

## CSCI 355: ALGORITHM DESIGN AND ANALYSIS

### 3. GREEDY ALGORITHMS I

- ▶ coin changing
- ▶ interval scheduling
- ▶ interval partitioning
- ▶ scheduling to minimize lateness



#### Coin changing

**Goal.** Given Canadian currency denominations { 1, 5, 10, 25, 100 }, devise a method to pay amount to customer using fewest 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.



3

#### Cashier's algorithm

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

**CASHIERS-ALGORITHM** ( $x, c_1, c_2, \dots, c_n$ )

**SORT**  $n$  coin denominations so that  $0 < c_1 < c_2 < \dots < c_n$ .

$S \leftarrow \emptyset$ . ← multiset of coins selected

**WHILE** ( $x > 0$ )

$k \leftarrow$  largest coin denomination  $c_k$  such that  $c_k \leq x$ .

**IF** (no such  $k$ )

**RETURN** "no solution."

**ELSE**

$x \leftarrow x - c_k$ .

$S \leftarrow S \cup \{ k \}$ .

**RETURN**  $S$ .

4

### Cashier's algorithm (for arbitrary coin denominations)

Q. Is cashier's algorithm optimal for any set of denominations?

A. No. Consider Canadian postage: { 1, 10, 22, 34, 71, 100, 250 }.

- Cashier's algorithm:  $142\text{¢} = 100 + 34 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1$
- Optimal:  $142\text{¢} = 71 + 71$



Remark. It may not even lead to a feasible solution if  $c_1 > 1: 7, 8, 9$ .

- Cashier's algorithm:  $15\text{¢} = 9 + ?$
- Optimal:  $15\text{¢} = 7 + 8$

### Properties of any optimal solution (for Canadian coin denominations)

**Property.** Number of pennies  $\leq 4$ .

**Pf.** Replace 5 pennies with 1 nickel.

**Property.** Number of nickels  $\leq 1$ .

**Pf.** Replace 2 nickels with 1 dime.

**Property.** Number of quarters  $\leq 3$ .

**Pf.** Replace 4 quarters with 1 dollar.

**Property.** Number of nickels + number of dimes  $\leq 2$ .

**Pf.**

- Recall: we need at most 1 nickel.
- Replace 3 dimes and 0 nickels with 1 quarter and 1 nickel;
- Replace 2 dimes and 1 nickel with 1 quarter.



dollars  
(100¢)

quarters  
(25¢)

dimes  
(10¢)

nickels  
(5¢)

pennies  
(1¢)

### Optimality of cashier's algorithm (for Canadian coin denominations)

**Theorem.** Cashier's algorithm is optimal for Canadian coins { 1, 5, 10, 25, 100 }.

**Pf.** [ by induction on amount to be paid  $x$  ]

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

$k$	$c_k$	all optimal solutions must satisfy	max value of coin denominations $c_1, c_2, \dots, c_{k-1}$ in any optimal solution
1	1	$P \leq 4$	-
2	5	$N \leq 1$	4
3	10	$N + D \leq 2$	$4 + 5 = 9$
4	25	$Q \leq 3$	$20 + 4 = 24$
5	100	no limit	$75 + 24 = 99$

## Room for improvement?

- Using currency denominations { 1, 5, 10, 25 }, any transaction up to \$1 results in 4.7 coins being returned on average.
- If we replace the 10¢ coin with an 18¢ coin, then 3.89 coins are returned on average!
- Using currency denominations { 1, 5, 10, 25, 100, 200 }, any transaction up to \$5 results in 5.9 coins being returned on average.
- If we add an 83¢ coin, then 4.578 coins are returned on average!



Ray Kuo

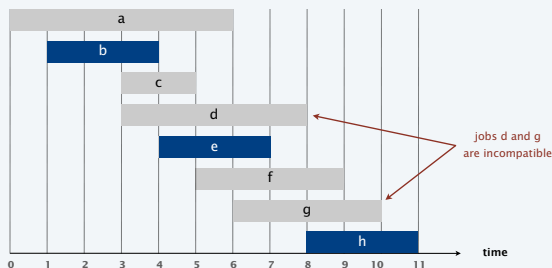


## CSCI 355: ALGORITHM DESIGN AND ANALYSIS 3. GREEDY ALGORITHMS I

- coin changing
- interval scheduling
- interval partitioning
- scheduling to minimize lateness

### Interval scheduling

- Job  $j$  starts at time  $s_j$  and finishes at time  $f_j$ .
- Two jobs are **compatible** if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



## Interval scheduling: earliest-finish-time-first algorithm



```

EARLIEST-FINISH-TIME-FIRST ( $n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n$ )
SORT jobs by finish times and renumber so that  $f_1 \leq f_2 \leq \dots \leq f_n$ .
 $S \leftarrow \emptyset$ .  $\leftarrow$  set of jobs selected
FOR  $j = 1$  TO  $n$ 
  IF (job  $j$  is compatible with  $S$ )
     $S \leftarrow S \cup \{j\}$ .
RETURN  $S$ .
    
```

**Proposition.** We can implement earliest-finish-time first in  $O(n \log n)$  time.

- Keep track of the job  $j^*$  that was added last to  $S$ .
- Job  $j$  is compatible with  $S$  iff  $s_j \geq f_{j^*}$ .
- Sorting by finish times takes  $O(n \log n)$  time.

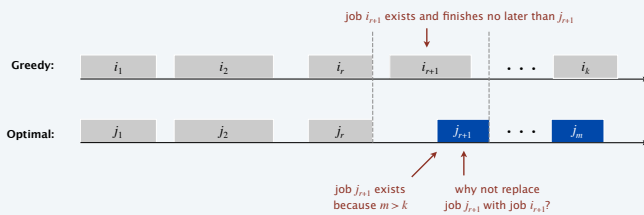
13

## Interval scheduling: analysis of earliest-finish-time-first algorithm

**Theorem.** The earliest-finish-time-first algorithm is optimal.

**Pf.** [by contradiction]

- Assume the greedy earliest-finish-time approach is not optimal, and see what happens.
- Let  $i_1, i_2, \dots, i_k$  denote the set of jobs selected by the greedy earliest-finish-time approach.
- Let  $j_1, j_2, \dots, j_m$  denote the set of jobs in an optimal solution with  $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$  for the largest possible value of  $r$ .



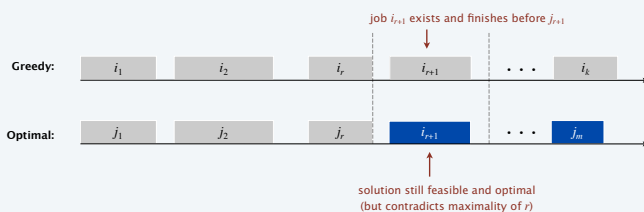
14

## Interval scheduling: analysis of earliest-finish-time-first algorithm

**Theorem.** The earliest-finish-time-first algorithm is optimal.

**Pf.** [by contradiction]

- Assume the greedy earliest-finish-time approach is not optimal, and see what happens.
- Let  $i_1, i_2, \dots, i_k$  denote the set of jobs selected by the greedy earliest-finish-time approach.
- Let  $j_1, j_2, \dots, j_m$  denote the set of jobs in an optimal solution with  $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$  for the largest possible value of  $r$ .



15

# CSCI 355: ALGORITHM DESIGN AND ANALYSIS

## 3. GREEDY ALGORITHMS I

- coin changing
- interval scheduling
- **interval partitioning**
- scheduling to minimize lateness

---

---

---

---

---

---

---

---

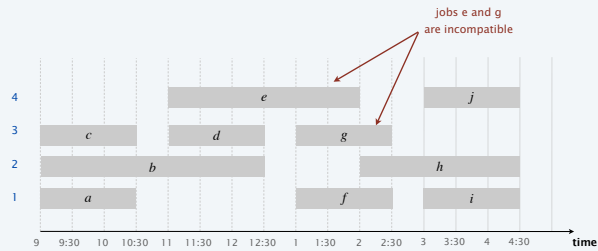
---

---

### Interval partitioning

- Lecture  $j$  starts at time  $s_j$  and finishes at time  $f_j$ .
- Goal: find the minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.

Ex. This schedule uses 4 classrooms to schedule 10 lectures.



---

---

---

---

---

---

---

---

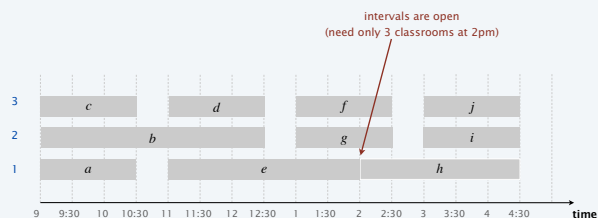
---

---

### 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 lectures occur at the same time in the same room.

Ex. This schedule uses 3 classrooms to schedule 10 lectures.



---

---

---

---

---

---

---

---

---

---

### Interval partitioning: earliest-start-time-first algorithm



```
EARLIEST-START-TIME-FIRST ( $n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n$ )
SORT lectures by start times and renumber so that  $s_1 \leq s_2 \leq \dots \leq s_n$ .
 $d \leftarrow 0$ .  $\leftarrow$  number of allocated classrooms
FOR  $j = 1$  TO  $n$ 
  IF (lecture  $j$  is compatible with some classroom)
    Schedule lecture  $j$  in any such classroom  $k$ .
  ELSE
    Allocate a new classroom  $d + 1$ .
    Schedule lecture  $j$  in classroom  $d + 1$ .
     $d \leftarrow d + 1$ .
RETURN schedule.
```

21

---

---

---

---

---

---

---

---

---

---

### Interval partitioning: earliest-start-time-first algorithm

**Proposition.** We can implement earliest-start-time-first in  $O(n \log n)$  time.

**Pf.**

- Sorting by start times takes  $O(n \log n)$  time.
- Store classrooms in a priority queue (key = finish time of its last lecture).
  - To allocate a new classroom, INSERT classroom onto priority queue.
  - To schedule lecture  $j$  in classroom  $k$ , INCREASE-KEY of classroom  $k$  to  $f_j$ .
  - To determine whether lecture  $j$  is compatible with any classroom, compare  $s_j$  to FIND-MIN
- Total number of priority queue operations is  $O(n)$ ; each takes  $O(\log n)$  time. ▀

**Remark.** This implementation chooses a classroom  $k$  whose finish time of its last lecture is the **earliest**.

22

---

---

---

---

---

---

---

---

---

---

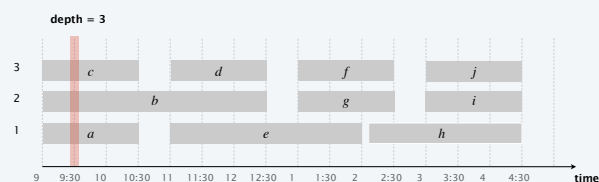
### Interval partitioning: lower bound on optimal solution

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

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

**Q.** Is the minimum number of classrooms needed always equal to the depth?

**A.** Yes! Moreover, the earliest-start-time-first algorithm finds a schedule where the number of classrooms is equal to the depth.



23

---

---

---

---

---

---

---

---

---

---

### Interval partitioning: analysis of earliest-start-time-first algorithm

**Observation.** The earliest-start-time-first algorithm never schedules two incompatible lectures in the same classroom.

**Theorem.** Earliest-start-time-first algorithm is optimal.

**Pf.**

- Let  $d$  = number of classrooms that the earliest-start-time-first algorithm allocates.
- Classroom  $d$  was opened because we needed to schedule a lecture, say  $j$ , that was incompatible with a lecture in each of the  $d - 1$  other classrooms.
- Thus, these  $d - 1$  lectures all end after  $s_j$ .
- Since we sorted by start time, each of these incompatible lectures start no later than  $s_j$ .
- Thus, we have  $d$  lectures overlapping at time  $s_j + \epsilon$ .
- Key observation  $\Rightarrow$  all schedules use  $\geq d$  classrooms. •

24

### CSCI 355: ALGORITHM DESIGN AND ANALYSIS 3. GREEDY ALGORITHMS I

- coin changing
- interval scheduling
- interval partitioning
- scheduling to minimize lateness

### Scheduling to minimize lateness

- A single resource processes one job at a time.
- Each job  $j$  requires  $t_j$  units of processing time and is due at time  $d_j$ .
- If job  $j$  starts at time  $s_j$ , it finishes at time  $f_j = s_j + t_j$ .
- The lateness of a job  $j$  is  $\ell_j = \max\{0, f_j - d_j\}$ .
- Goal: schedule all jobs to minimize the **maximum** lateness  $L = \max_j \ell_j$ .

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



26

## Minimizing lateness: earliest-deadline-first

EARLIEST-DEADLINE-FIRST ( $n, t_1, t_2, \dots, t_n, d_1, d_2, \dots, d_n$ )

**Sort** jobs by due times and renumber so that  $d_1 \leq d_2 \leq \dots \leq d_n$ .

$t \leftarrow 0$ .

**FOR**  $j = 1$  **TO**  $n$

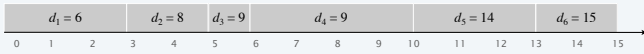
  Assign job  $j$  to interval  $[t, t + t_j]$ .

$s_j \leftarrow t$ ;  $f_j \leftarrow t + t_j$ .

$t \leftarrow t + t_j$ .

**RETURN** intervals  $[s_1, f_1], [s_2, f_2], \dots, [s_n, f_n]$ .

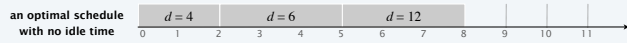
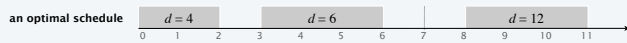
max lateness  $L = 1$



28

## Minimizing lateness: no idle time

**Observation 1.** There exists an optimal schedule with no idle time.

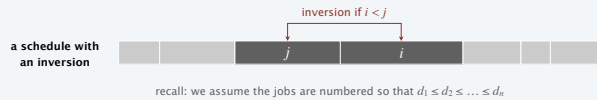


**Observation 2.** The earliest-deadline-first schedule has no idle time.

29

## Minimizing lateness: inversions

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



**Observation 3.** The earliest-deadline-first schedule is the unique idle-free schedule with no inversions.

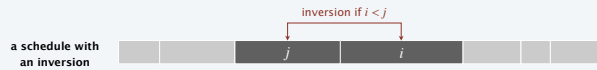


30



## Minimizing lateness: inversions

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

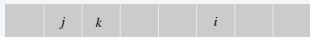


recall: we assume the jobs are numbered so that  $d_1 \leq d_2 \leq \dots \leq d_n$

**Observation 4.** If an idle-free schedule has an inversion, then it has an adjacent inversion.

**Pf.** ← two inverted jobs scheduled consecutively

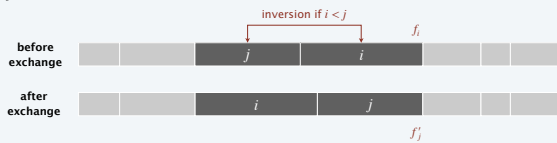
- Let  $i-j$  be a closest inversion.
- Let  $k$  be the element immediately to the right of  $j$ .
- Case 1. [ $j > k$ ] Then  $j-k$  is an adjacent inversion.
- Case 2. [ $j < k$ ] Then  $i-k$  is a closer inversion since  $i < j < k$ . \*



31

## Minimizing lateness: inversions

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



**Key claim.** Exchanging two adjacent inverted jobs  $i$  and  $j$  reduces the number of inversions by 1 and does not increase the max lateness.

**Pf.** Let  $\ell$  be the lateness before the swap, and let  $\ell'$  be the lateness afterwards.

- $\ell'_k = \ell_k$  for all  $k \neq i, j$ .
- $\ell'_i \leq \ell_i$ .
- If job  $j$  is late,  $\ell'_j = f'_j - d_j$  ← definition
  - $= f_i - d_j$  ←  $j$  now finishes at time  $f_i$
  - $\leq f_i - d_i$  ←  $i < j \Rightarrow d_i \leq d_j$
  - $\leq \ell_i$ . ← definition

32

## Minimizing lateness: analysis of earliest-deadline-first algorithm

**Theorem.** The earliest-deadline-first schedule  $S$  is optimal.

**Pf.** [by contradiction]

Define  $S^*$  to be an optimal schedule with the fewest inversions.

- We can assume  $S^*$  has no idle time. ← Observation 1
- Case 1. [ $S^*$  has no inversions] Then  $S = S^*$ . ← Observation 3
- Case 2. [ $S^*$  has an inversion]
  - Let  $i-j$  be an adjacent inversion. ← Observation 4
  - Exchanging jobs  $i$  and  $j$  decreases the number of inversions by 1 without increasing the max lateness. ← key claim
  - This contradicts the "fewest inversions" part of the definition of  $S^*$ . \*

← optimal schedule can have inversions

33

## 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.

**Structure.** 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.** Gale–Shapley, Kruskal, Prim, Dijkstra, Huffman, ...



---

---

---

---

---

---

---

---

---

---