Intro: Coin Changing
Coin Changing

Goal. Given currency denominations: 1, 5, 10, 25, 100, give change to customer using \textit{fewest} number of coins.

Ex: 34¢.

Cashier's algorithm. At each iteration, give the \textit{largest} coin valued \leq the amount to be paid.

Ex: $2.89.

Algorithm is "Greedy": One large coin better than two or more smaller ones.
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.

Algorithm is “Greedy”, but also short-sighted – attractive choice now may lead to dead ends later.

Correctness is key!
Outline & Goals

“Greedy Algorithms”
  what they are

Pros
  intuitive
  often simple
  often fast

Proof techniques
  stay ahead
  structural
  exchange arguments
Plan

Greed

Greeeed

Greeeeded

Greeeeeedeeded
4.1 Interval Scheduling

Proof Technique 1: “greedy stays ahead”
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.
Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

- What order?
- Does that give best answer?
- Why or why not?
- Does it help to be greedy about order?
Interval Scheduling: Greedy Algorithms

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

[Earliest start time] Consider jobs in ascending order of start time $s_j$.

[Earliest finish time] Consider jobs in ascending order of finish time $f_j$.

[Shortest interval] Consider jobs in ascending order of interval length $f_j - s_j$.

[Fewest conflicts] For each job, count the number of conflicting jobs $c_j$. Schedule in ascending order of conflicts $c_j$. 
Greedy template. Consider jobs in some 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 Algorithms

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

- **[Earliest start time]** Consider jobs in ascending order of start time $s_j$.

- **[Earliest finish time]** Consider jobs in ascending order of finish time $f_j$.

- **[Shortest interval]** Consider jobs in ascending order of interval length $f_j - s_j$.

- **[Fewest conflicts]** For each job, count the number of conflicting jobs $c_j$. Schedule in ascending order of conflicts $c_j$. 


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 $f_1 \leq f_2 \leq \ldots \leq f_n$.

```
A = {}
for j = 1 to n {
    if (job j compatible with A)
        A = A U {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

![Interval Scheduling Diagram](image_url)
Interval Scheduling
Interval Scheduling
Interval Scheduling

Time

0 1 2 3 4 5 6 7 8 9 10 11

B
A
C
E
D
F
G
H

B A
Interval Scheduling

- A
- B
- C
- D
- E
- F
- G
- H

Time
Interval Scheduling

![Interval Scheduling Diagram]
Interval Scheduling
Interval Scheduling

Time

0 1 2 3 4 5 6 7 8 9 10 11

B
C
A
E
D
F
G
H
Interval Scheduling: Correctness

Theorem. Greedy algorithm is optimal.

Pf. (“greedy stays ahead”)

Let \( i_1, i_2, \ldots, i_k \) be jobs picked by greedy, \( j_1, j_2, \ldots, j_m \) those in some optimal solution

Show \( f(i_r) \leq f(j_r) \) by induction on \( r \).

Basis: \( i_1 \) chosen to have min finish time, so \( f(i_1) \leq f(j_1) \)

Ind: \( f(i_r) \leq f(j_r) \leq s(j_{r+1}) \), so \( j_{r+1} \) is among the candidates considered by greedy when it picked \( i_{r+1} \), & it picks min finish, so \( f(i_{r+1}) \leq f(j_{r+1}) \)

Similarly, \( k \geq m \), else \( j_{k+1} \) is among (nonempty) set of candidates for \( i_{k+1} \)
4.1 Interval Partitioning

Proof Technique 2: “Structural”
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 as Interval Graph Coloring

Vertices = classes;
edges = conflicting class pairs;
different colors = different assigned rooms

Note: graph coloring is very hard in general, but graphs corresponding to interval intersections are a much simpler special case.
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 Scheduling: Greedy Algorithms

Greedy template. Consider lectures in some order. If next lecture fits in the schedule we have, add it to one of the classrooms, otherwise open a new classroom.

[Earliest start time] Consider lectures in ascending order of start time $s_j$.

[Earliest finish time] Consider lectures in ascending order of finish time $f_j$.

[Shortest interval] Consider lectures in ascending order of interval length $f_j - s_j$.

[Fewest conflicts] For each lecture, count the number of conflicting lectures $c_j$. Schedule in ascending order of conflicts $c_j$. 
Interval Scheduling: Greedy Algorithms

counterexample for earliest finish time

counterexample for shortest interval

counterexample for fewest conflicts
**Interval Scheduling: Greedy Algorithms**

**Greedy template.** Consider lectures in some order. If next lecture fits in the schedule we have, add it to one of the classrooms, otherwise open a new classroom.

- **[Earliest start time]** Consider lectures in ascending order of start time $s_j$.

- **[Earliest finish time]** Consider lectures in ascending order of finish time $f_j$.

- **[Shortest interval]** Consider lectures in ascending order of interval length $f_j - s_j$.

- **[Fewest conflicts]** For each lecture, count the number of conflicting lectures $c_j$. Schedule in ascending order of conflicts $c_j$. 
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 \( s_1 \leq s_2 \leq \ldots \leq s_n \).

\[
d = 0 \quad \text{number of allocated classrooms}
\]

for \( j = 1 \) to \( n \) {
    if (lect \( j \) is compatible with some classroom \( k, 1 \leq k \leq d \))
        schedule lecture \( j \) in classroom \( k \)
    else
        allocate a new classroom \( d + 1 \)
        schedule lecture \( j \) in classroom \( d + 1 \)
        \( d = d + 1 \)
}

Implementation? Run-time?
Exercises
Interval Partitioning

Room 1

Time

9:30 10 10:30 11 11:30 12 12:30 1 1:30 2 2:30 3 3:30 4 4:30

c b d e g f h j i
Interval Partitioning
Interval Partitioning
Interval Partitioning

Room 3

Room 2

Room 1

<table>
<thead>
<tr>
<th>Time</th>
<th>9</th>
<th>9:30</th>
<th>10</th>
<th>10:30</th>
<th>11</th>
<th>11:30</th>
<th>12</th>
<th>12:30</th>
<th>1</th>
<th>1:30</th>
<th>2</th>
<th>2:30</th>
<th>3</th>
<th>3:30</th>
<th>4</th>
<th>4:30</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c</td>
<td>b</td>
<td>d</td>
<td></td>
<td>e</td>
<td></td>
<td>f</td>
<td></td>
<td>g</td>
<td></td>
<td>h</td>
<td></td>
<td>i</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- Room 1
- Room 2
- Room 3

Legend:
- c
- d
- e
- f
- g
- h
- i
- j
Interval Partitioning

Room 1

Room 2

Room 3
Interval Partitioning

Room 1

Room 2

Room 3

Time
Interval Partitioning: A “Structural” 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 ≥ depth.

Ex: Depth of schedule below = 3 ⇒ schedule below is optimal.

Q. Does there always exist a schedule equal to depth of intervals?
Interval Partitioning: Greedy Analysis

**Theorem.** Greedy algorithm is optimal.

**Pf (exploit structural property).**
- 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 \( d-1 \) previously used classrooms.
- 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 \( s_j \), i.e. depth \( \geq d \).
- “Key observation” all schedules use \( \geq \) depth classrooms, so \( d = \) depth and greedy is optimal.