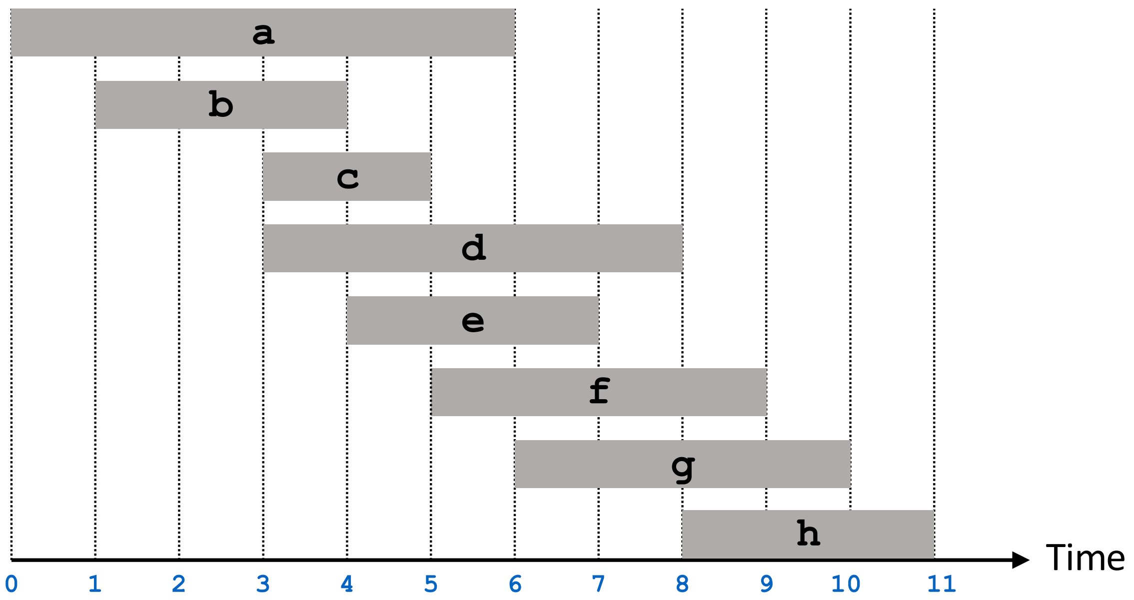
Greedy Algorithms

a. Unweighted Interval Scheduling

Interval Scheduling: Problem Definition

- 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 Attempts

- **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 Attempts

counterexample for earliest start time

counterexample for shortest interval

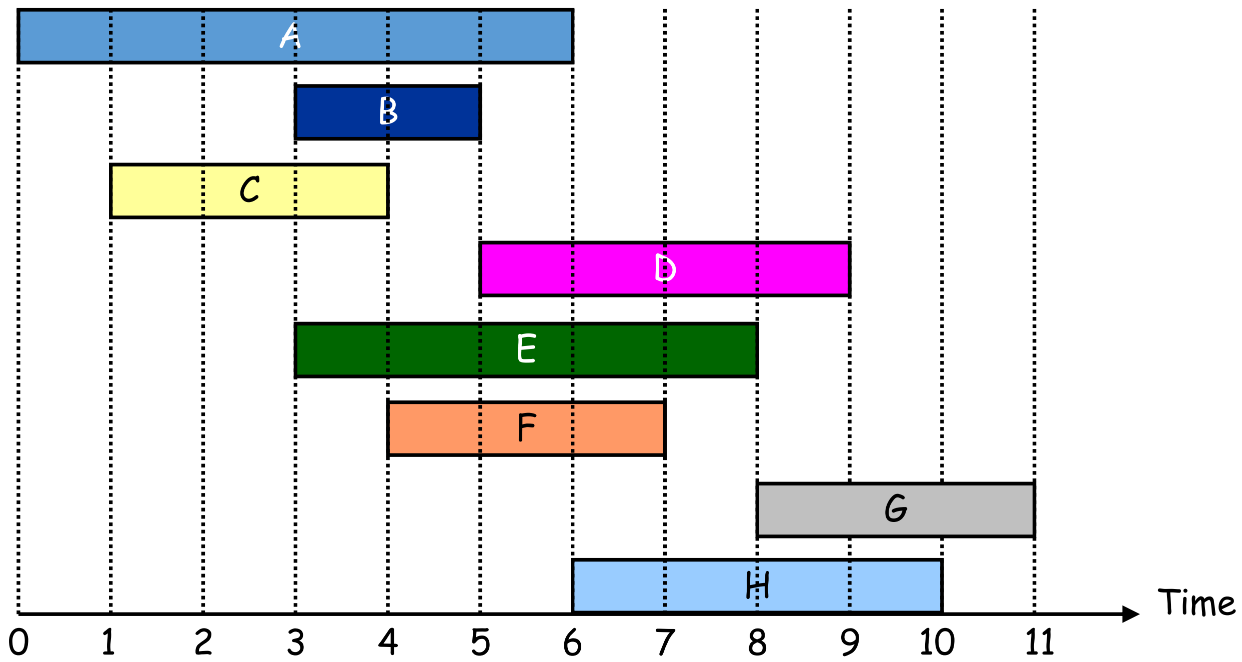
counterexample for fewest conflicts

Interval Scheduling: Greedy Algorithm

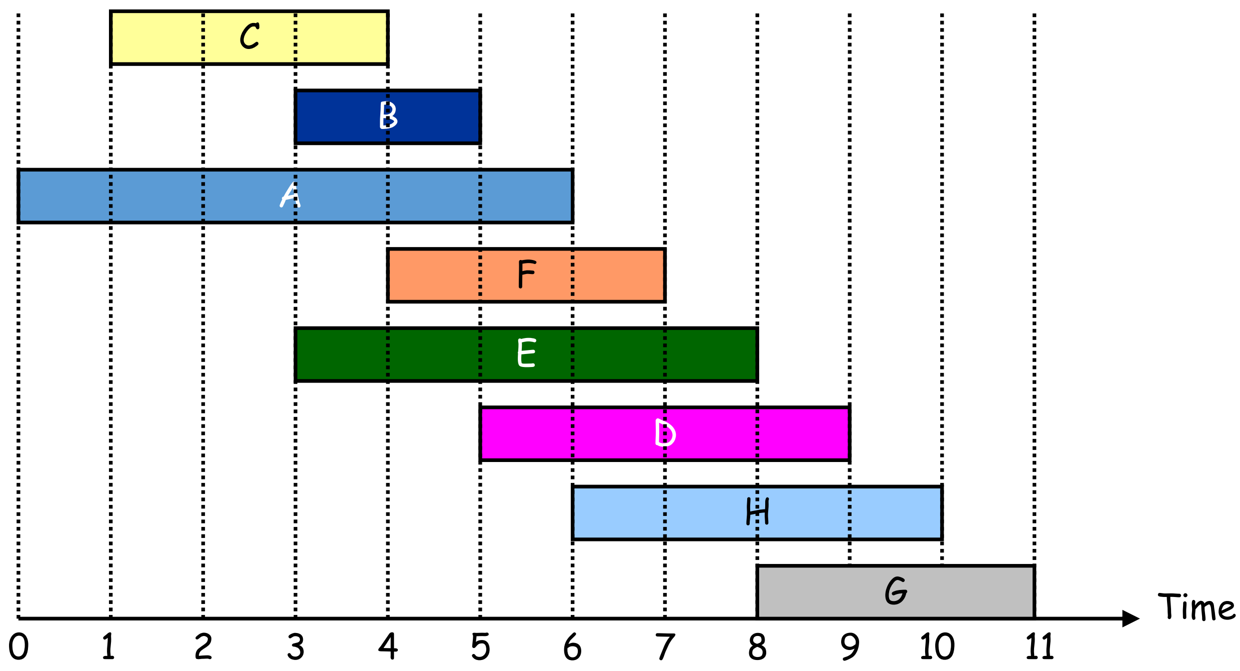
- **Greedy algorithm:** Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

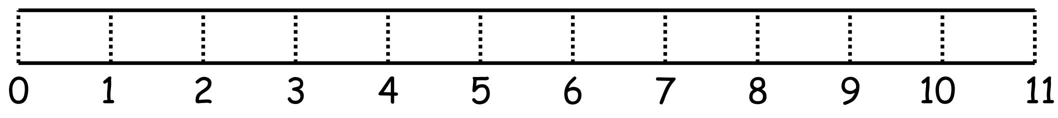
```
Sort jobs by finish times so that f₁ ≤ f₂ ≤ ... ≤ fₙ.

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

Interval Scheduling: Example



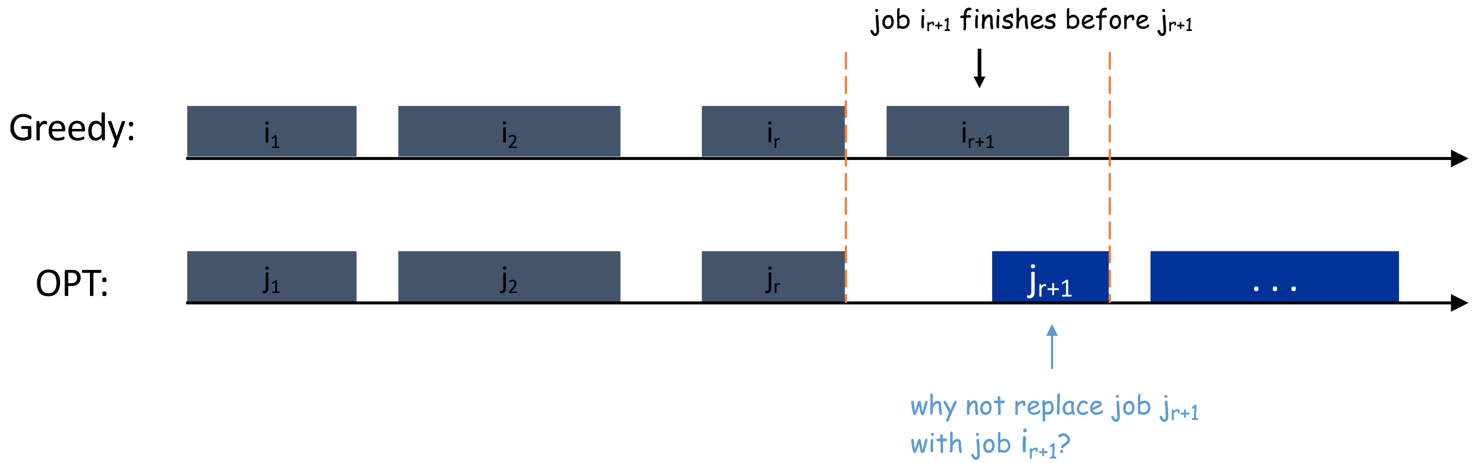
Interval Scheduling: Example





Interval Scheduling: Proof

- Theorem. Greedy algorithm is optimal.
- Pf. (by contradiction) greedy-stays-ahead approach
  - Assume greedy is not optimal, and let's see what happens.
  - Let i_1, i_2, ... i_k denote set of intervals selected by greedy.
  - Let j_1, j_2, ... j_m denote set of intervals in the optimal solution with $i_1 = j_1$, $i_2 = j_2$, ..., $i_r = j_r$ for the largest possible value of r.



Interval Scheduling: Implementation

- Finding the next earliest finishing time of remaining intervals via linear search:
  - $O(n^2)$.
- Sorting
  - Sort all the requests by finishing time — $O(n \log n)$
  - Iterate through the sorted array taking the next legal request — $O(n)$
  - $O(n \log n)$

Summary

- Scheduling problems are often amenable to greedy approach
- But there may be many greedy choices and it is important to select the right one
- **Main Takeaway:** Greedy-stays-ahead is a useful proof approach

Greedy Algorithms

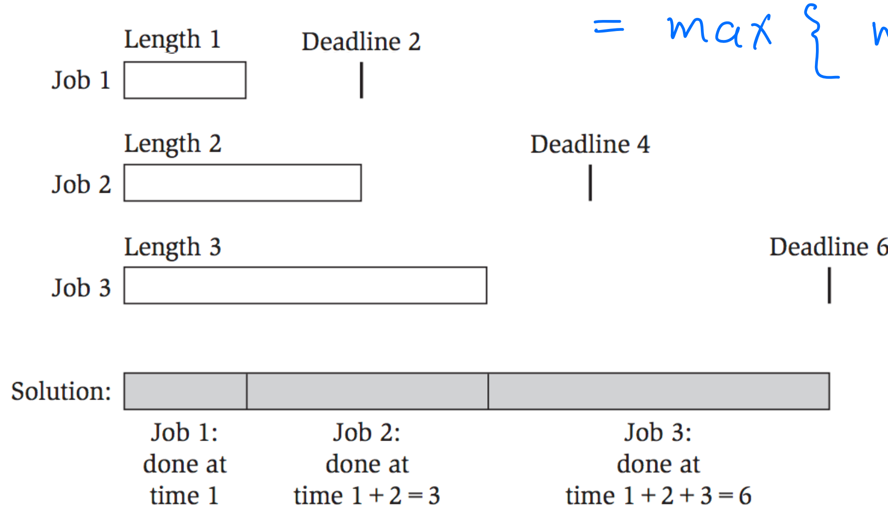
a. Interval Scheduling– Greedy Stays Ahead

b. Minimum Lateness Scheduling

Minimum Lateness Scheduling

- Input: n jobs with length t_i and deadline d_i
  - Simplifying assumption: all deadlines are distinct
- Output: a minimum--lateness schedule for the jobs
  - Can only do one job at a time, no overlap
  - The lateness of job i is $\max\{ f_i - d_i , 0 \}$
  - The lateness of a schedule is $\max \{\max_i \{ f_i - d_i \}, 0\}$

$= \max \{ \max_i \{ f_i - d_i , 0 \} \}$



Possible Greedy Rules

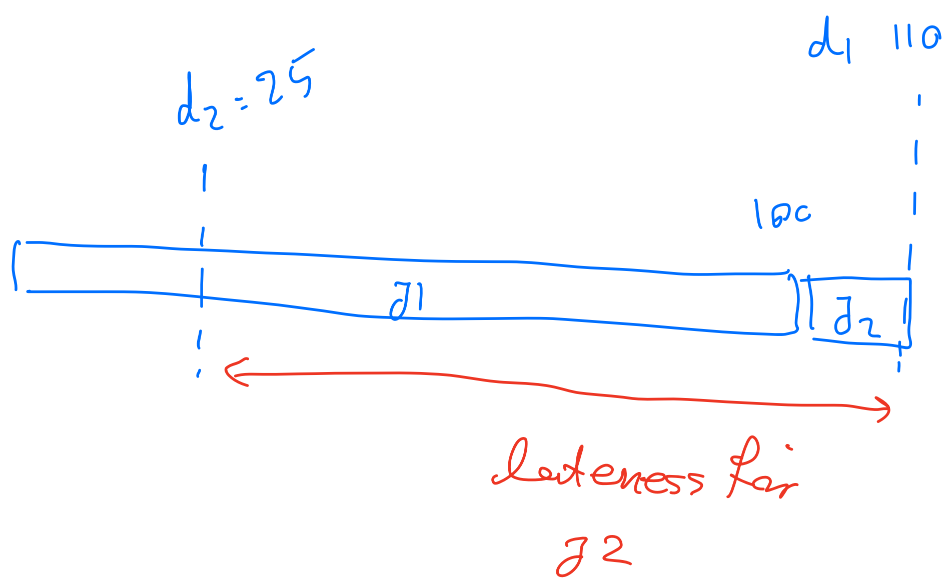
- Choose the shortest job first (min t_i)?
- Choose the most urgent job first (min $d_i - t_i$)?
- Others?

$t_1 = 100$ $d_1 = 110$

$t_2 = 10$ $d_2 = 25$

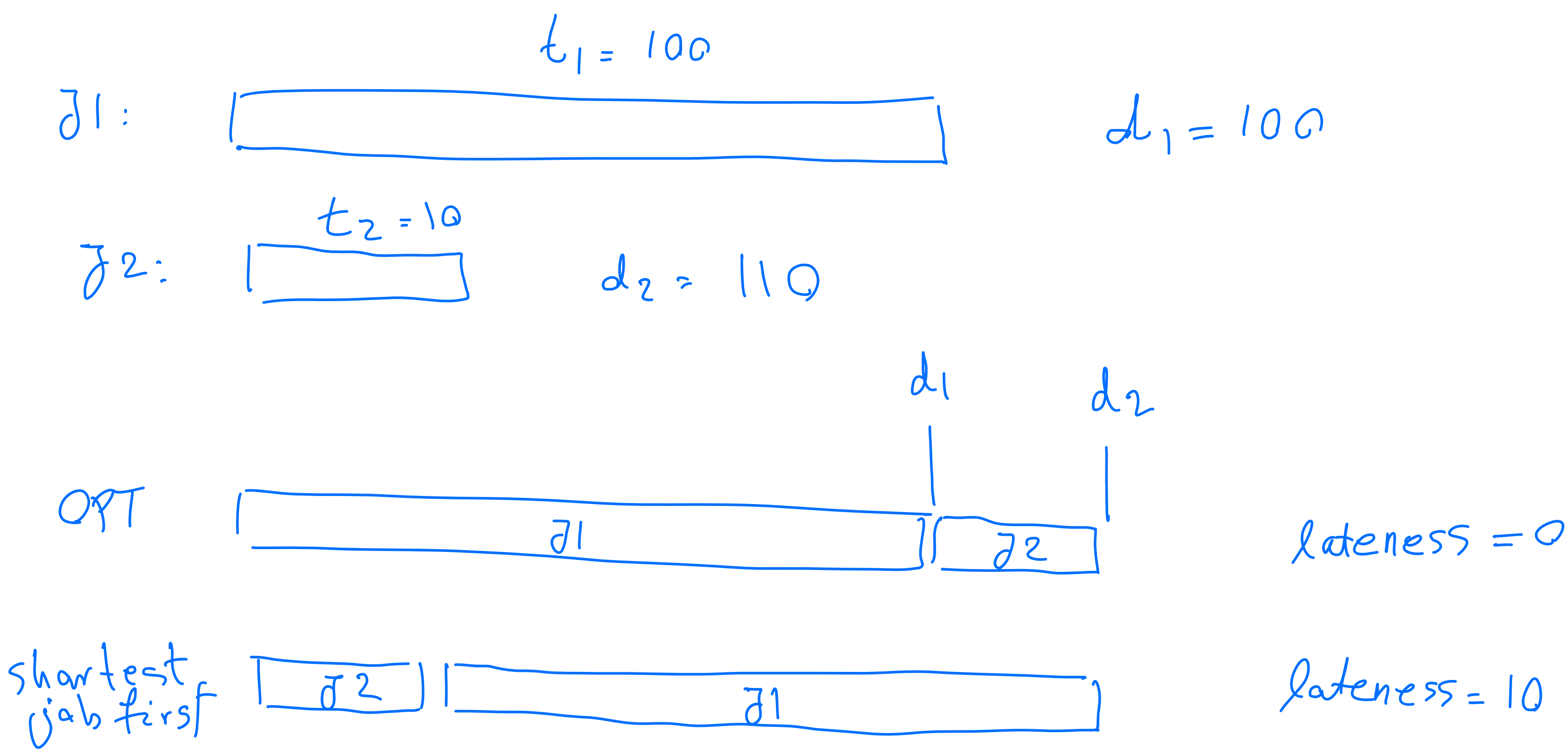
most urgent : (1, 2) lateness 110 - 25

opt : (2, 1) lateness 0



Greedy Algorithm: Earliest Deadline First

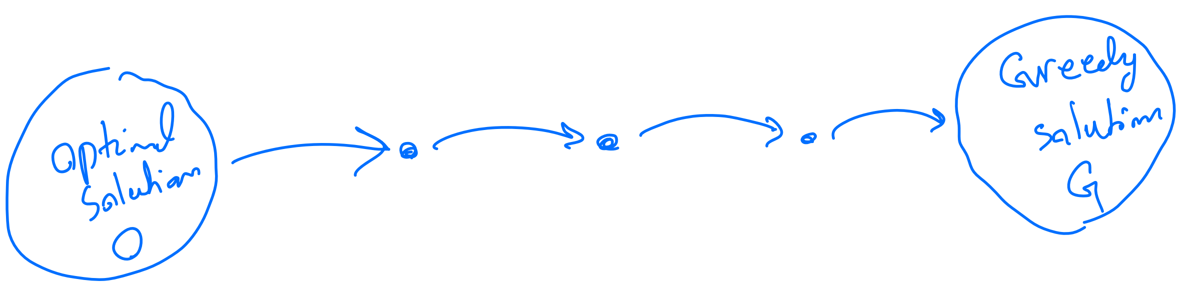
- Sort jobs so that $d_1 \leq d_2 \leq \cdots \leq d_\mathrm{n}$
- For $i = 1, \dots, n$:
  - Schedule job i right after job $i - 1$ finishes



Exchange Argument

- G = greedy schedule, O = some other schedule

- Exchange Argument:
  - We can transform O to G by exchanging pairs of jobs
  - No exchange increases the lateness of O
  - Therefore, the lateness of G is at most that of O
  - G has the minimum possible lateness



Exchange Argument

- G = greedy schedule, O = (supposedly) optimal schedule

- We say that two jobs i, j are inverted in O if $d_{\mathrm{i}} < d_{\mathrm{j}}$ but j comes before i in the schedule
  - Observation: greedy has no inversions

Example: two jobs

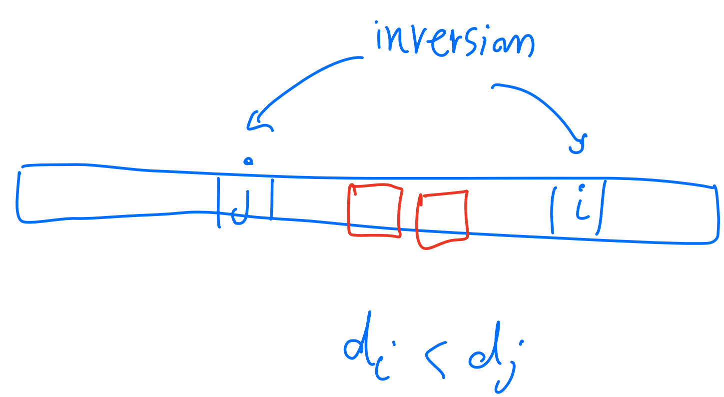
- Two jobs with deadlines $d_1 < d_2$ and lengths t_1, t_2
- Greedy schedule: 1, 2
- O: 2, 1 (inversion)

Lateness of O: $\max(t_2 - d_2,\ t_1 + t_2 - d_1) = t_1 + t_2 - d_1$

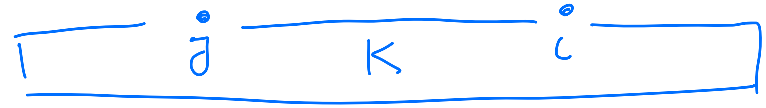
Flipping them: $\max(t_1 + t_2 - d_2,\ t_1 - d_1) \leq t_1 + t_2 - d_1$

Exchange Argument

- We say that two jobs i, j are inverted in O if $d_i < d_j$ but j comes before i in O
- Claim: an optimal schedule has no inversions
  - Step 1: If O has an inversion, then it has an inversion i, j which are scheduled consecutively in O
  - Step 2: if i, j are consecutive jobs that are inverted then flipping them only reduces the lateness



Exchange Argument



- **Step 1:** If O has an inversion, then it has an inversion i, j which are scheduled consecutively in O

- Take an inversion i, j where i and j are closest in the schedule O
  - By definition, $d_j > d_i$ but j comes before i
- Suppose there is a job k scheduled between ~~k~~ i and j.
- **Case 1**: $d_k < d_j$
  - In this case j, k is an inversion, contradiction
- **Case 2**: $d_k > d_j$ because i, j is an inversion
  - Since $d_j > d_i$, we have $d_k > d_i$
  - Therefore, k, i is an inversion, contradiction

Exchange Argument

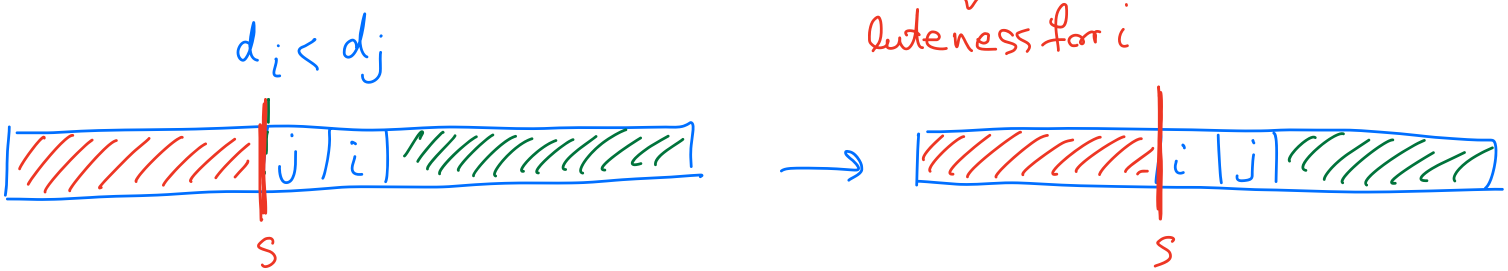
- **Step 2:** If i, j are consecutive jobs that are inverted then flipping them only reduces the lateness
  - Does not change the lateness of the other jobs
  - Let's assume these jobs have $d_i < d_j$ and lengths t_i, t_j
  - Assume job j starts at time s in schedule O.

Max lateness of ~~1~~ i and ~~2~~ j before flipping:

$$\max(\underbrace{s + t_j - d_j}_{\text{lateness for } j},\ \underbrace{s + t_j + t_i - d_i}_{\text{lateness for } i}) = s + t_i + t_j - d_i$$

Max lateness of ~~1~~ i and ~~2~~ j after flipping:

$$\max(s + t_i + t_j - d_j,\ \underbrace{s + t_i - d_i}_{\text{lateness for } i}) < s + t_i + t_j - d_i$$



Exchange Argument

- We say that two jobs i, j are inverted in O if $d_{\mathrm{i}} < d_{\mathrm{j}}$ but j comes before i in O
- Claim: an optimal schedule has no inversions
  - Step 1: If O has an inversion, then it has an inversion i, j which are scheduled consecutively in O
  - Step 2: if i, j are consecutive jobs that are inverted then flipping them only reduces the lateness

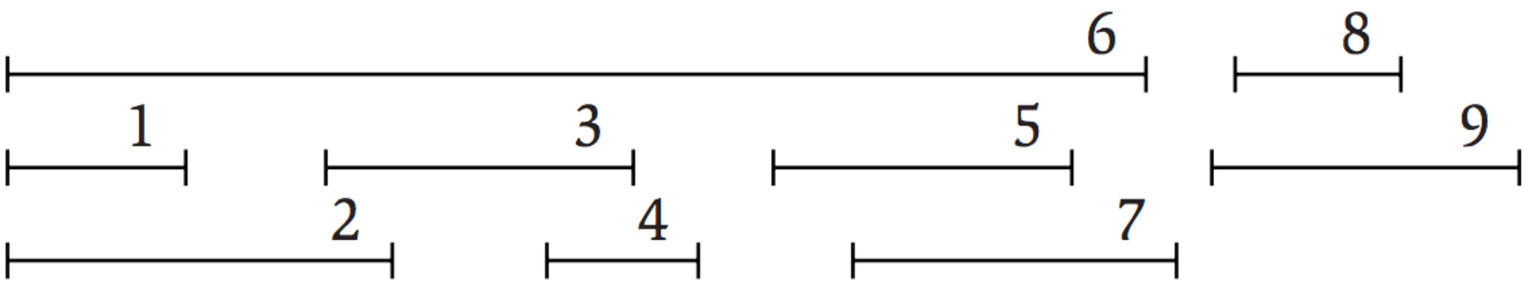
- G is the unique schedule with no inversions, lateness(G) $\leq$ lateness(O)

Greedy Algorithms

a. Interval Scheduling – Greedy Stays Ahead
b. Minimum Lateness Scheduling
c. Interval Scheduling – Exchange Argument

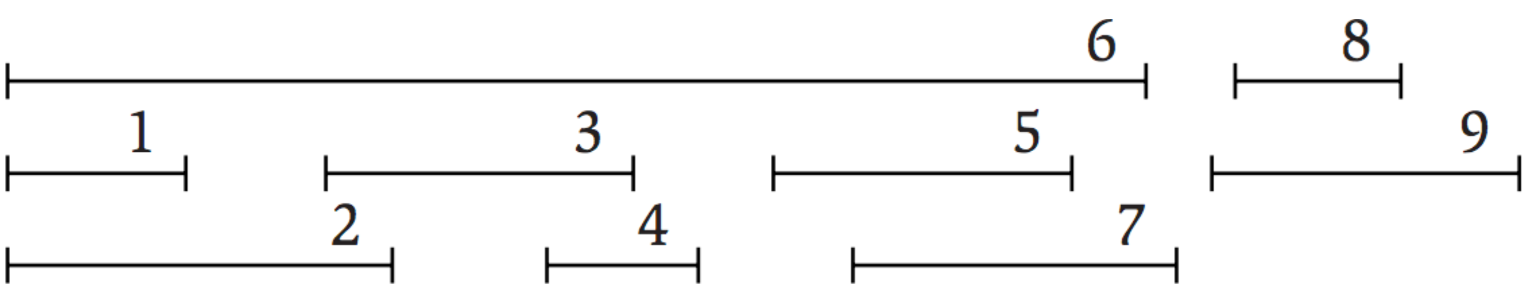
(Unweighted) Interval Scheduling

- Input: n intervals $(s_{\mathrm{i}}, f_{\mathrm{i}})$
- Output: a compatible schedule S with the largest possible size
  - A schedule is a subset of intervals $S \subseteq \{1, \ldots, n\}$
  - A schedule S is compatible if no two $i, j \in S$ overlap



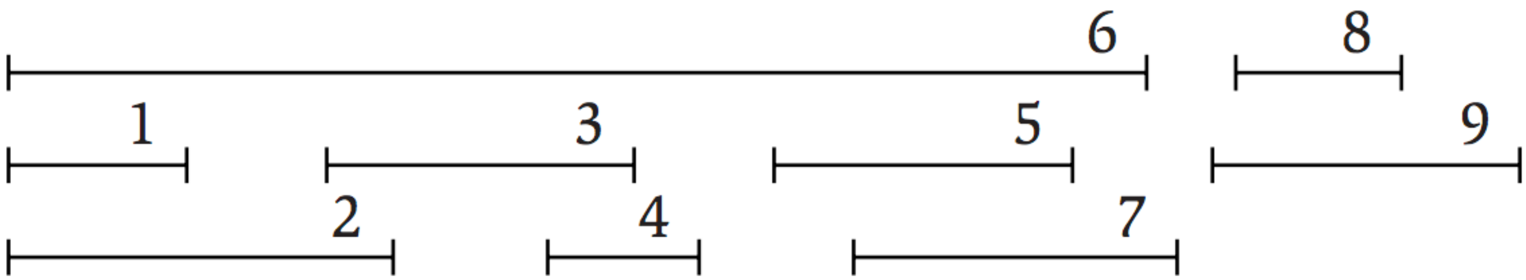
Greedy Algorithm: Earliest Finish First

- Sort intervals so that $f_1 \leq f_2 \leq \cdots \leq f_n$
- Let S be empty
- For $i = 1, \ldots, n$:
  - If interval i doesn't create a conflict, add i to S
- Return S



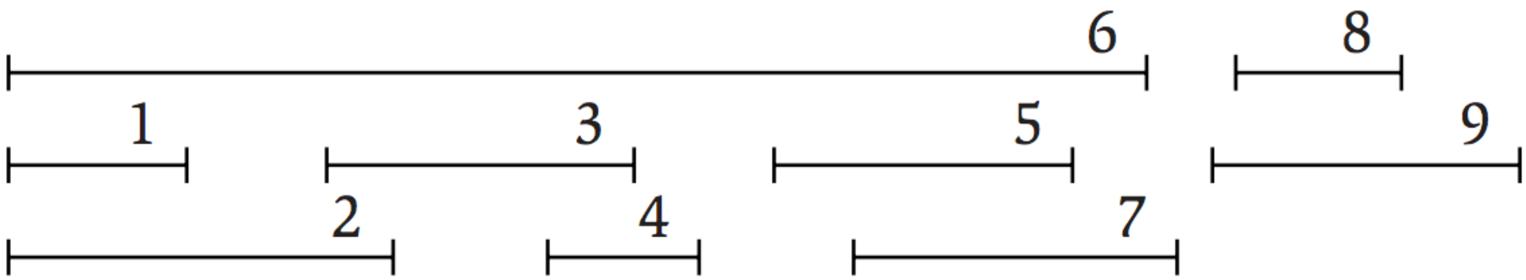
Exchange Argument

- Let $G = \{ i_1, \dots, i_r\}$ be greedy's schedule
- Let $O = \{j_1, \dots , j_s\}$ be some other schedule
- Let k be the first time G and O diverge.
  - $\{i_1, \dots , i_{k-1}\} = \{j_1, \dots , j_{k-1}\}$
  - $i_k \neq j_k$



Exchange Argument

- Let $G = \{ i_1, \ldots, i_r\}$ be greedy's schedule
- Let $O = \{j_1, \ldots, j_s\}$ be some other schedule
- Let k be the first time G and O diverge.
  - $\{i_1, \ldots, i_{k-1}\} = \{j_1, \ldots, j_{k-1}\}$
  - $i_k \neq j_k$
- Exchange j_k for i_k in O.



Greedy Algorithms

a. Interval Scheduling – Greedy Stays Ahead
b. Minimum Lateness Scheduling
c. Interval Scheduling – Exchange Argument

Fractional Knapsack

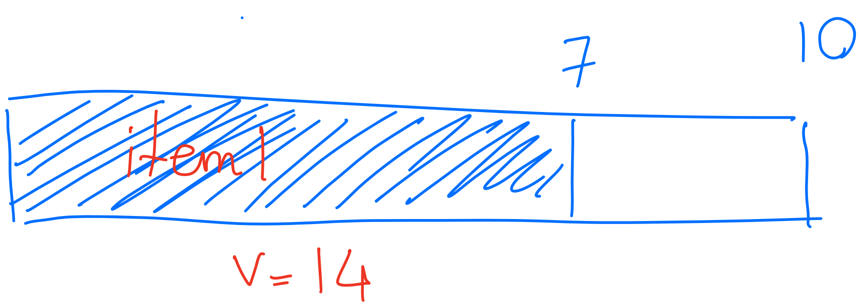
- Like Knapsack, except that every item can be cut or divided into arbitrarily small quantities (e.g., salt, spices)

- Given:
  - n items
  - Item i has weight w_i and value v_i
  - Knapsack has weight limit W

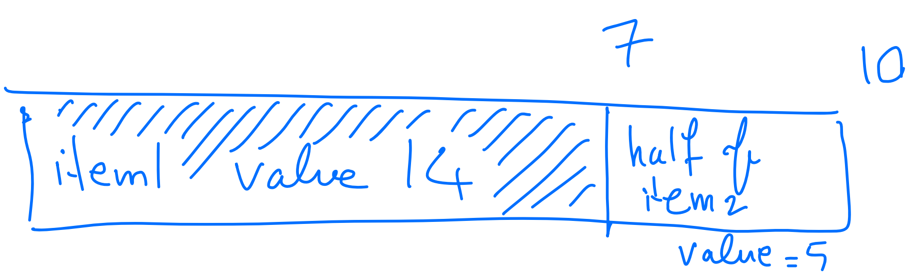
- Goal:
  - Determine (fractions of) items to select, with total weight at most W, so that total value is maximized

Fractional Knapsack: Example

- Capacity (W): 10
- $w_1 = 7,\ v_1 = 14$ $\qquad v_1/w_1 = 2$
- $w_2 = 6,\ v_2 = 10$ $\qquad v_2/w_2 = 1.666$
- $w_3 = 4,\ v_3 = 6$ $\qquad v_3/w_3 = 1.5$



greedy gets value 14
for (non fractional) Knapsack
But {2,3} provide value 16



greedy gets value $14 + \frac{10}{2} = 19$

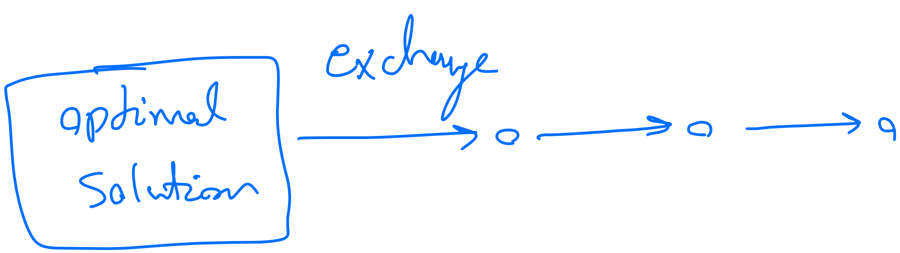
Fractional Knapsack: Greedy Algorithm

- Algorithm:
  - Sort in decreasing order of density = value/weight, and add to knapsack until you cannot fit anymore
  - Possibly using only fraction of final item added

Fractional Knapsack: Greedy Algorithm

- Algorithm:
  - Sort in decreasing order of density = value/weight, and add to knapsack until you cannot fit anymore
  - Possibly using only fraction of final item added

- Proof by Exchange Argument:
  - Suppose $v_1/w_1 > v_2/w_2 > v_3/w_3, \ldots.$
  - Compare GREEDY and another solution say O.



Sort items in O in the decreasing order of densities. If there is an item i whose density is larger than the density of some item j in O and i is not completely in O, we can delete some ε weight of j from O and instead include ε weight of i in O. This improves O since i has higher density than j. This contradicts O being optimal $\Rightarrow$ no such i exists.