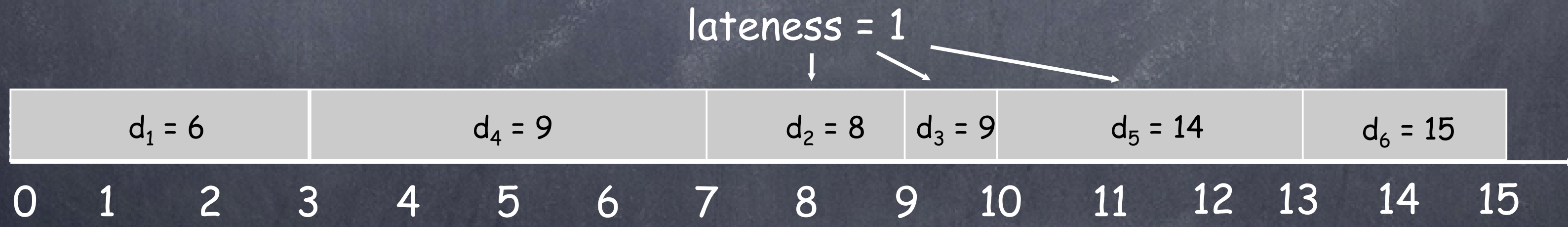
Attempt 1: Sort by t



Max lateness: 6

Scheduling Example: Smallest Slack time first

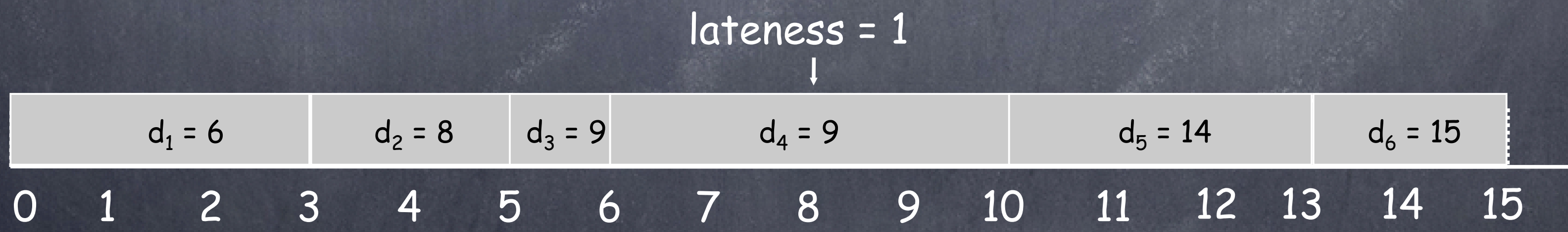
	1	2	3	4	5	6
t_i	3	2	1	4	3	2
d_i	6	8	9	9	14	15
$slack_i$	3	6	8	5	11	13



Max lateness: 1

Scheduling Example: Earliest Deadline First

	1	2	3	4	5	6
t_i	3	2	1	4	3	2
d_i	6	8	9	9	14	15



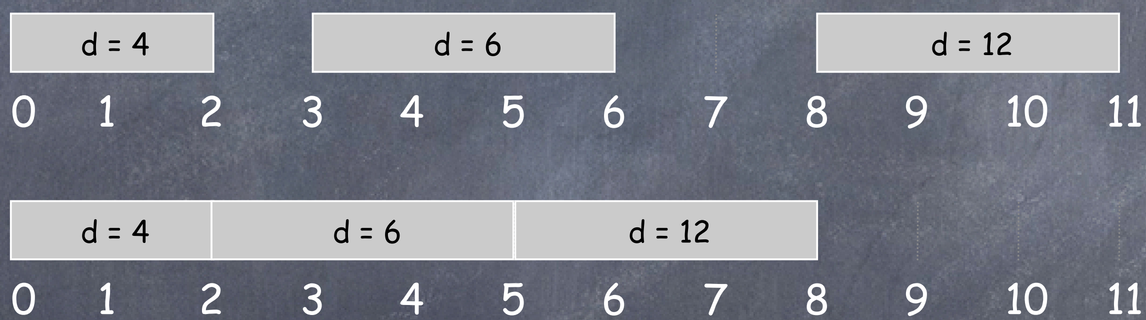
Max lateness: 1

Minimizing Lateness: Analysis

- Claim: scheduling jobs by their deadline is optimal
- Let's establish some basic facts for the proof...

Minimizing Lateness: No Idle Time

Observation. There exists an optimal schedule with no idle time.



Observation. The greedy schedule has no idle time.

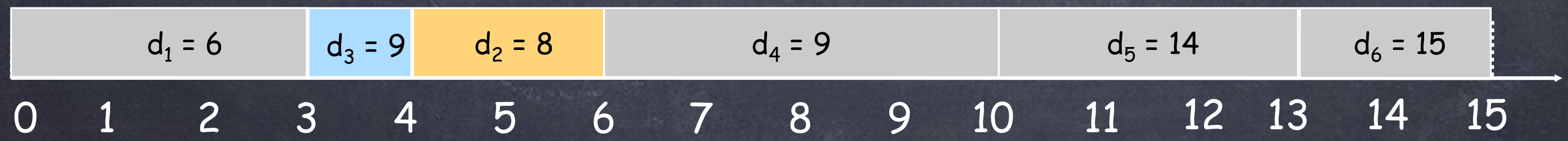
Minimizing Lateness: Proof Approach

- Idea: start with an optimal solution with no idle time, and gradually transform it into the greedy solution (*), without increasing the maximum lateness
- Discuss and outline on board

Minimizing Lateness: Inversions

An inversion in schedule S is a pair of jobs i and j such that i is scheduled before j but $d_j < d_i$.

	1	2	3	4	5	6
t_i	3	2	1	4	3	2
d_i	6	8	9	9	14	15



Minimizing Lateness: Inversions

- Goal: modify optimal solution to eliminate inversions to match greedy solution. But: this might not give exactly the greedy solution.
- Lemma A: all solutions with no idle time and no inversions have same maximum lateness
- Proof on board

Minimizing Lateness: Proof!

- Theorem: the greedy solution is optimal
- Proof on board

Proof Strategies for Greedy Algorithms

- Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as an optimal solution.
- Exchange argument. Gradually transform an optimal solution to the one found by the greedy algorithm(*) without hurting its quality.

(*) Or one just like it