Greedy algorithms 1
Scheduling

CS240 Spring 2023

Rui Fan

Greedy algorithms

- Make the best choice at the moment.
  - No planning ahead. “Short-sighted”.
- Once choice made, it’s fixed.
  - No take-backs.
- Cons Doesn’t always find optimal answer.
- Pros Simple and fast. Sometimes optimal.

Overview

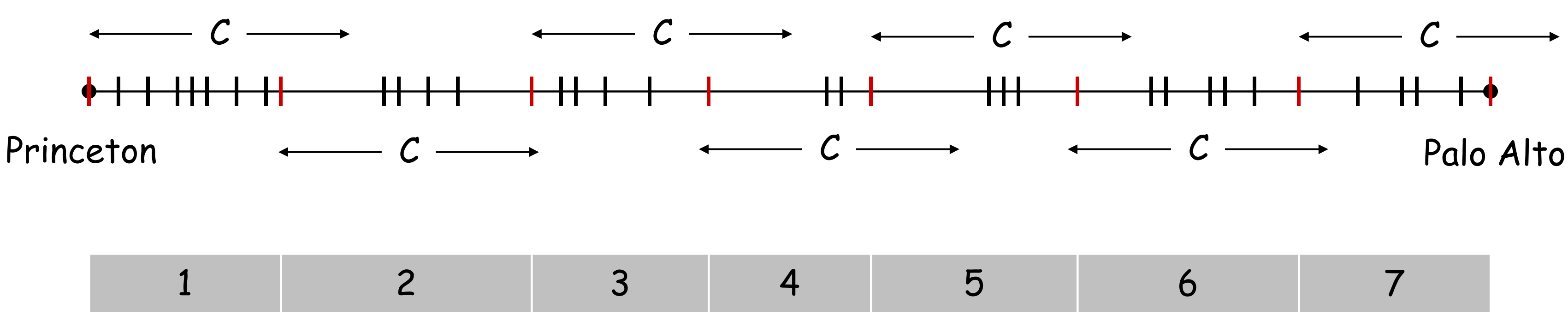
- Selecting breakpoints
- Coin change
- Interval scheduling
- Interval coloring
- Scheduling to minimizing lateness

Selecting Breakpoints

Selecting breakpoints.

- Road trip from Princeton to Palo Alto along fixed route.
- Refueling stations at certain points along the way.
- Fuel capacity = C.
- Goal: makes as few refueling stops as possible.

Greedy algorithm. Go as far as you can before refueling.



Selecting Breakpoints: Greedy Algorithm

Truck driver's algorithm.

```
Sort breakpoints so that: 0 = b0 < b1 < b2 < ... < bn = L

S ← {0}   ← breakpoints selected
x ← 0     ← current location

while (x ≠ bn)
   let p be largest integer such that bp ≤ x + C
   if (bp = x)
      return "no solution"
   x ← bp
   S ← S ∪ {p}
return S
```

Implementation.

O(n log n) to sort.

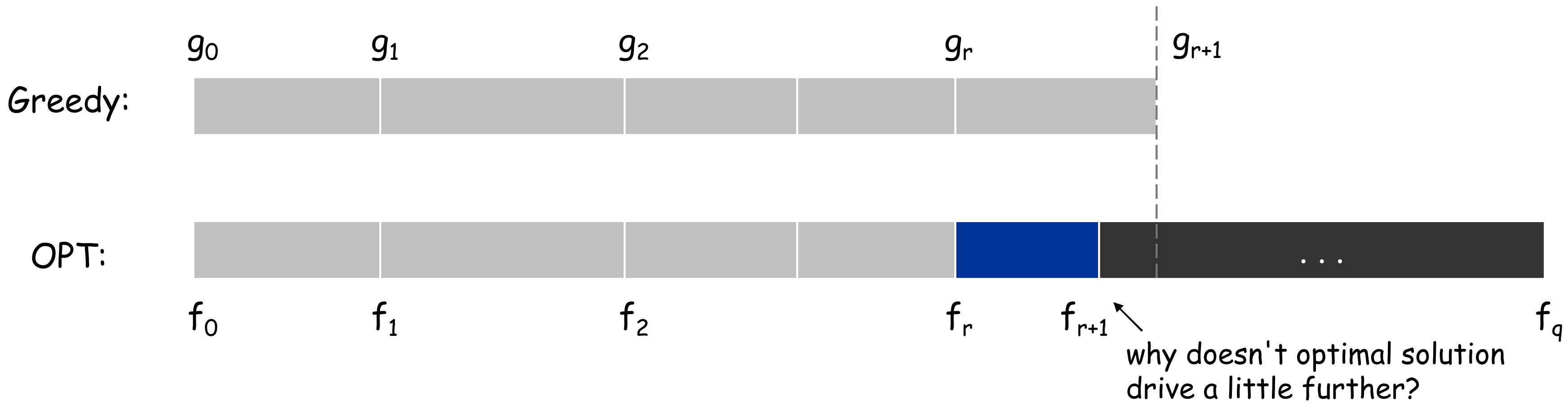
O(n) time for while loop.

Selecting Breakpoints: Correctness

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $0 = g_0 < g_1 < \ldots < g_p = L$ denote set of breakpoints chosen by greedy.
- Let $0 = f_0 < f_1 < \ldots < f_q = L$ denote set of breakpoints in an optimal solution with $f_0 = g_0$, $f_1 = g_1$, . . . , $f_r = g_r$ for largest possible value of r.
- Note: $g_{r+1} > f_{r+1}$ by greedy choice of algorithm.

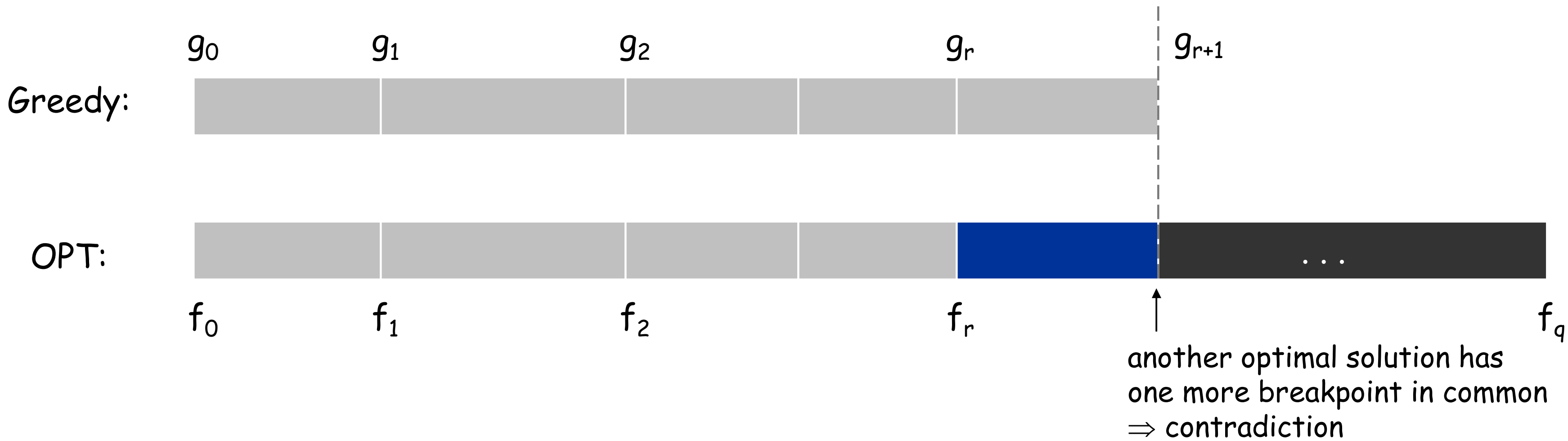


Selecting Breakpoints: Correctness

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $0 = g_0 < g_1 < \ldots < g_p = L$ denote set of breakpoints chosen by greedy.
- Let $0 = f_0 < f_1 < \ldots < f_q = L$ denote set of breakpoints in an optimal solution with $f_0 = g_0, f_1 = g_1, \ldots, f_r = g_r$ for largest possible value of r.
- Note: $g_{r+1} > f_{r+1}$ by greedy choice of algorithm.



Coin Changing

Goal. Given currency denominations: 1, 5, 10, 25, 100 (penny, nickel, dime, quarter, dollar) devise a method to pay amount to customer using fewest number of coins.

Ex: 34¢.

Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

Ex: $2.89.

Coin-Changing: Greedy Algorithm

Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

```
Sort coins denominations by value: c1 < c2 < … < cn.

  coins selected
S ← φ
while (x ≠ 0) {
   let k be largest integer such that ck ≤ x
   if (k = 0)
      return "no solution found"
   x ← x - ck
   S ← S ∪ {k}
}
return S
```

Q. Is cashier's algorithm optimal?

Coin-Changing: Analysis of Greedy Algorithm

Theorem. Greed is optimal for U.S. coinage: 1, 5, 10, 25, 100.

Pf. (by induction on x)

- Consider optimal way to change $c_k \leq x < c_{k+1}$: greedy takes coin k.
- We claim that any optimal solution must also take coin k.
  - if not, it needs enough coins of type $c_1, \ldots, c_{k-1}$ to add up to x
  - table below indicates no optimal solution can do this
- Problem reduces to coin-changing x - c_k cents, which, by induction, is optimally solved by greedy algorithm. ▪

k	c_k	All optimal solutions must satisfy	Max value of coins 1, 2, …, k-1 in any OPT
1	1	P ≤ 4	-
2	5	N ≤ 1	4
3	10	N + D ≤ 2	4 + 5 = 9
4	25	Q ≤ 3	20 + 4 = 24
5	100	no limit	75 + 24 = 99

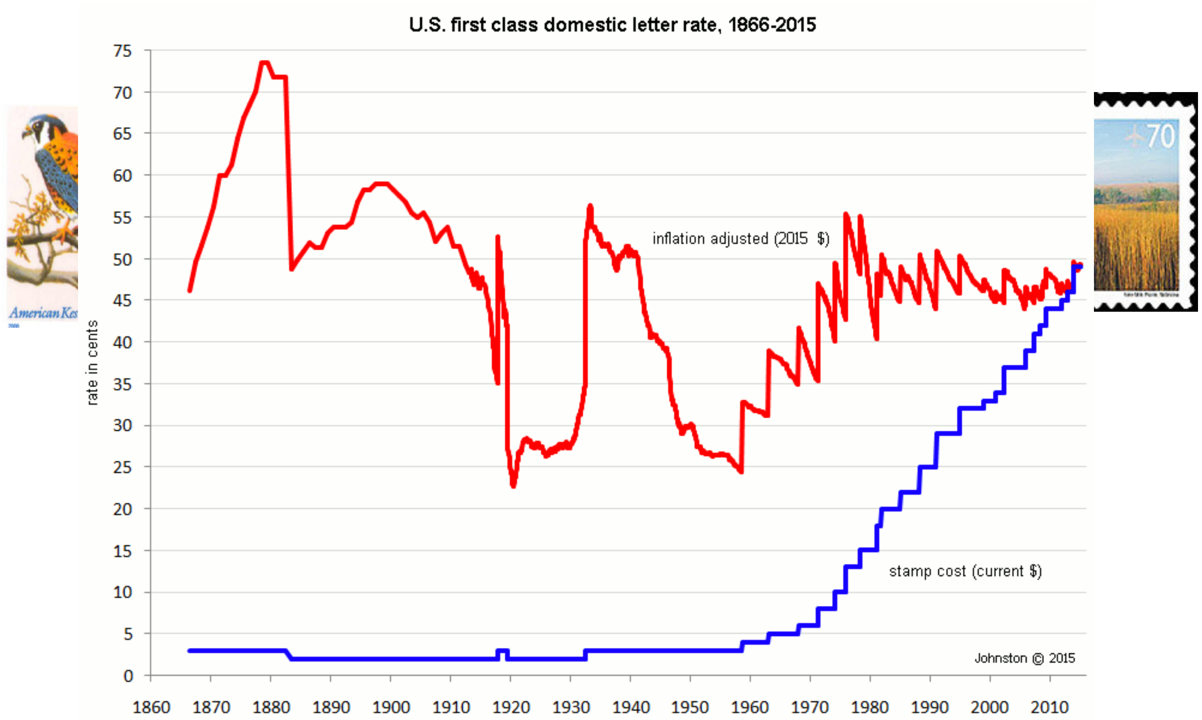
- Opt must have $p \leq 4$, because if $p \geq 5$, can replace 5 p with 1 n.
- Opt must have $n \leq 1$, because if $n \geq 2$, can replace with one dime, etc.
- So if don't use c_k, then must use $c_1, \ldots, c_{k-1}$, and these must add up to $\geq x \geq c_k$.
- By case analysis, we see coins of type $c_1, \ldots, c_{k-1}$ never add up to $\geq c_k$.

Coin-Changing: Analysis of Greedy Algorithm

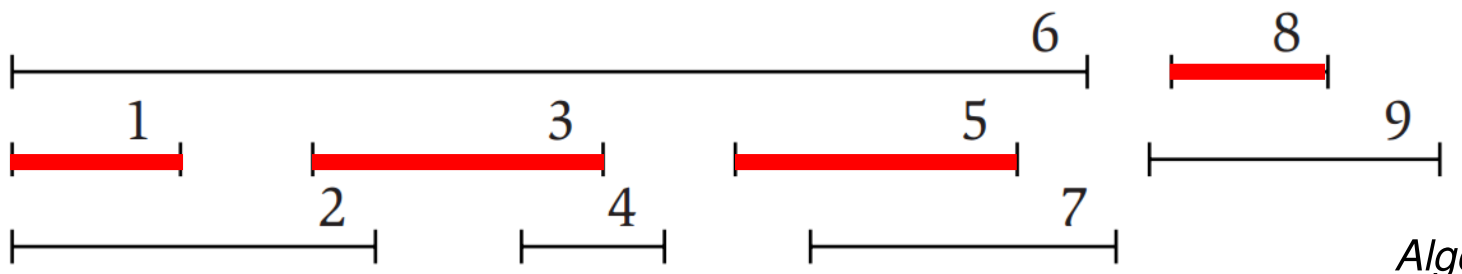
Observation. Greedy algorithm is sub-optimal for US postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

Counterexample. 140¢.

- Greedy: 100, 34, 1, 1, 1, 1, 1, 1.
- Optimal: 70, 70.



Interval scheduling



Algorithm Design
Kleinberg, Tardos

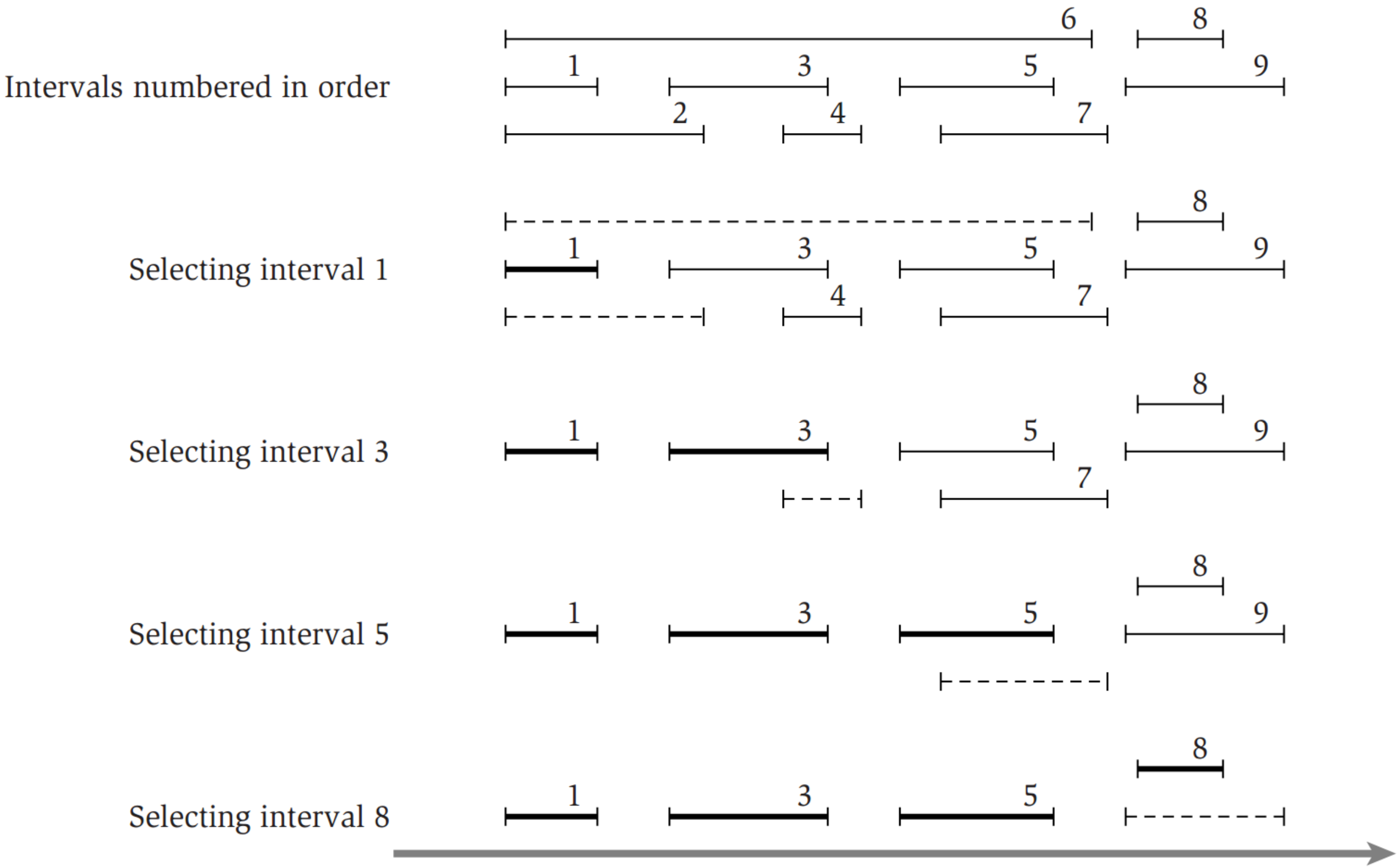
- Given a set of intervals, pick the largest number of nonoverlapping ones.
  - Each interval given by a start and finishing time.
- Models use of a shared resource.
  - Ex 9 people want to use a room. Different people want to use it at different times. Let max number of people use room.

A greedy algorithm

- Let's pick the intervals from left to right.
- Intuition Since can't pick a new interval until the previous one ends, want to pick intervals that end as quickly as possible.
- So we sort the intervals by finishing times. Then keep selecting earliest finishing one that doesn't overlap previous selected interval.

Initially let R be the set of all requests, and let A be empty
While R is not yet empty
  Choose a request $i \in R$ that has the smallest finishing time
  Add request i to A
  Delete all requests from R that are not compatible with request i
EndWhile
Return the set A as the set of accepted requests

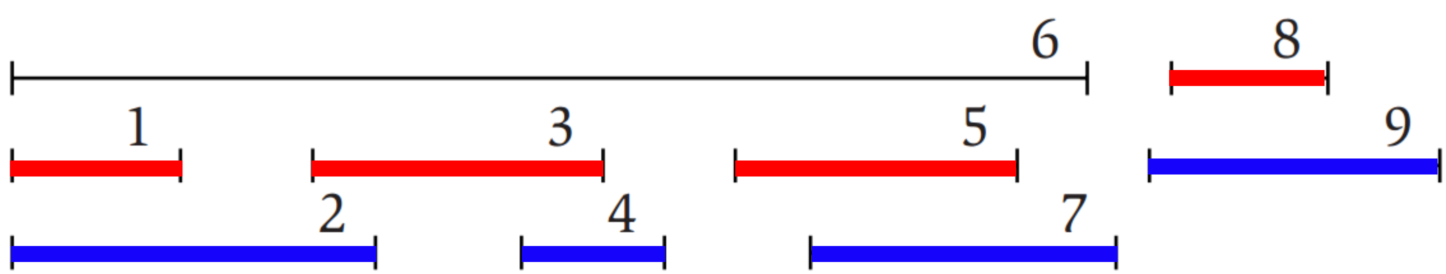
A greedy algorithm



Algorithm Design
Kleinberg, Tardos

Correctness

- We'll compare algorithm's solution S to an optimal solution T.
  - Can have $S \neq T$, since there may be several optimal solutions.
- Def Let s_k and t_k be k'th interval in S and T, resp.
- Def let fin(i) be finishing time of an interval i.
- Claim $\text{fin}(s_k) \leq \text{fin}(t_k)$ for all k.
- Proof By induction. True for k=1 since s_k is interval with min fin time.
  - Suppose true for < k, i.e. $\text{fin}(s_{k-1}) \leq \text{fin}(t_{k-1})$.
  - Then {intervals not intersecting s_{k-1} and finishing after s_{k-1}} $\supseteq$ {intervals not intersecting t_{k-1} and finishing after t_{k-1}}.
  - By the algorithm, s_k is earliest finishing interval in the first set.
  - t_k is some interval in the latter set.
  - So $\text{fin}(s_k) \leq \text{fin}(t_k)$.
- Corollary S has at least as many intervals as T, i.e. S is optimal.



algorithm's solution

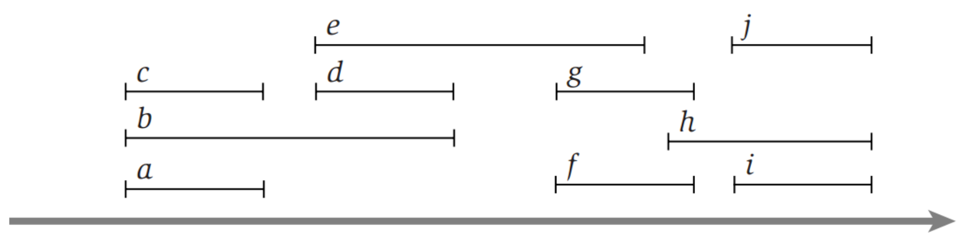
an optimal solution

Analysis

- If there are n intervals, sorting the intervals takes O(n log n) time.
- Then we go through the intervals in order of finishing times.
  - For each interval, check if it intersects last selected one, and select it if it doesn't.
  - Takes O(n) time.
- Total O(n log n) time.

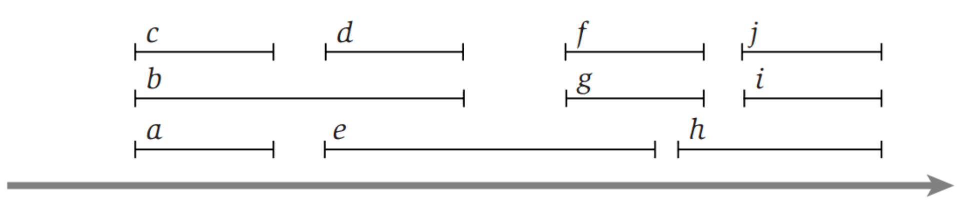
Interval coloring problem

- In interval scheduling, we scheduled the max number of intervals on one resource.
- In interval coloring, we need to schedule all the intervals on some number of resources.
  - Intervals on the same resource cannot overlap.
  - Goal Minimize the number of resources used.
- Example application is to schedule people who need to use a room in the min number of rooms.



(a)

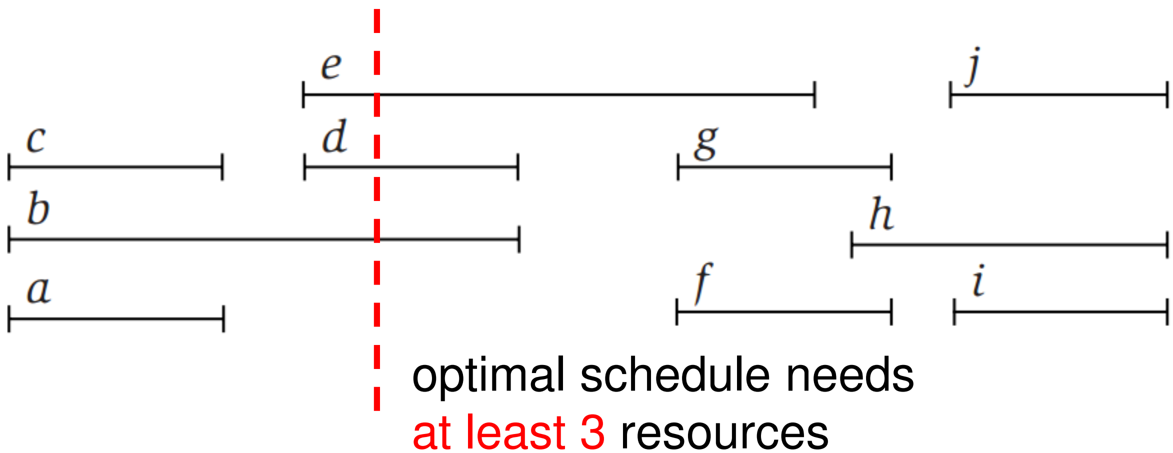
All intervals scheduled using 4 resources



All intervals scheduled using 3 resources

Optimality criterion

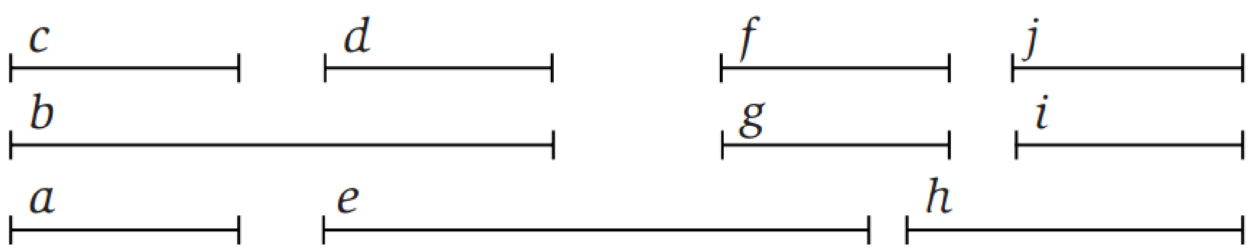
- Observation Suppose k intervals intersect at some time point. Then the optimal schedule needs at least k resources.
  - Since intervals on same resource can't overlap, then the k intersecting intervals need to be assigned to k different resources in any solution.
- Def Depth of a set of intervals is the max number of intervals that intersect at any time.



These intervals have depth 3

Optimality criterion

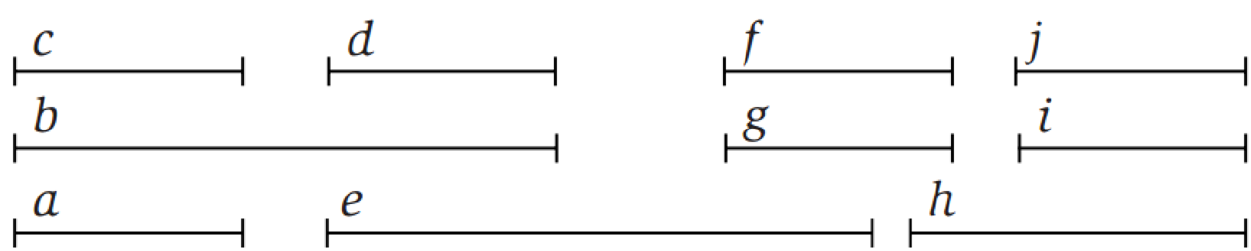
- Corollary Let d be the depth of a set of intervals, and suppose we find a schedule using d resources. Then the schedule is optimal.
  - By the observation, any schedule needs at least d resources. Since our schedule uses d resources, it's optimal.



Since depth is 3, this schedule is optimal

A greedy algorithm

- Sweep through intervals in order of increasing start time.
  - Break ties arbitrarily.
- For each interval, assign it to smallest resource not already assigned to an intersecting interval.



Correctness

- Claim Let the set of intervals have depth d. Then the algorithm uses d resources.
- Proof Suppose algorithm is processing some interval s. Then $\leq$ d-1 intervals intersect s.
  - So $\leq$ d-1 resources assigned to these intervals.
  - So s can be assigned some resource $\leq$ d.
- Corollary The algorithm is optimal
  - Follows by the optimality criterion.

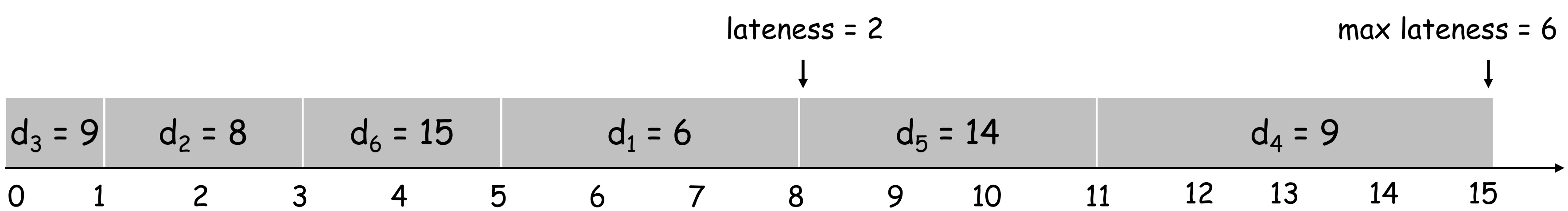
Scheduling to Minimizing Lateness

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

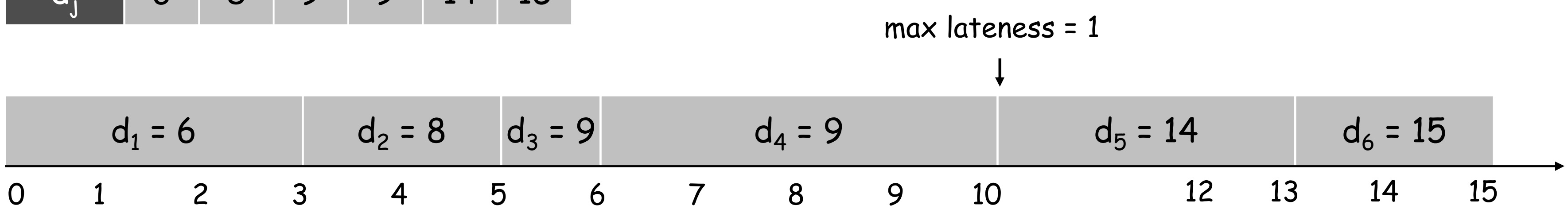
counterexample

Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

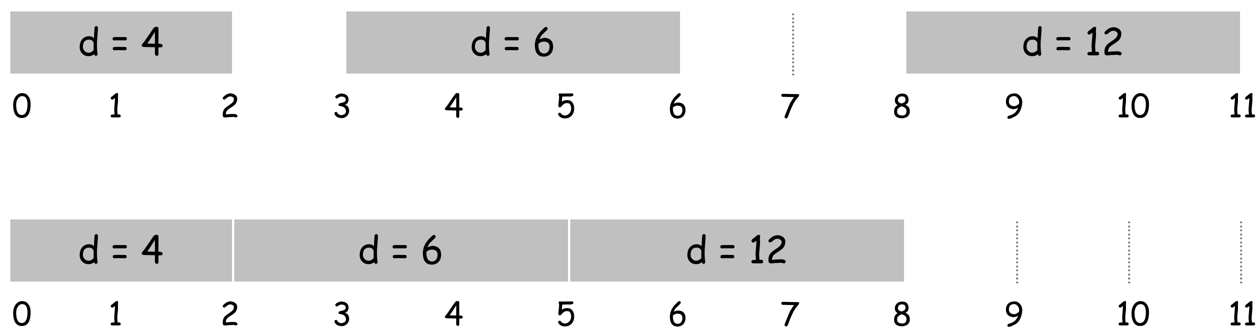
```
Sort n jobs by deadline so that d1 ≤ d2 ≤ … ≤ dn

t ← 0
for j = 1 to n
   Assign job j to interval [t, t + tj]
   sj ← t, fj ← t + tj
   t ← t + tj
output intervals [sj, fj]
```

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



Minimizing Lateness: No Idle Time

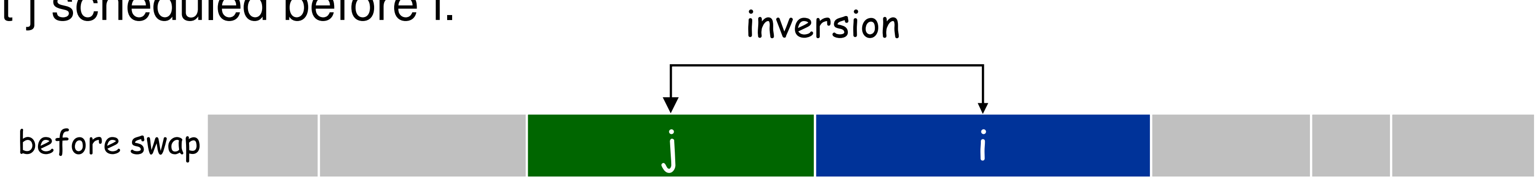
Observation. There exists an optimal schedule with no idle time.



Observation. The greedy schedule has no idle time.

Minimizing Lateness: Inversions

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

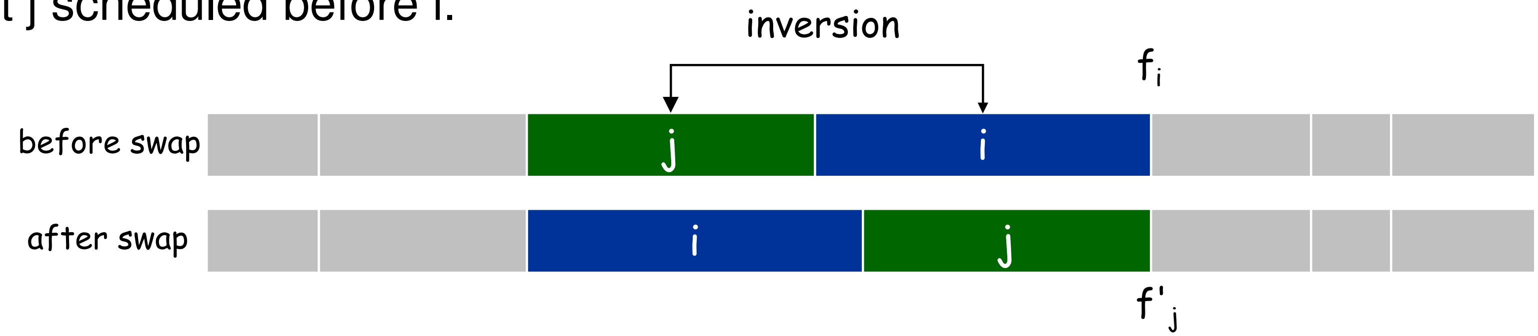


Observation. Greedy schedule has no inversions.

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



Claim. Swapping two adjacent, 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* ▪