Various algorithm design paradigms:

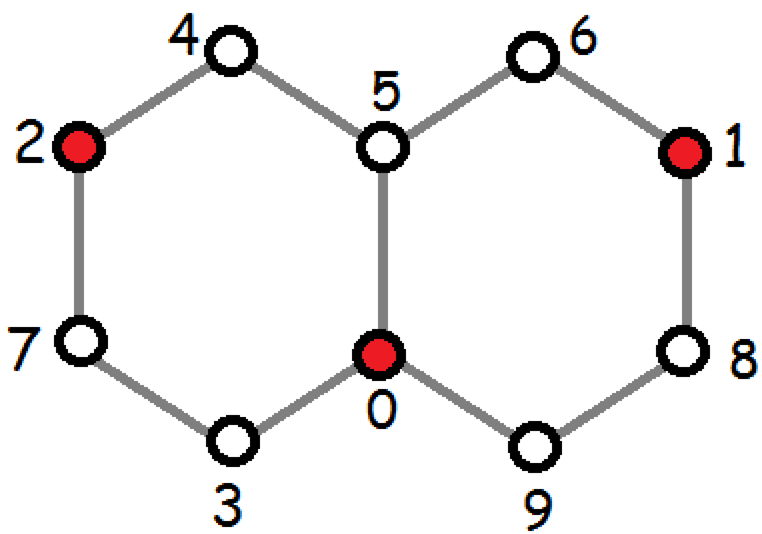
Greedy
Divide and Conquer
Dynamic Programming
Flows
Tactics for dealing with hard problems
Backtracking
Approximation algorithms
Local Search
Randomized algorithms
Heuristics

Greedy Algorithms:
They make decisions that look to be "good ones" at the current time and don't revise those.

Sometimes greedy algorithms provide optimal solutions!

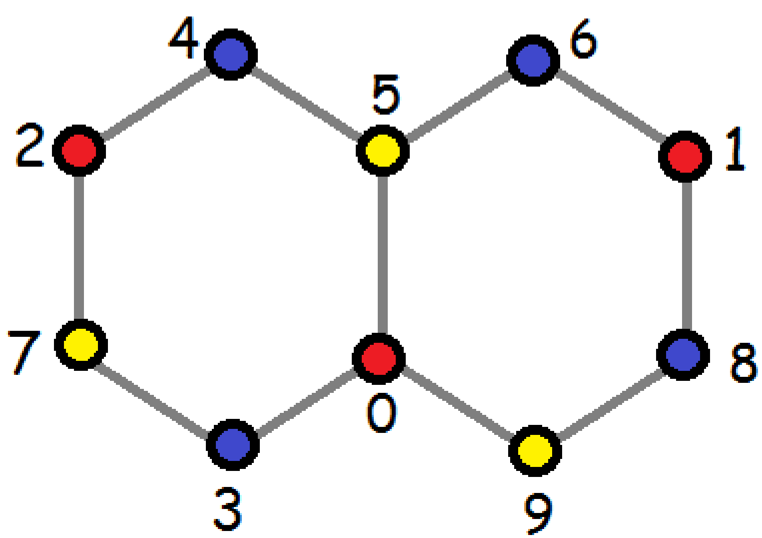
Other times they can provide potentially good but not optimal solutions.

Sometimes the answers they give are not very good.

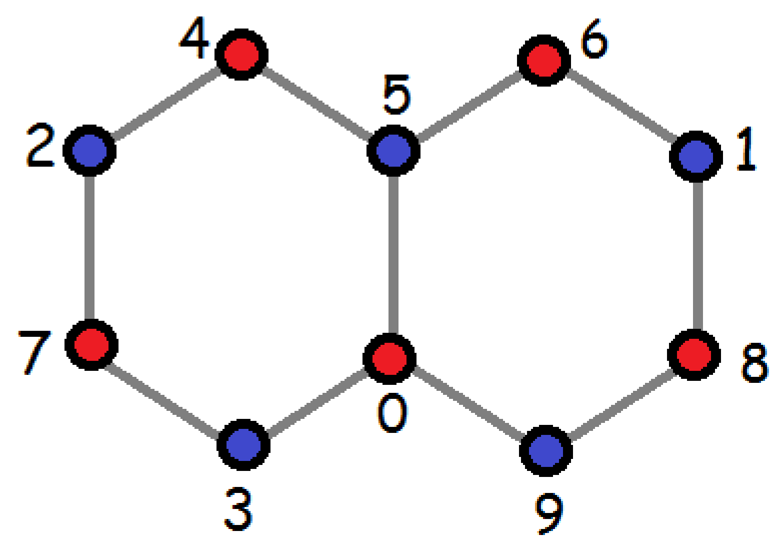


Greedy coloring:

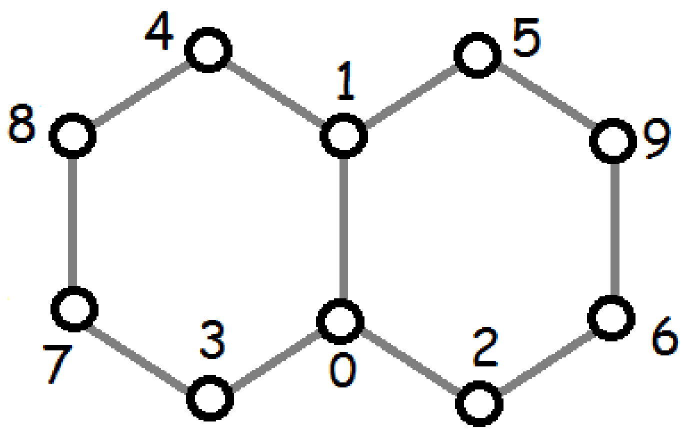
1. Order the vertices.
2. Order the colors.
3. For each vertex, color it with the first available color not already used on one of its neighbours.

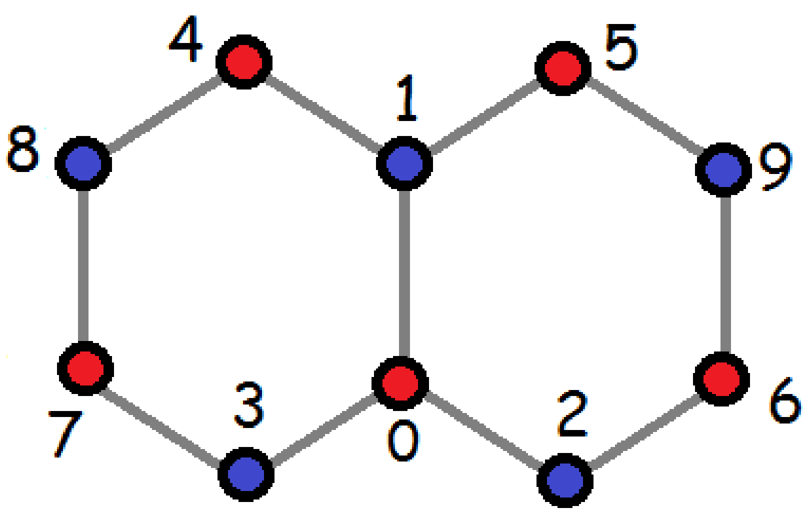


Greedy coloring



Optimal coloring





If you order vertices of a bipartite graph using the BFI then the greedy coloring algorithm will yield an optimal coloring.

What greedy approaches might give a good solution for dominating set?
How would you order the vertices?

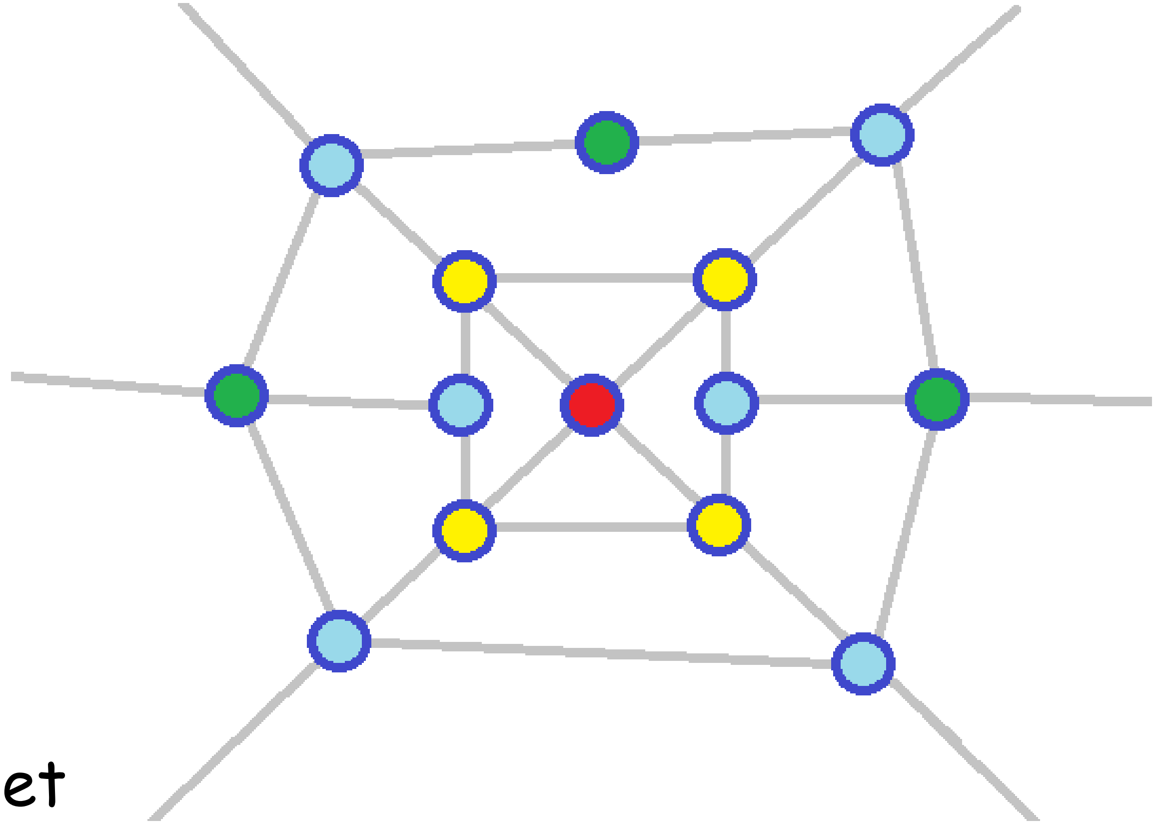
Note: a quick approach to finding a really good dominating set would speed up our backtracking algorithms. The algorithm I gave you has to do more work when it does not have a small dominating set yet.

Some things that would likely help:

- Add vertices of large degree.

- Add vertices that dominate a maximum number of undominated vertices.

- Strive for a perfect dominating set (each vertex is dominated exactly once).



Red: in dominating set
Yellow: distance 1 from dominating set vertex
Teal : distance 2 from dominating set vertex
Green : distance 3 from dominating set vertex

I would probably want to give preference to the green ones over the teals as they help to avoid double domination of the yellow vertices.

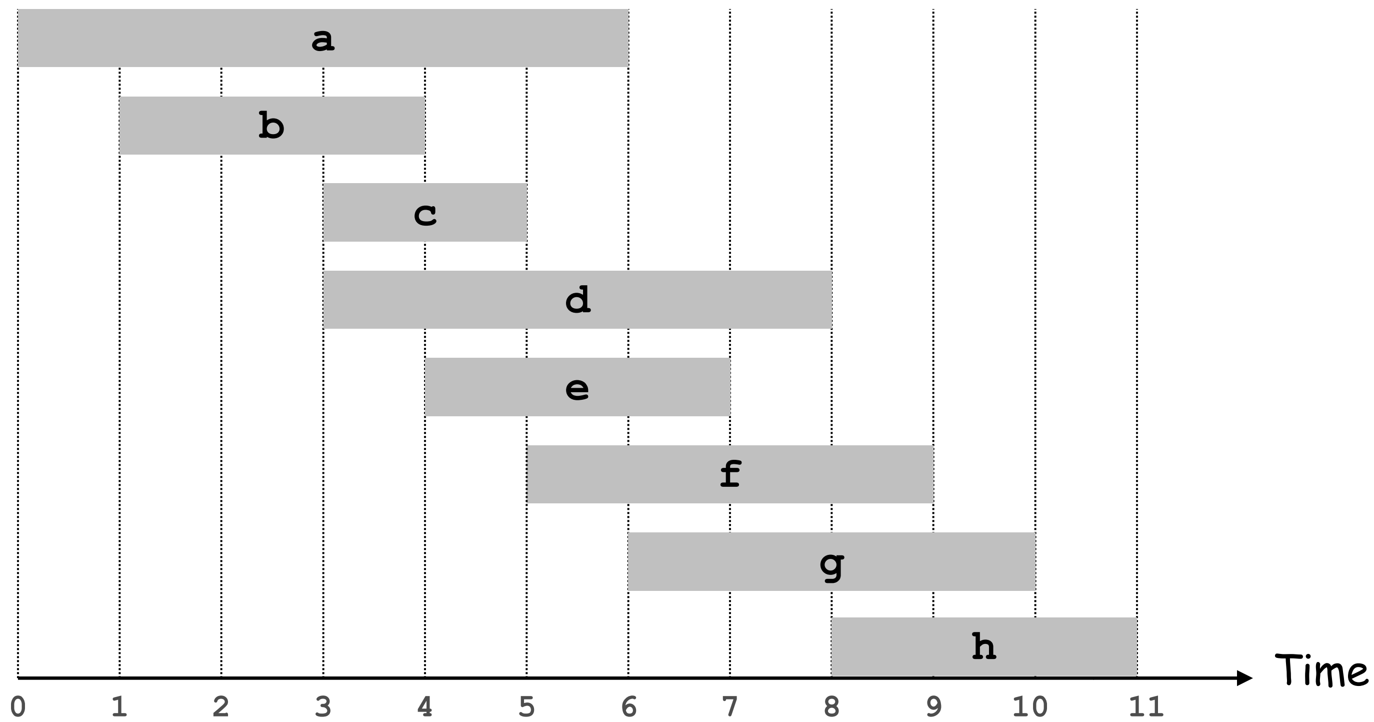
4.1 Interval Scheduling

Slides by Kevin Wayne.

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 Algorithms

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

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

Interval Scheduling: Greedy Algorithms

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

counterexample for earliest start time

counterexample for shortest interval

counterexample for 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.

set of jobs selected

```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

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

Implementation. $O(n \log n)$.

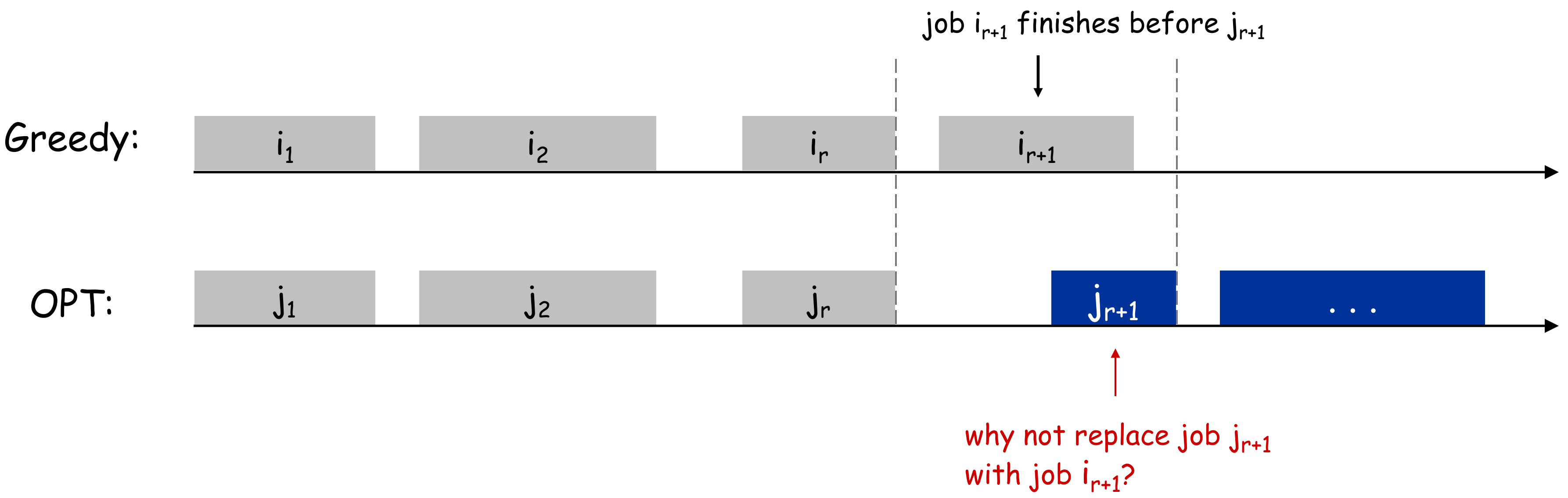
- Remember job j^* that was added last to A.
- Job j is compatible with A if $s_j \geq f_{j^*}$.

Interval Scheduling: Analysis

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots i_k$ denote 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.

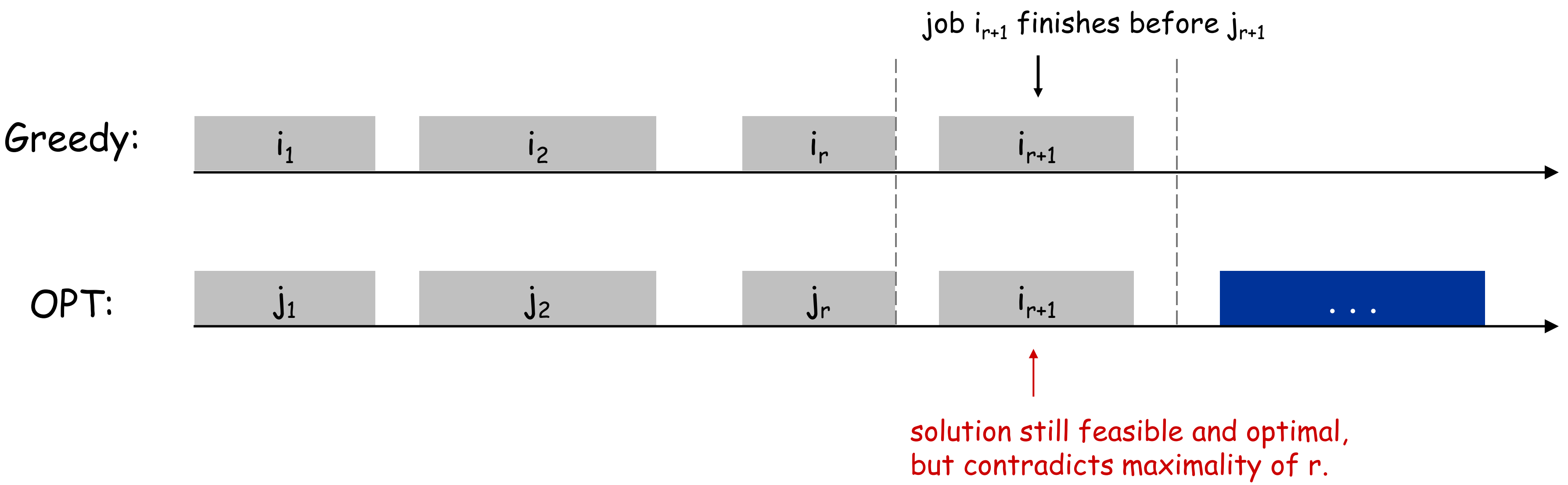


Interval Scheduling: Analysis

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots i_k$ denote 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.



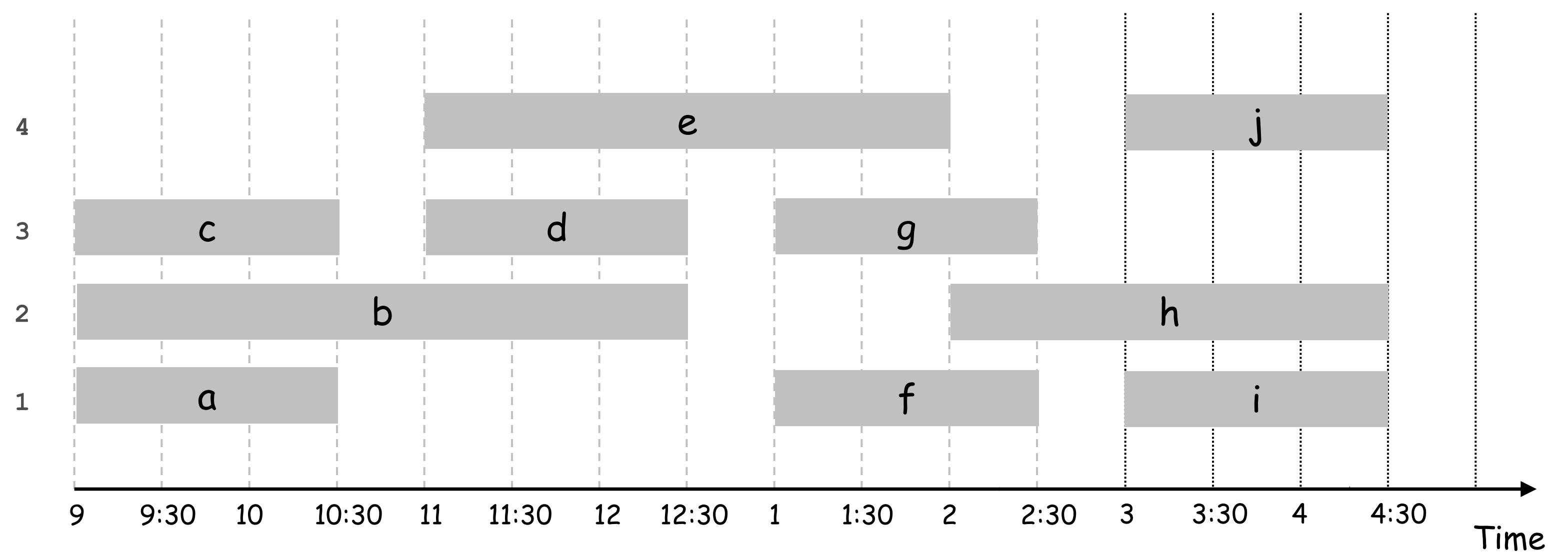
4.1 Interval Partitioning

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.

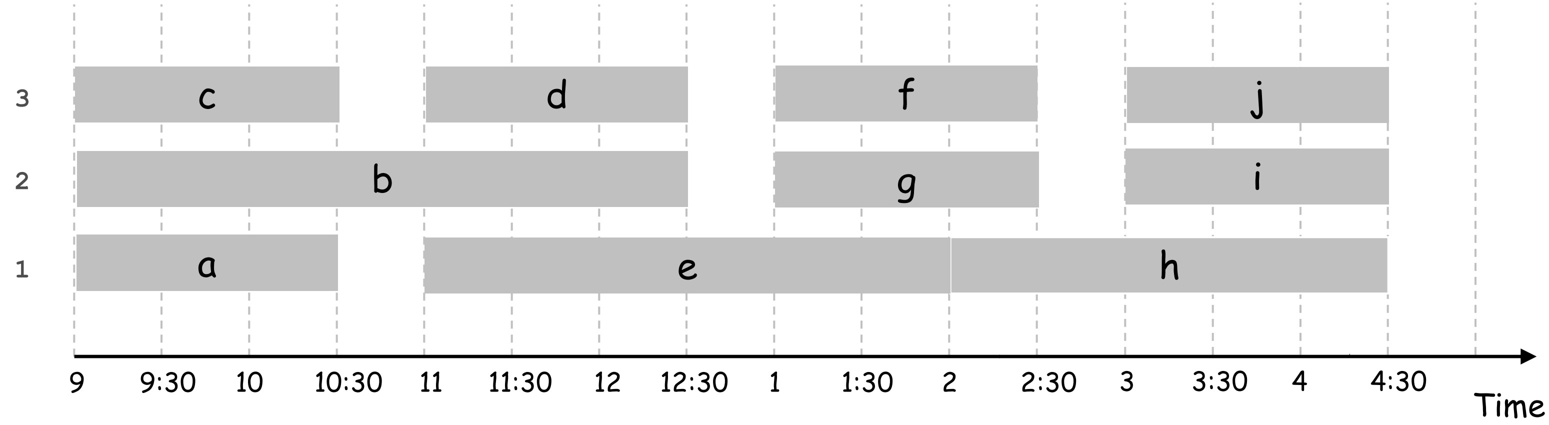


Interval Partitioning

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

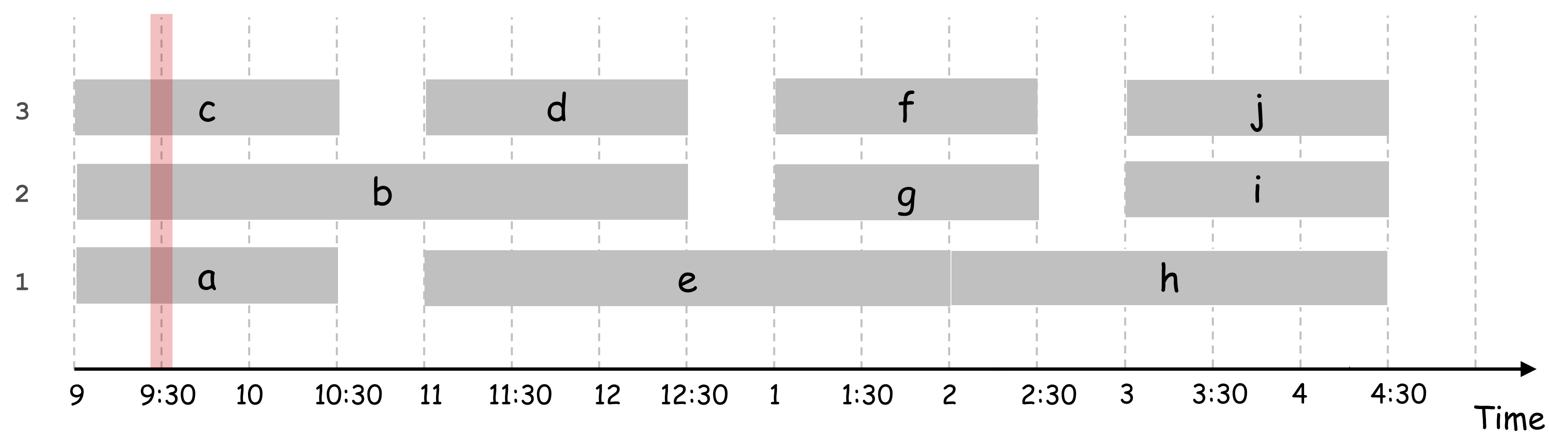
Def. The depth of a set of open intervals is the maximum number that contain any given time.

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

Ex: Depth of schedule below = 3 $\Rightarrow$ 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
}
```

Implementation. $O(n \log n)$.

- For each classroom k, maintain the finish time of the last job added.
- Keep the classrooms in a priority queue.

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.
- Classroom d is opened because we needed to schedule a job, say j, that is incompatible with all d-1 other classrooms.
- These d jobs each end after s_j.
- 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 $\Rightarrow$ all schedules use $\geq$ d classrooms. ▪

4.2 Scheduling to Minimize Lateness

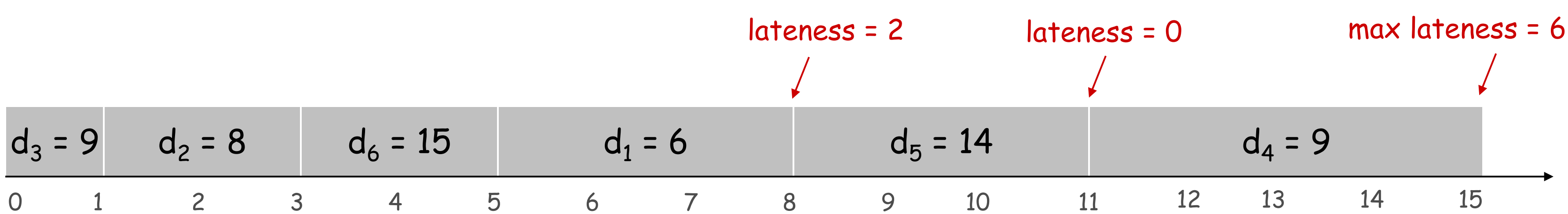
Scheduling to Minimizing Lateness

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.

- [Earliest deadline first] Consider jobs in ascending order of deadline d_j.

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

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

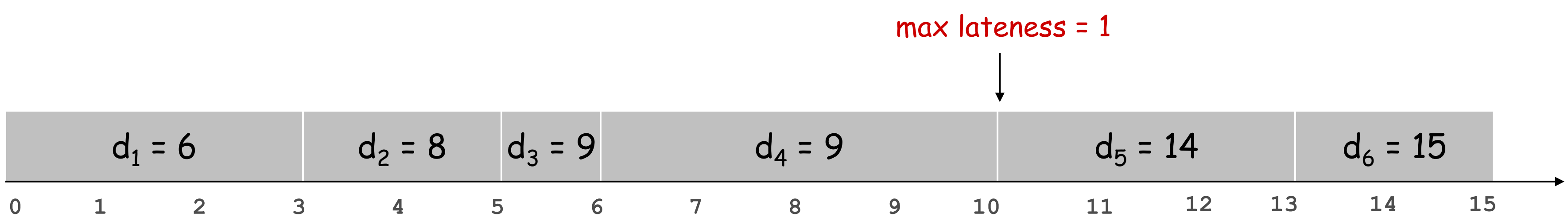
counterexample

Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

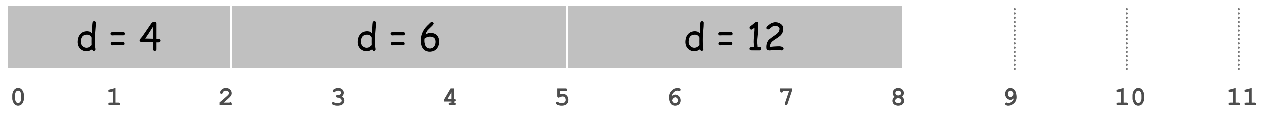
```
Sort n jobs by deadline so that d_1 ≤ d_2 ≤ … ≤ d_n

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]
```



Minimizing Lateness: No Idle Time

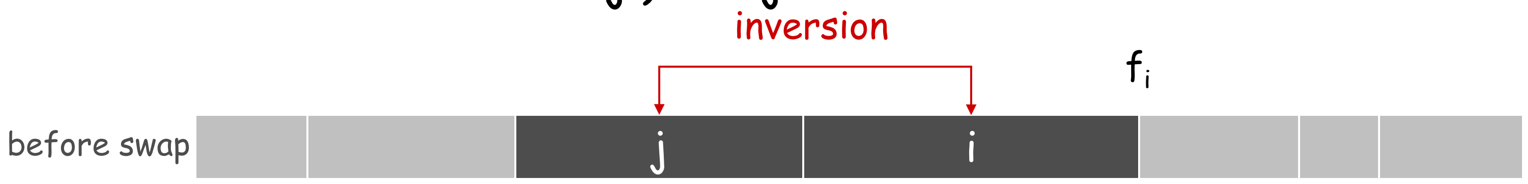
Observation. There exists an optimal schedule with no idle time.



Observation. The greedy schedule has no idle time.

Minimizing Lateness: Inversions

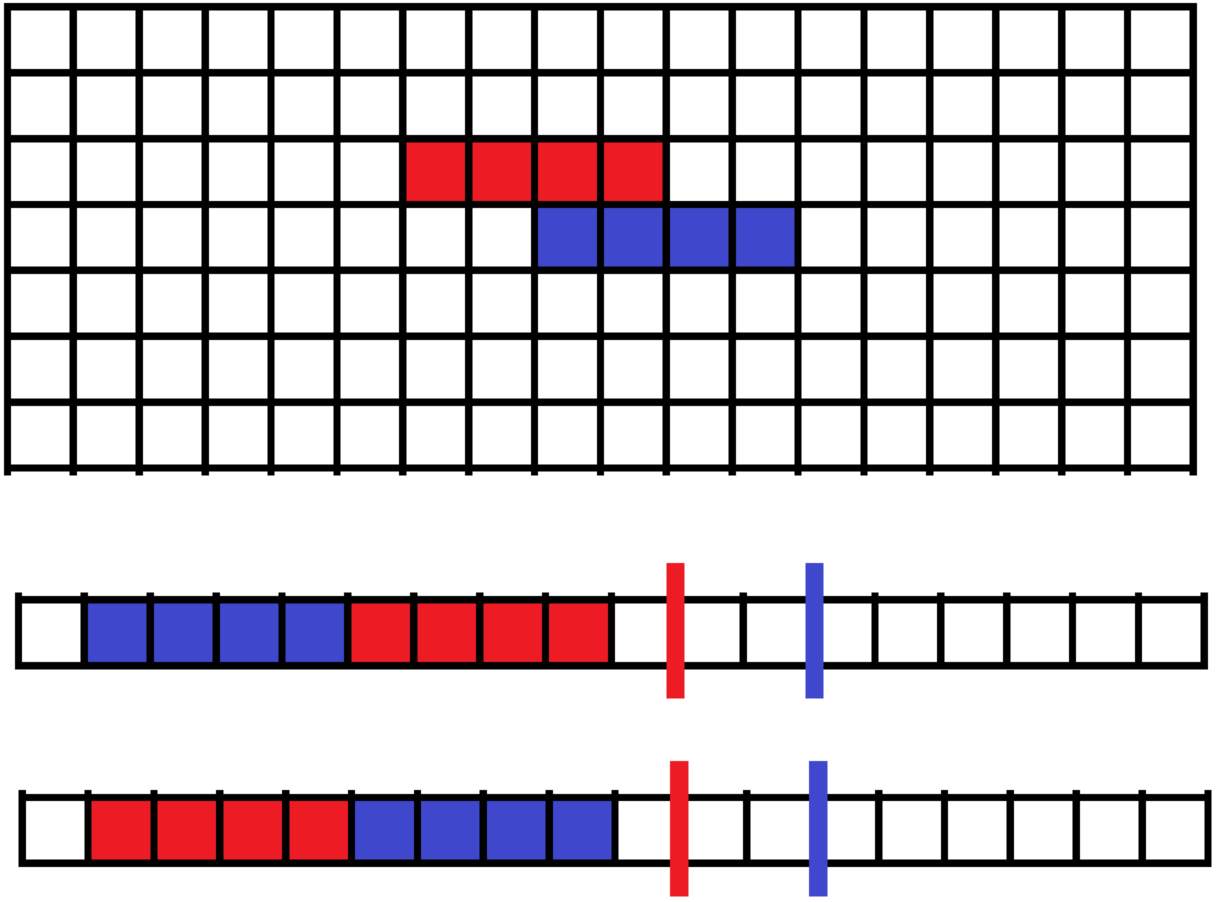
Def. Given a schedule S, an inversion is a pair of jobs i and j such that: $i < j$ (deadline for $i <$ deadline for j) but j scheduled before i.

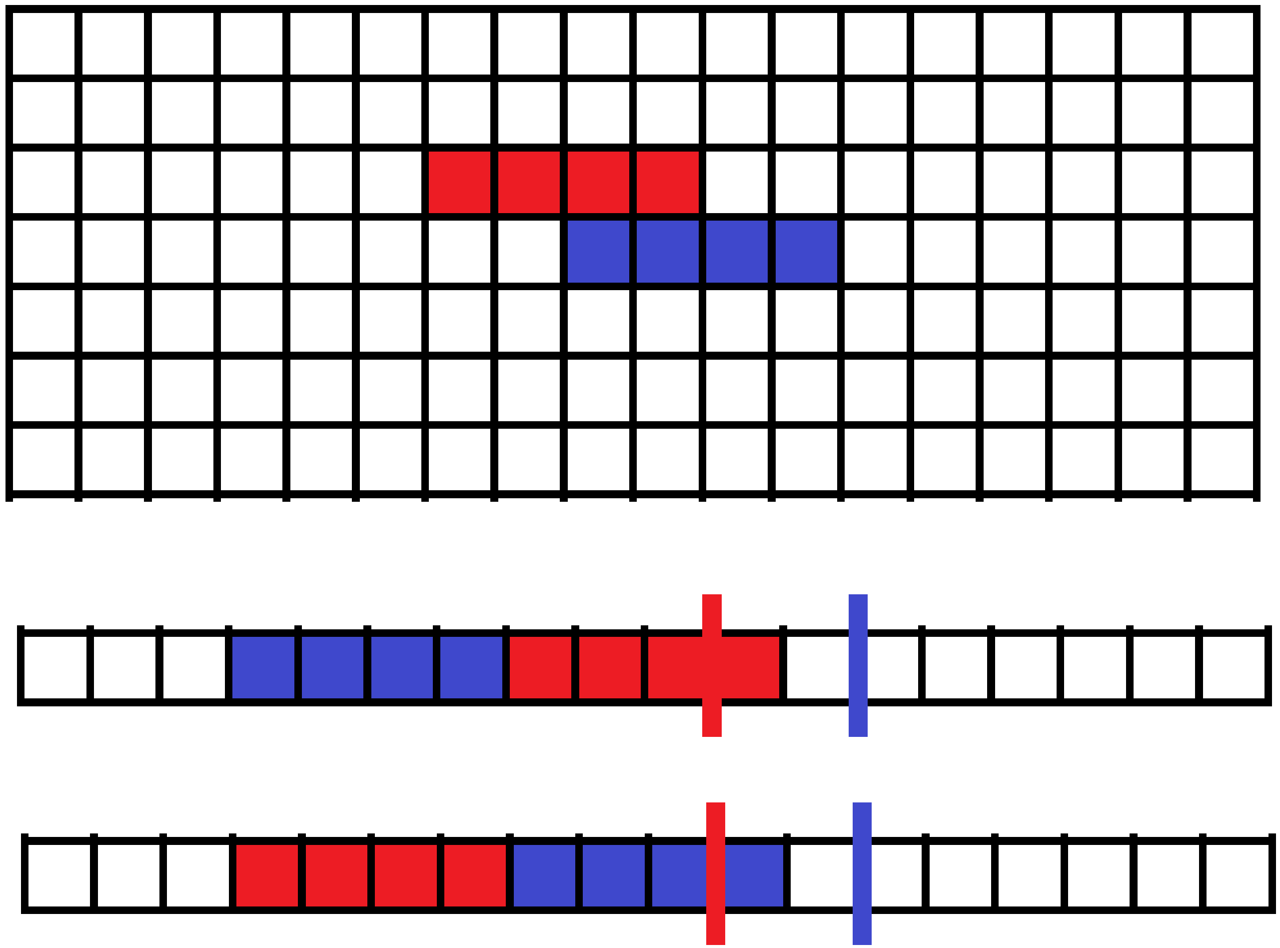


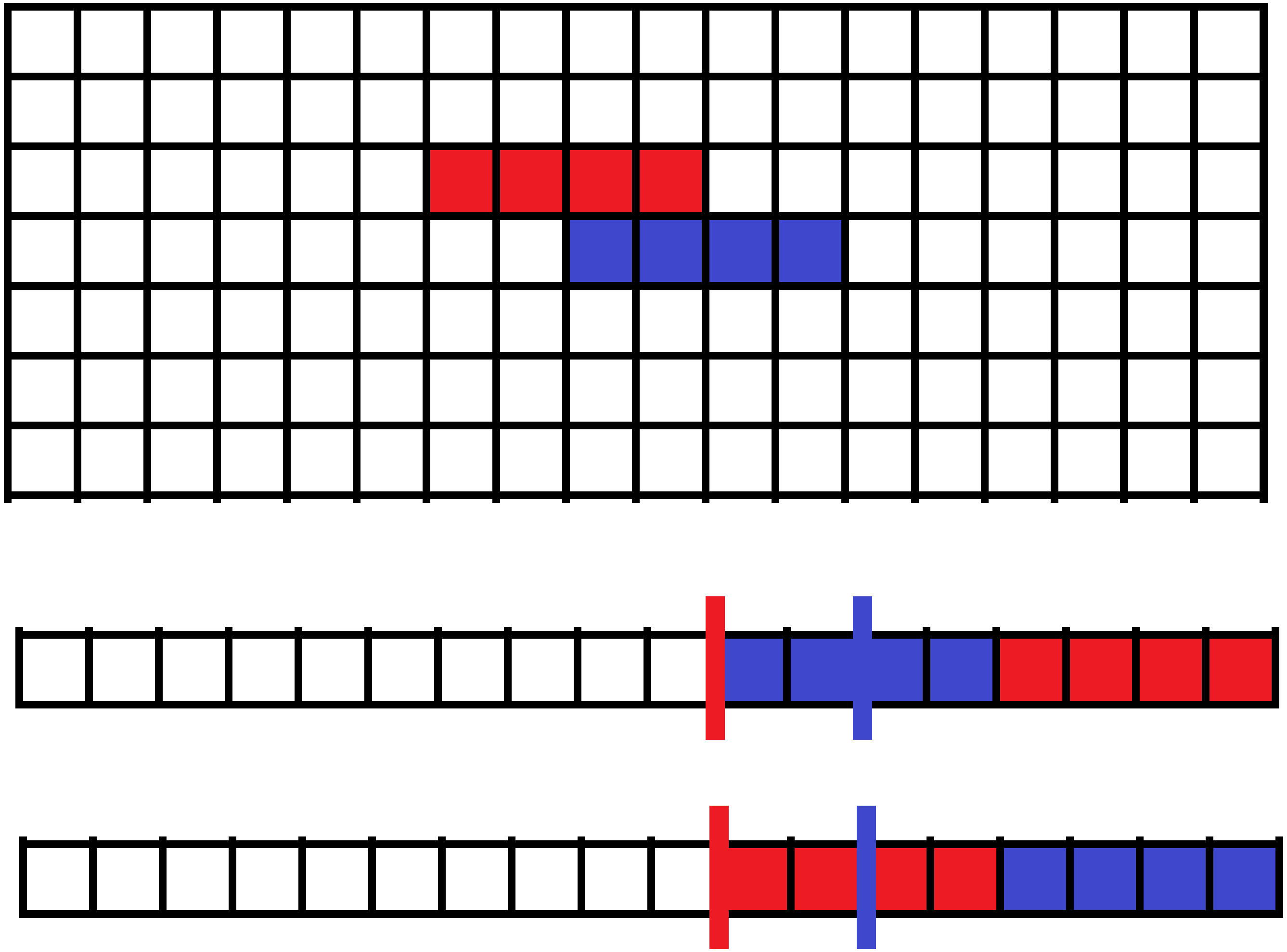
[as before, we assume jobs are numbered so that $d_1 \leq d_2 \leq \ldots \leq d_n$]

Observation. Greedy schedule has no inversions.

Observation. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

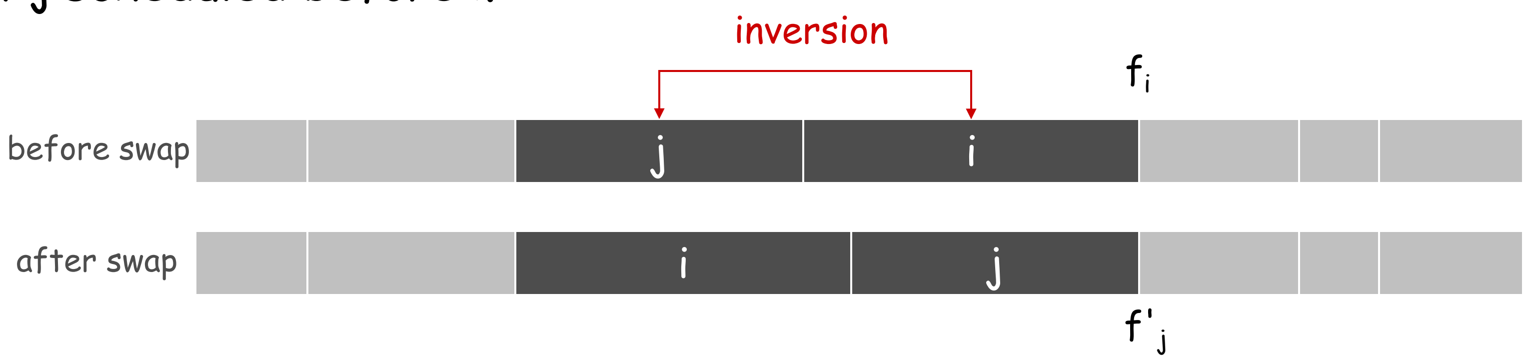






Minimizing Lateness: Inversions

Def. Given a schedule S, an **inversion** is a pair of jobs i and j such that: i < j but j scheduled before i.



Claim. Swapping two consecutive, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

Pf. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards.

- $\ell'_k = \ell_k$ for all $k \neq i, j$
- $\ell'_i \leq \ell_i$
- If job j is late:

$$
\begin{aligned}
\ell'_j &= f'_j - d_j && \text{(definition)} \\
&= f_i - d_j && (j \text{ finishes at time } f_i) \\
&\leq f_i - d_i && (i < j) \\
&\leq \ell_i && \text{(definition)}
\end{aligned}
$$

Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy schedule S is optimal.

Pf. Define S* to be an optimal schedule that has the fewest number of inversions, and let's see what happens.

- Can assume S* has no idle time.
- If S* has no inversions, then S = S*.
- If S* has an inversion, let i-j be an adjacent inversion.
  - swapping i and j does not increase the maximum lateness and strictly decreases the number of inversions
  - this contradicts definition of S* ▪

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.

Other greedy algorithms. Kruskal, Prim, Dijkstra, Huffman, …