Lecture 6 - Greedy Scheduling

CMPT 307 D-1 Fall 2024

Instructor: David Mitchell

Main Topics

Greedy algorithms for
- Interval Scheduling
- Interval Partitioning
- Scheduling with Deadlines

Greedy Algorithms

- Construct a solution in small steps, at each step myopically making a choice that optimizes some simple property.
  - Gale-Shapley is a kind of greedy algorithm
  - Cashiers (used to) use a greedy algorithm for making change with the fewest coins:
    - add as many quarters as possible
    - add as many dimes as possible
    - add as many nickels as possible
    - add as many pennies as needed

- Often simple & efficient
- Thinking of them is often easy
  - finding those that work &
    proving they work not so much...

Interval Scheduling

Instance: · Set of jobs $J = \{1, \ldots, n\}$ // (jobs, requests, activities, ...)
· Start and finish times for each job: $s(i) < f(i)$

Jobs i, j are **compatible** if their times don't overlap:

$$f(i) < s(j) \quad \text{or} \quad f(j) < s(i)$$

and **conflict** if they do:

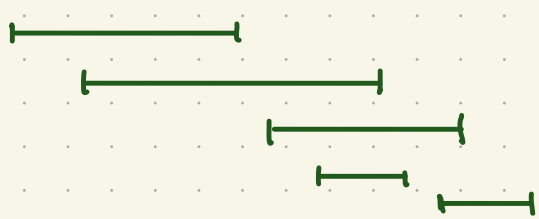
$$[s(i), f(i)] \cap [s(j), f(j)] \neq \emptyset$$

Problem: Find a maximum size compatible subset of J

$S \subseteq J$ s.t. no two jobs in S conflict.

an optimal subset of jobs

Eg.:

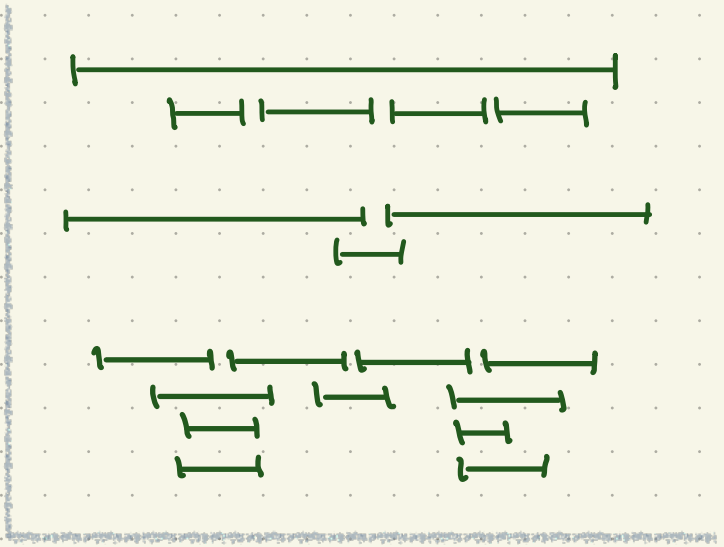


} 5 intervals

} optimal compatible subset.

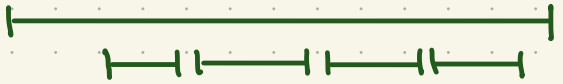
Greedy algorithm idea:

- order jobs by some property
- repeat: select "best" remaining job.
  - reject all jobs that conflict with it.
- output selected jobs

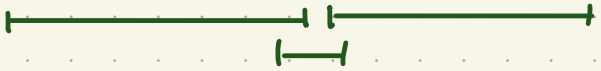


Candidates for Property.

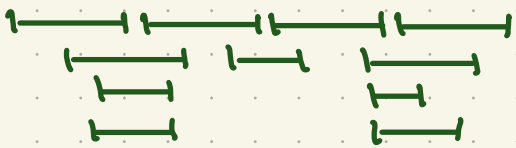
- Earliest Start Time (ie. min $s(i)$): Fails for:



- Shortest Duration (min. $f(i)-s(i)$):
  - Works for:
  - Fails for:



- Minimum Conflicts:
  - Works for these two
  - Fails for

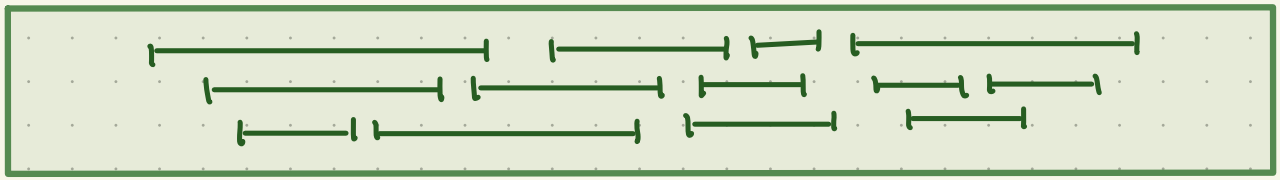


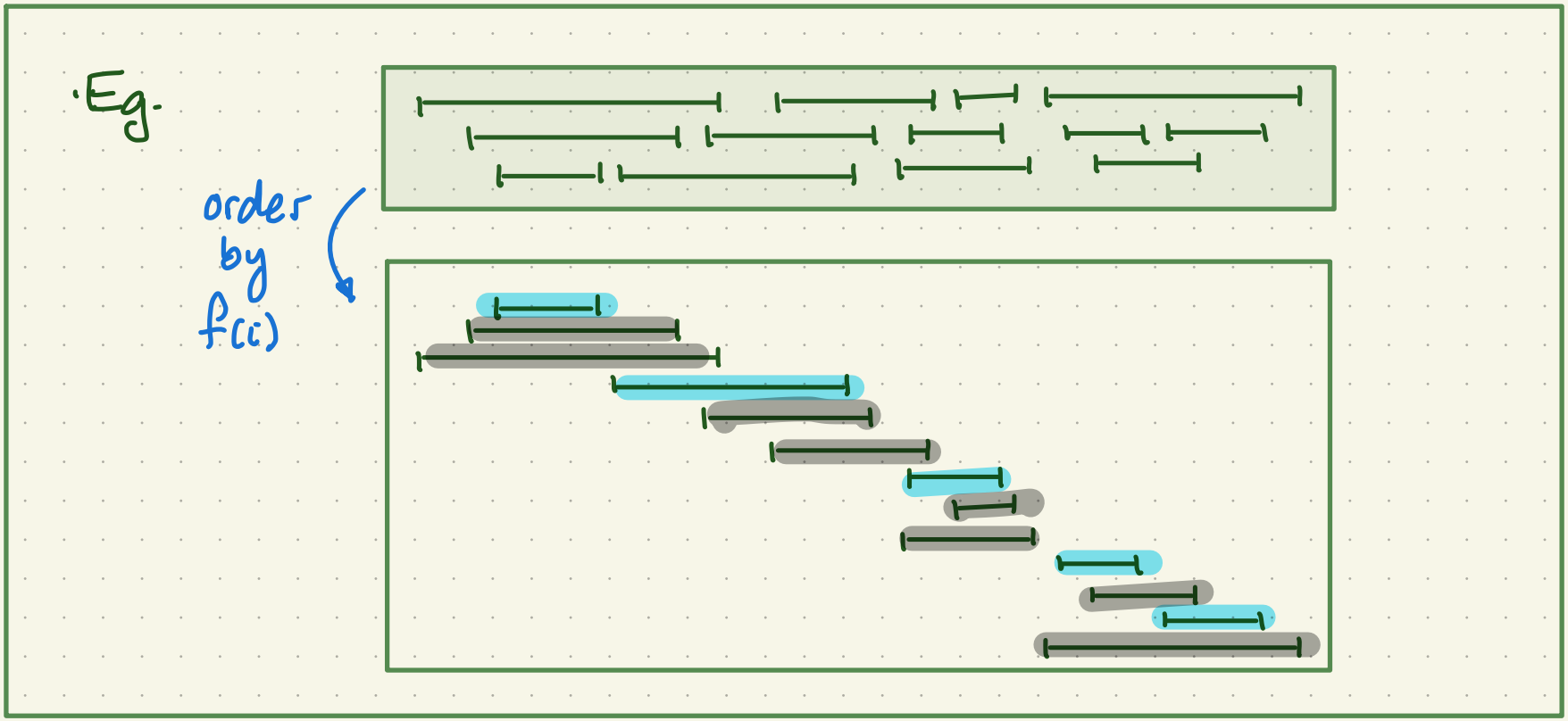
Greedy Algorithm for Interval Scheduling.

- Correct Greedy Property: earliest finish time (ie, min $f(i)$).

```
A = ∅ // set of accepted jobs
while J ≠ ∅ {
    i = job in J with min finish time
    add i to A
    delete all jobs that conflict with i from J //includes i
}
output A
```

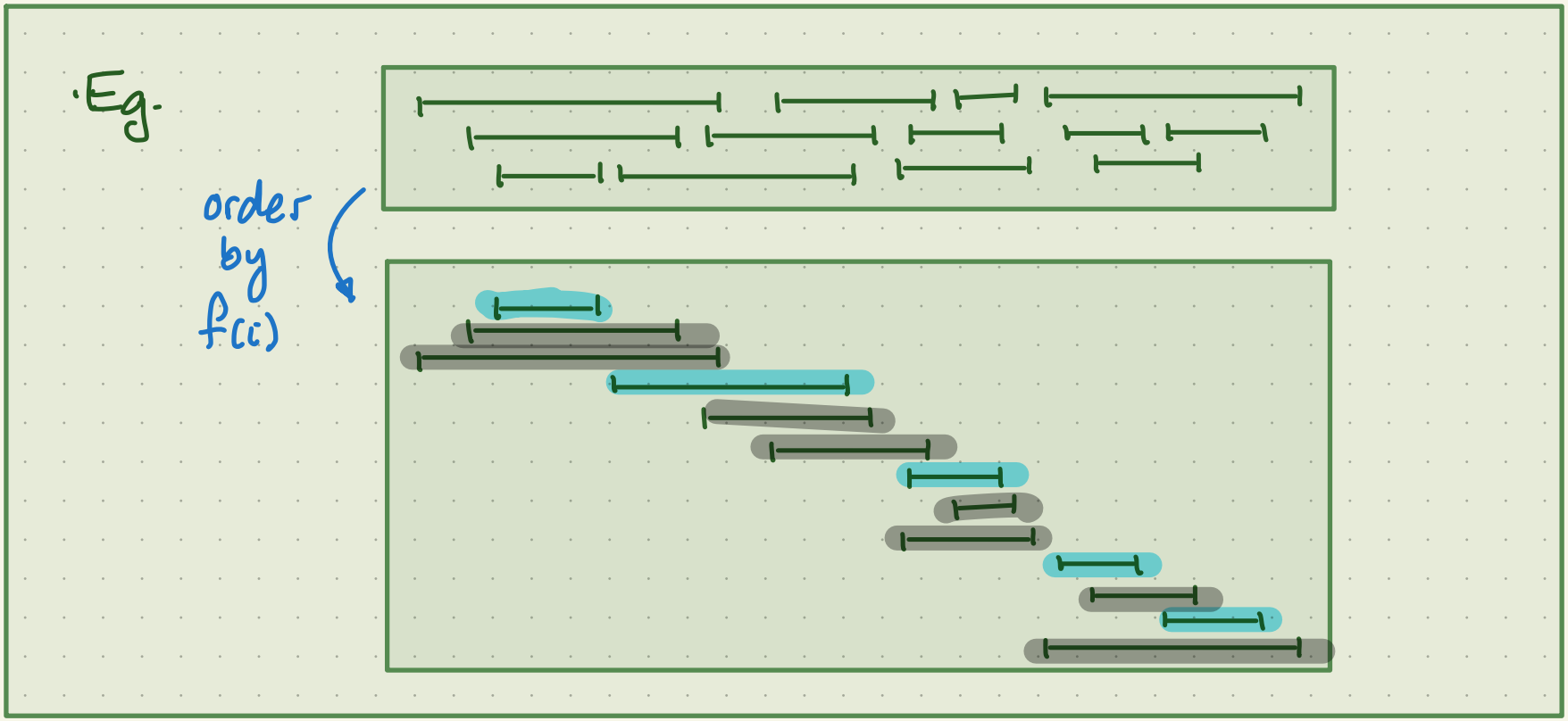
- Eg.





- Optimal Algorithm: always produces an optimal solution

- Q1 : How can we tell this algorithm is optimal?
- Q2 : How efficient is it?

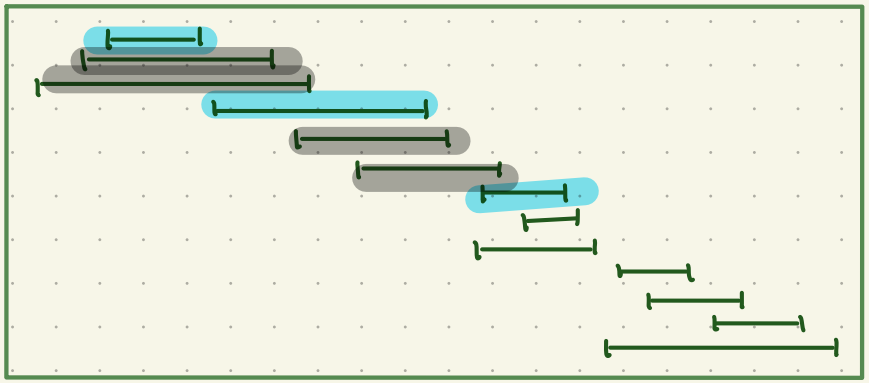


- Optimal Algorithm: always produces an optimal solution

- Q1 : How can we tell this algorithm is optimal?
- Q2 : How efficient is it?

Efficient Implementation

- order jobs by finish time // so $i < j \Rightarrow f(i) \leq f(j)$
- $A = \emptyset$ // set of accepted jobs
- $f = 0$ // max. finish time of a job in A – assumes times all ≥ 0
- for $i = 1..n$ {
  - if $s(i) > f$ {
    - $A = A \cup \{i\}$
    - $f = f(i)$
  }
}



Time: $\Theta(n \log n)$ to sort by $f(i)$

$\Theta(n)$ for for loop

$\Rightarrow \Theta(n \log n)$

- Notice: "Delete all jobs that conflict with i" is implicit.
  We don't delete them, just pass them by.

Optimality of Earliest Finishing Time

Theorem: The earliest-finishing-time greedy alg. is optimal.

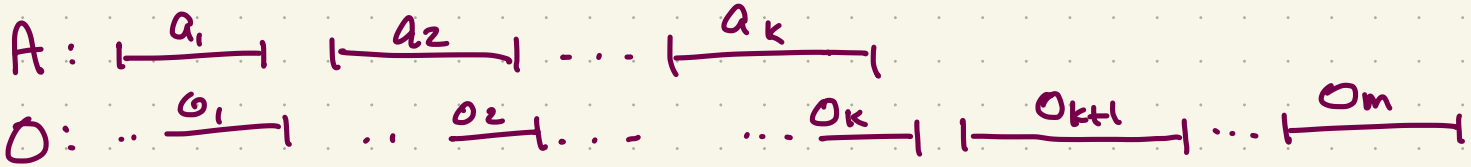
Pf: ("Greedy stays ahead")

Let $A = \{a_1, \dots a_k\}$ be algorithm output,
$O = \{o_1, \dots o_m\}$ some optimal solution

Suppose O is better than A, ie, $m > k$.

We will prove that, for every i, $1 \le i \le k$, $f(a_i) \le f(o_i)$. (*)

If $m > k$, we then have:



This cannot be, because $o_{k+1} \dots o_m$ are compatible with a_k, so the algorithm would have added intervals after a_k. ⚡

□

Proof of (*)

Lemma: Let $A = \{a_1, \dots a_k\}$ be algorithm output,
$O = \{o_1, \dots o_m\}$ some optimal solution.
Then $f(a_i) \le f(o_i)$, for every i, $1 \le i \le k \le m$.

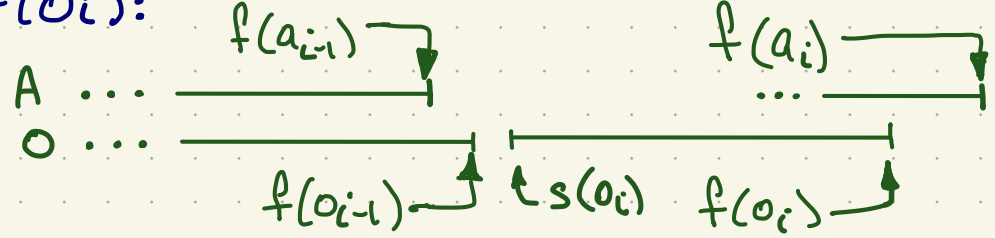
Pf: By induction on i.

Basis) $i = 1$. $f(a_1)$ is the min. finish time, so $f(a_1) \le f(o_1)$.

IH) Let $i > 1$, and assume $f(a_{i-1}) \le f(o_{i-1})$.

IS) Need to show $f(a_i) \le f(o_i)$.

Assume $f(a_i) > f(o_i)$:



Then $f(a_{i-1}) \le f(o_{i-1}) \le s(o_i)$,

I.H. — o_{i-1} & o_i are compatible

So, when a_i was chosen, o_i was compatible with $a_1 \dots a_{i-1}$. But $f(o_i) < f(a_i)$, so a_i would not have been chosen - a contradiction.

So $f(a_i) \le f(o_i)$. ∎

Algorithm Strategy:

"leave as large a remaining free interval as possible"?

Interval Partitioning

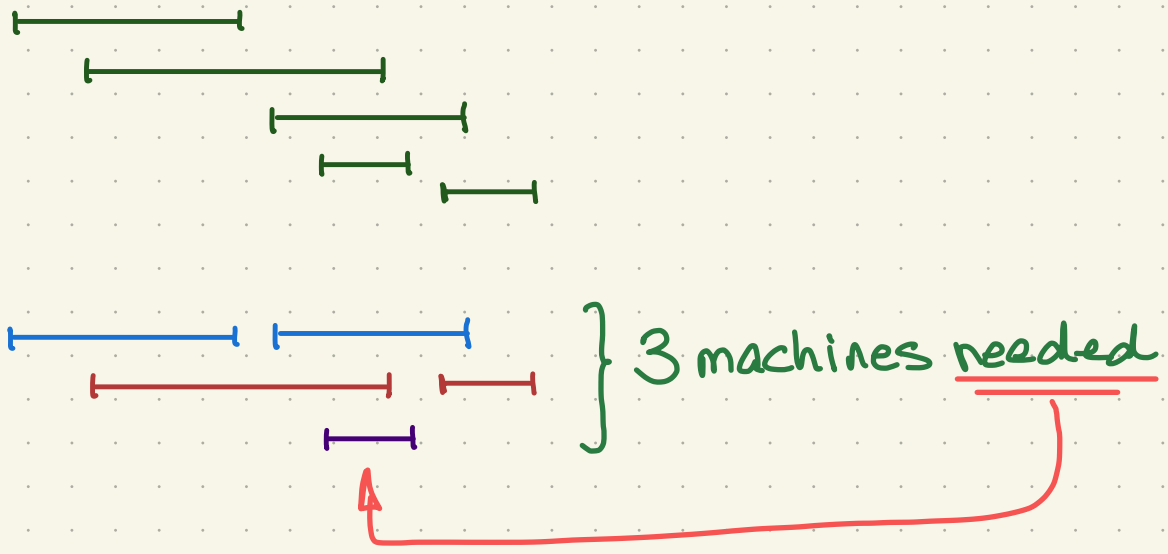
Instance: · Set of jobs $J = \{1, \ldots, n\}$ // (jobs, requests, activities, ...)
· Start and finish times for each job: $s(i) < f(i)$

- Have multiple "machines" // (machines, servers, rooms, production lines...)
- Must assign each job to a machine
- Jobs assigned to the same machine must be compatible.

Problem: Schedule all jobs using the minimum # of machines.

Assignment ≈ partition or colouring of jobs.

Eg.:



Machines ≥ Depth

- Depth of set of intervals =
  max., over all times t, of # intervals containing t.

Claim: Let S be a set of intervals // i.e, instance of interval partitioning
Every schedule for S has at least depth of S machines.

Pf: Let d be the depth of S. There are intervals $I_1, \dots I_d \in S$ and time t s.t $t \in \cap\{I_1, \dots, I_d\}$. At time t, each of $I_1 \dots I_d$ must be assigned to a different machine.

Q1: Is depth of S machines always sufficient?

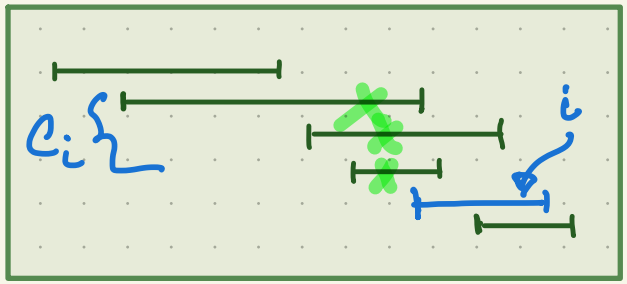
Q2: Is there an efficient algorithm to construct an optimal schedule - i.e, using a min. # of machines?

Greedy Interval Partitioning

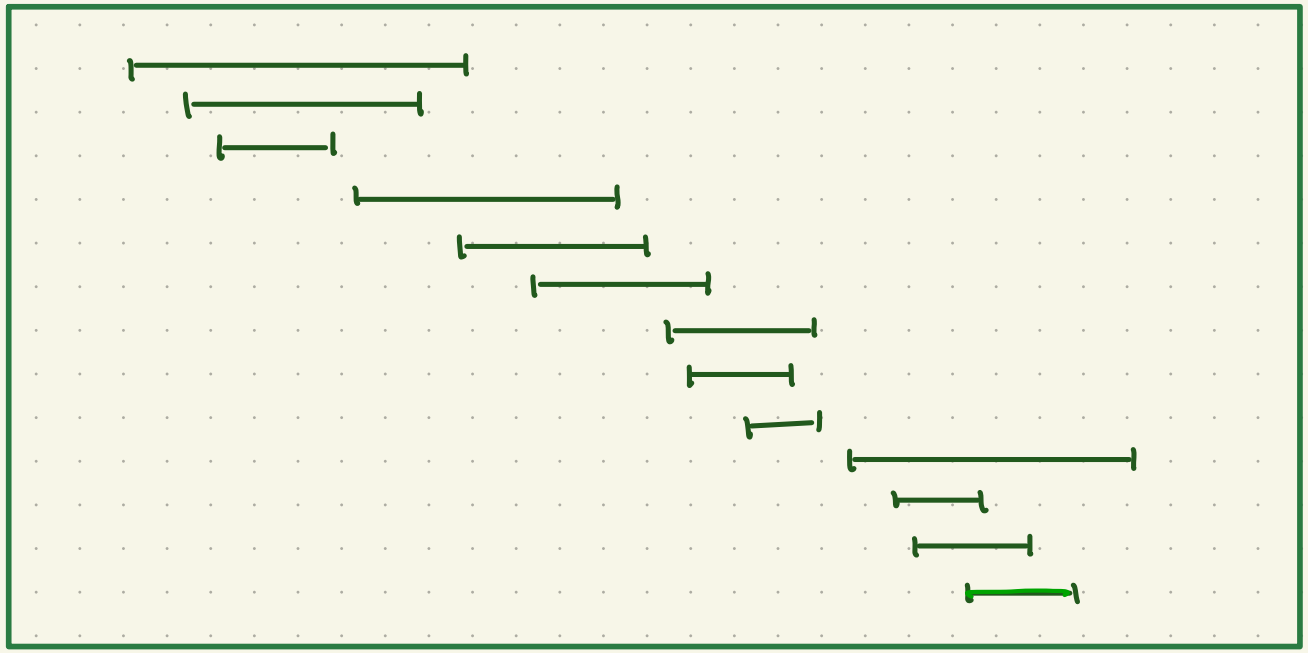
we write $[d]$ for $\{1, 2, \ldots, d\}$

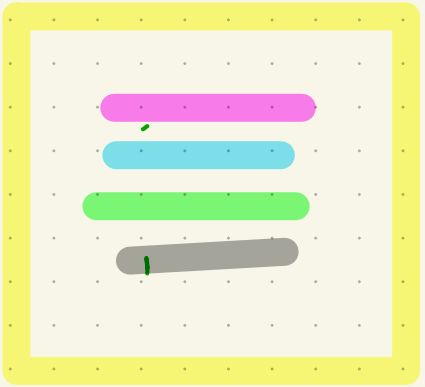
Input: Set I of n intervals.
Sort intervals by non-decreasing start time
// i.e, so $j < i \Rightarrow s(j) \leq s(i)$, for $1 \leq i, j \leq n$
d = depth of I
Define $C_i = \{ j < i : f(j) \geq s(i) \}$ // $C_i = \{ j \mid s(j) < s(i) \leq f(j) \geq s(i) \}$
// C_i is intervals that precede i in the order and intersect with i
for $i = 1..n$ {
    $X_i = \{ x \in [d] : x$ is the label of an interval in $C_i \}$ // Colours not
    label i with an element of $[d] - X_i$ // (*) // available
}
Output the set of labels

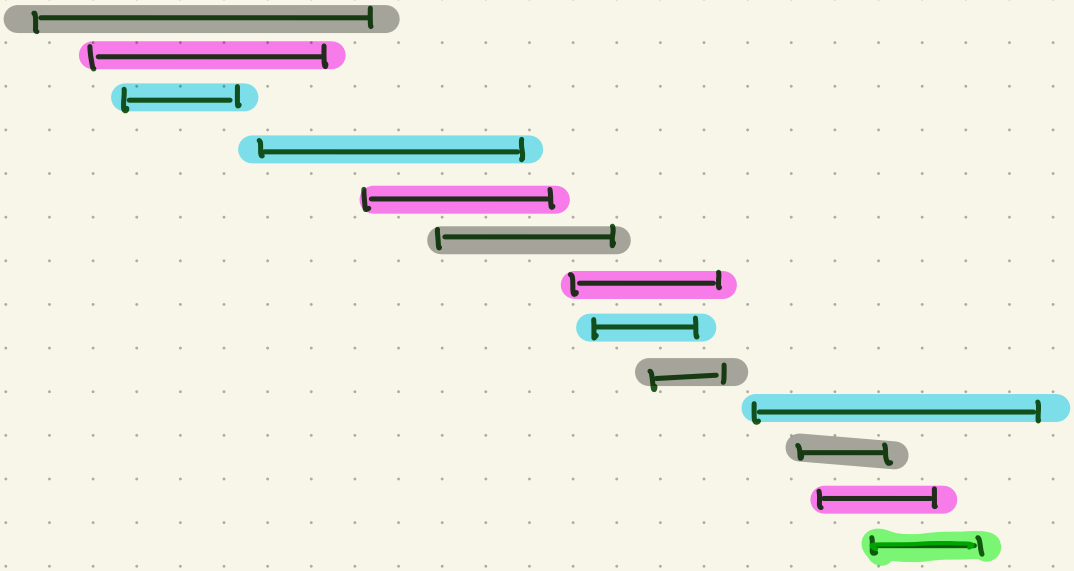
- Assuming (*) is always possible, the algorithm labels every interval in I.
- We must show that:
    1) (*) is always possible
    2) Intersecting intervals have different labels.
- Optimality follows by choice of d.



(*) = label i with a colour that is not currently in use.







Greedy Interval Partitioning – Correctness + Complexity

Proof of 1) Consider an interval i, and suppose $|C_i| = t$.
$C_i \cup \{i\}$ is a set of $t+1$ intervals that all contain $s(i)$
Then $t+1 \leq d$, so $t \leq d-1$.
It follows that $|X_i| < d$, so (*) is possible

Proof of 2) Suppose intervals i, j intersect, and $i < j$.
Then $i \in C_j$, so the label of i is in X_i.
The algorithm chooses a label for j that is not in X_i,
so i, j have different labels.

(Optimality Proof Method: Greedy meets a structural bound)

Claim: Greedy Interval Partitioning can be implemented to take time $O(n \log n)$

Pf: Exercise.

Scheduling to Minimize Lateness

Instance: Jobs $1..n$ with durations $t(i)$ and deadlines $d(i)$

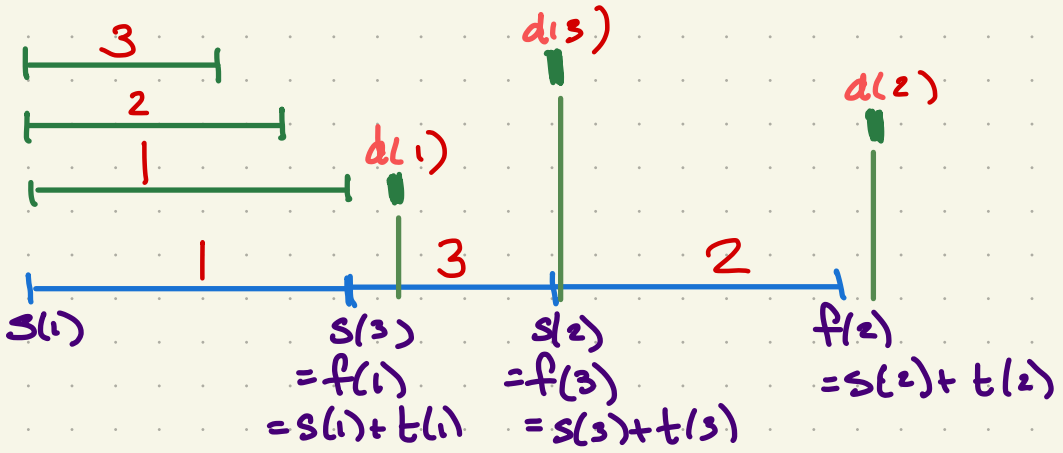
- We have 1 machine // 1 job at a time
- must schedule all jobs // assign start times $s(i)$
  // $f(i) = s(i) + t(i)$
- no jobs conflict // $[s(i), f(i)) \cap [s(j), f(j)) = \emptyset$
- job i is late if $f(i) > d(i)$ // lateness of i is $L_i = \max\{0, f(i) - d(i)\}$
- maximum lateness of a given schedule: $\max_{i \in [n]} L_i$

Problem: Find a schedule with minimum maximum lateness.

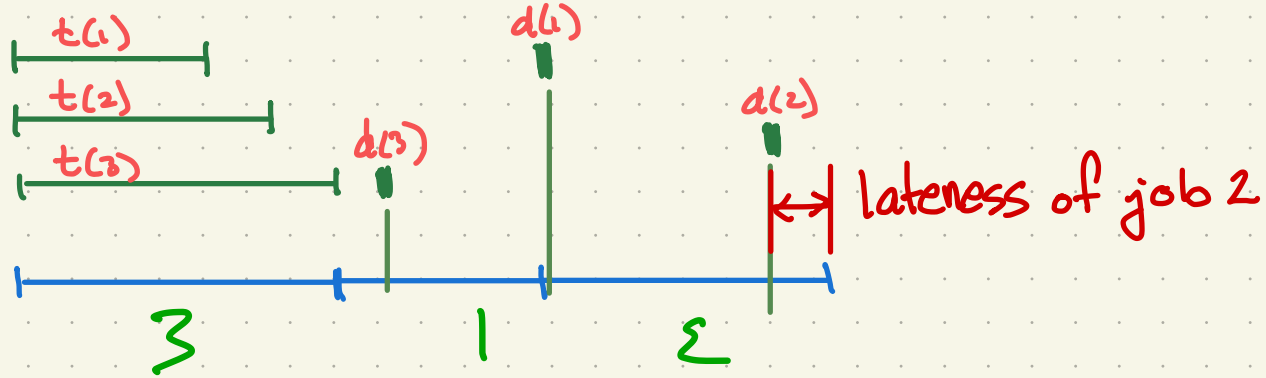
$[5, 10)$ & $[10, 15)$ do not conflict.

Scheduling to Minimize Lateness

Ex. 1.



Ex. 2.



Minimizing Lateness – Earliest Deadline First

Order jobs by non-decreasing deadline.
// $i < j \Rightarrow d(i) \leq d(j)$
$f = 0$ // f is finish time of last job scheduled so far
for $i = 1..n$ {
    assign $s(i) = f$
        // job i goes right after job i-1
    set $f = f + t(i)$
}
output $\{[s(i), s(i) + t(i)] : i \in [n]\}$

OR:
- Order jobs by deadline
- Let $s(i) = \sum_{j<i} t(i)$, for $i \in [n]$

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

Q: How can we show it is optimal?

Earliest Deadline First

- Let A be the output of the algorithm.
- We show A is optimal by (roughly):
  - start with an optimal schedule O
  - transform O into A by making changes that do not increase lateness

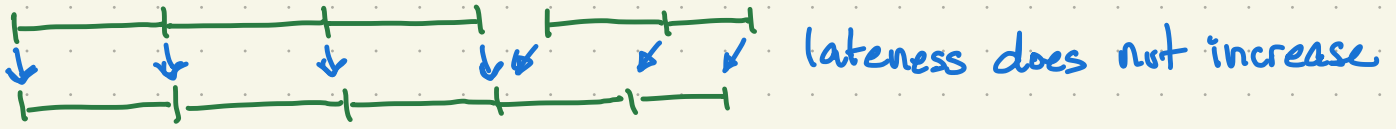
(Optimality proof method: exchange argument).

- We first establish some terms + facts about schedules.

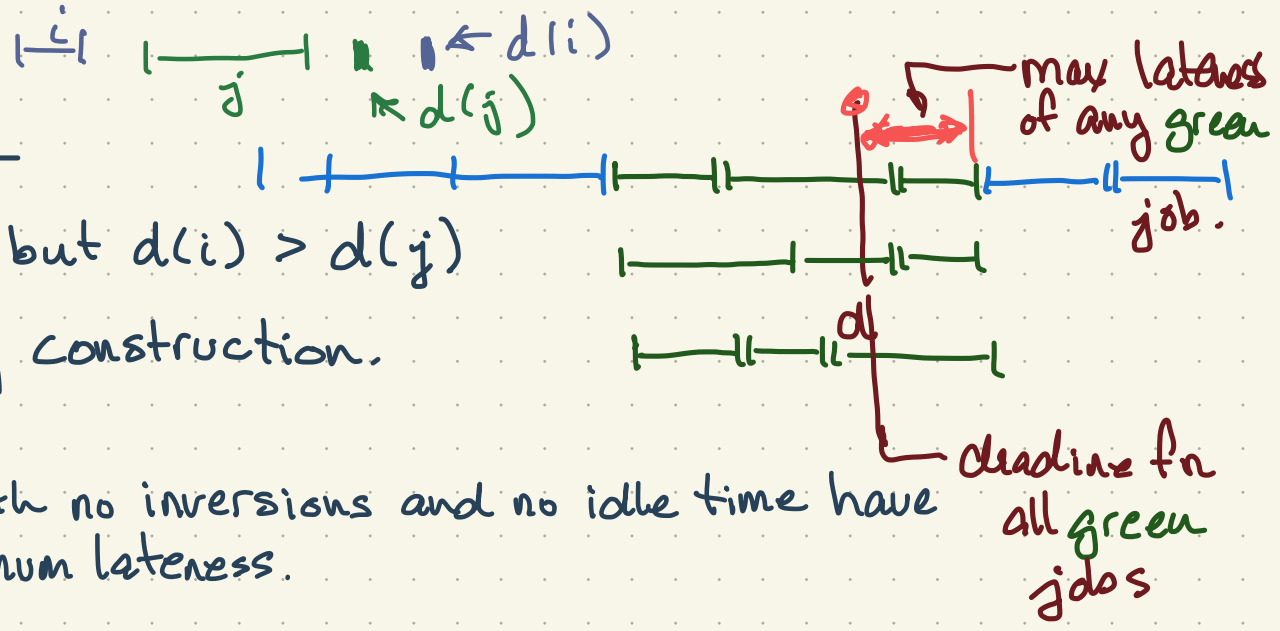
Idle time:

- time when no job scheduled, but some job is scheduled later.
  // machine is sitting idle, but work left to do
- A has no idle time, by construction

Fact 1: There is an optimal schedule with no idle time.



Earliest Deadline First



Inversion: $s(i) < s(j)$ but $d(i) > d(j)$

- A has no inversions, by construction.

Fact 2: All schedules with no inversions and no idle time have the same maximum lateness.

Pf: Consider two schedules B, C.

No inversions ⇒ jobs ordered by non-decreasing deadline.

No inversions, all deadlines distinct ⇒ B, C order jobs the same

No inversions, distinct deadlines, no idle time, ⇒ B, C the same.

So, B, C can differ only in order of jobs with same deadline.

Jobs with the same deadline are scheduled consecutively.

//after jobs with earlier deadlines, before jobs with later deadlines

Among jobs with deadline d, the last has greatest lateness.

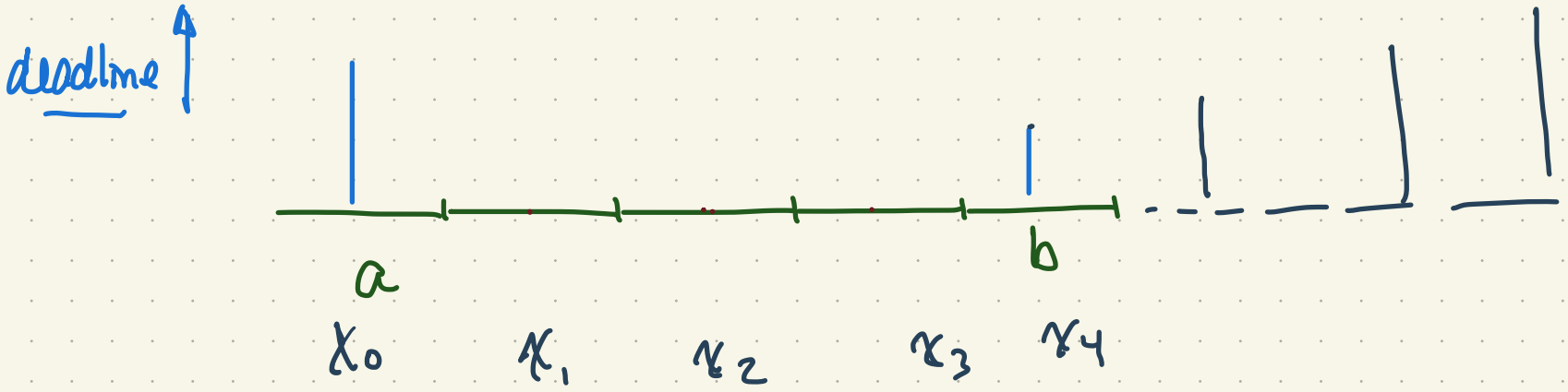
* This lateness does not depend on the order of those jobs.

So B, C have the same max. lateness ∎

Earliest Deadline First

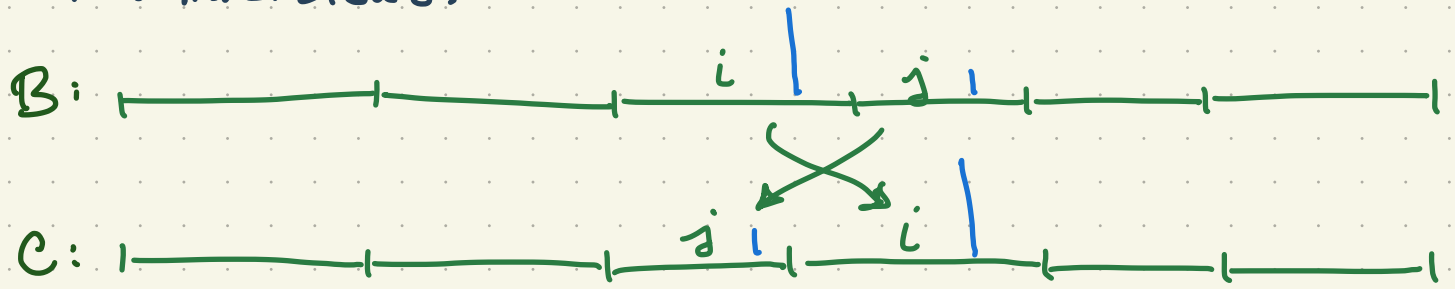
Fact 3: If a schedule has an inversion, it has two consecutive jobs that are an inversion.

Pf: Suppose jobs a and b are an inversion & the schedule has jobs ordered $a = x_0, x_1, x_2, \ldots x_k = b$. We have that $d(a) > d(b)$. Let i be the least index s.t $d(x_i) > d(x_{i+1})$. Then x_i, x_{i+1} are consecutive jobs that are inverted.



Earliest Deadline First

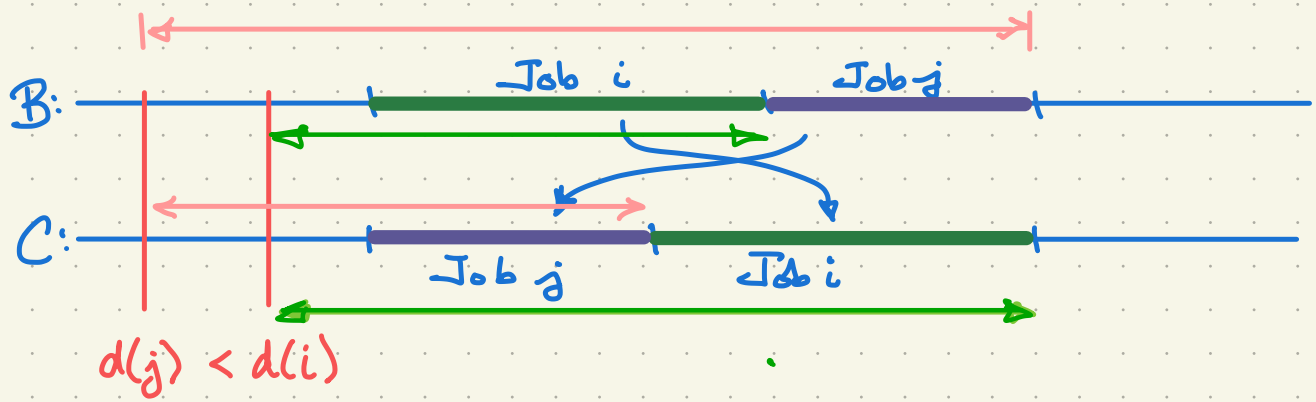
Fact 4: Let B be a schedule with no idle time and $k>0$ inversions, and i, j a consecutive inverted pair. Let C be the result of swapping i, j in B. Then C is a schedule with no idle time and $k-1$ inversions.



Earliest Deadline First

Fact 5: Let B be a schedule with no idle time and $k>0$ inversions, i, j a consecutive inverted pair, and C the result of swapping i, j in B. Then C has maximum lateness no greater than B

Pf: We have:



Write s_i^B, f_i^B, L_i^B for start, finish & lateness of job i in schedule B.

* Job j moves earlier, so cannot be more late in C than in B:

$$L_j^C = f_j^C - d(j) < f_j^B - d(j) = L_j^B$$

* Job i moves later, but its lateness in C cannot be greater than the lateness of job j in B:

$$L_i^C = f_i^C - d(i) = f_j^B - d(i) < f_j^B - d(j) = L_j^B$$

All other jobs are unchanged, so max. lateness did not increase.

Earliest Deadline First is Optimal

Lemma: There is an optimal schedule with no inversions and no idle time.

Pf: Let B be an optimal schedule with no idle time. (Fact 1)
Suppose B has k inverted pairs. Modify B k times by swapping a consecutive inverted pair. The resulting schedule has no idle time, no inversions, and is optimal. (Facts 3, 4, 5). ∎

Thm: The schedule A produced by the Earliest-Deadline-First algorithm is optimal.

Pf: A has no idle time and no inversions. (By construction)
There is an optimal schedule O with no inversions and no idle time (Lemma)
All schedules with no idle time and no inversions have the same maximum lateness. (Fact 2)
Therefore A and O have the same maximum lateness, so A is optimal.

∎

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.

(Interval Scheduling)

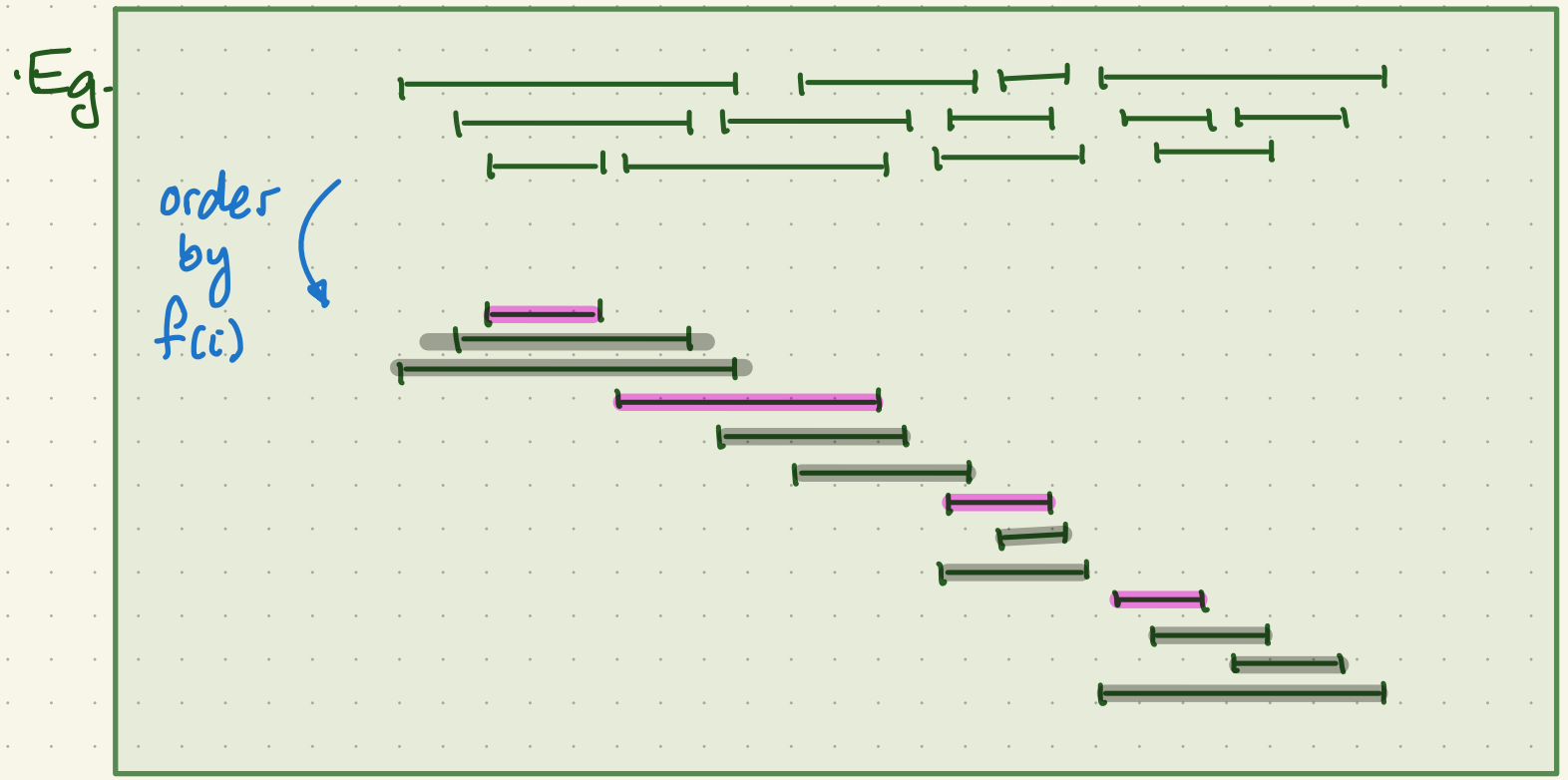
Structural. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

(Interval Partitioning)

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

(Minimizing Lateness)

End.



- We call an algorithm for an optimization problem optimal if, for every problem instance, it produces an optimal solution.
- Q1: How can we tell this algorithm is optimal?
- Q2: How efficient is it?