Greedy Algorithms

Lecture 19
Tuesday, November 3, 2020
19.1
Greedy algorithms by example
Greedy algorithms

Why don’t you do right?

1 greedy algorithms: do locally the right thing...

...and they suck.

Problem: VertexCoverMin

<table>
<thead>
<tr>
<th>Instance:</th>
<th>Vertex Cover!Minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question:</td>
<td>A graph $G$.</td>
</tr>
</tbody>
</table>

Return the smallest subset $S \subseteq V(G)$, s.t. $S$ touches all the edges of $G$.

2 GreedyVertexCover: pick vertex with highest degree, remove, repeat.
Greedy algorithms
Why don’t you do right?

1 greedy algorithms: do locally the right thing...

2 ...and they suck.

Problem: VertexCoverMin

<table>
<thead>
<tr>
<th>Instance:</th>
<th>Vertex Cover!Minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question:</td>
<td>A graph $G$.</td>
</tr>
</tbody>
</table>

Return the smallest subset $S \subseteq V(G)$, s.t. $S$ touches all the edges of $G$.

GreedyVertexCover: pick vertex with highest degree, remove, repeat.
Greedy algorithms

Why don’t you do right?

1. **greedy algorithms**: do locally the right thing...

2. ...and they suck.

   **Problem: VertexCoverMin**

   **Instance**: Vertex Cover!Minimization
   **Question**: A graph $G$.

   Return the smallest subset $S \subseteq V(G)$, s.t. $S$ touches all the edges of $G$.

3. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.
Greedy algorithms

Why don’t you do right?

1. **greedy algorithms**: do locally the right thing...

2. ...and they suck.

**Problem: VertexCoverMin**

**Instance:** Vertex Cover!Minimization

**Question:** A graph $G$.

Return the smallest subset $S \subseteq V(G)$, s.t. $S$ touches all the edges of $G$.

3. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.
Greedy algorithms

GreedyVertexCover in action...
Greedy algorithms

Greedy Vertex Cover in action...
Greedy algorithms

GreedyVertexCover in action...
Greedy algorithms

GreedyVertexCover in action...
Greedy algorithms

GreedyVertexCover in action...
Greedy algorithms

GreedyVertexCover in action...
Greedy algorithms

GreedyVertexCover in action...
Greedy algorithms

GreedyVertexCover in action...
Greedy algorithms

GreedyVertexCover in action...
Greedy algorithms

Greedy\textit{VertexCover} in action...
Greedy algorithms

*GreedyVertexCover* in action...

**Observation 19.1.**

*GreedyVertexCover* returns 4 vertices, but opt is 3 vertices.
Back to **GreedyVertexCover**

1. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.

2. Returns 4, but opt is 3!

3. Can **not** be better than a $4/3$-approximation algorithm.

4. Actually it is much worse!
Back to **GreedyVertexCover**

1. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.
2. Returns 4, but opt is 3!
3. Can **not** be better than a $4/3$-approximation algorithm.
4. Actually it is much worse!
Back to **GreedyVertexCover**

1. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.
2. Returns 4, but opt is 3!
3. Can **not** be better than a $4/3$-approximation algorithm.
4. Actually it is much worse!
1. **GreedyVertexCover**: pick vertex with highest degree, remove, repeat.

2. Returns 4, but opt is 3!

3. Can **not** be better than a $4/3$-approximation algorithm.

4. Actually it is much worse!
Theorem 19.2.

There is a graph over \( n \) vertices, such that the smallest Vertex Cover has \( k \) vertices, but the greedy algorithm outputs a vertex cover of size \( \Theta(k \log n) \) approximation.

Proof: Outside the scope of this class...
...left as a hard exercise to the interested reader.

Vertex Cover is NP-Hard: Believe it requires exponential time to solve exactly.
Greedy Vertex Cover

**Theorem 19.2.**

There is a graph over $n$ vertices, such that the smallest Vertex Cover has $k$ vertices, but the greedy algorithm outputs a vertex cover of size $\Theta(k \log n)$ approximation.

Proof: Outside the scope of this class...

...left as a hard exercise to the interested reader.

---

**Vertex Cover** is **NP-Hard**: Believe it requires exponential time to solve exactly.
THE END

... (for now)
19.2
Greedy Algorithms: Tools and Techniques
What is a Greedy Algorithm?

No real consensus on a universal definition.

Greedy algorithms:

1. make decision incrementally in small steps without backtracking
2. decision at each step is based on improving local or current state in a myopic fashion without paying attention to the global situation
3. decisions often based on some fixed and simple priority rules
What is a Greedy Algorithm?

No real consensus on a universal definition.

Greedy algorithms:
1. make decision incrementally in small steps without backtracking
2. decision at each step is based on improving local or current state in a myopic fashion without paying attention to the global situation
3. decisions often based on some fixed and simple priority rules
What is a Greedy Algorithm?

No real consensus on a universal definition.

Greedy algorithms:

1. make decision incrementally in small steps **without** backtracking
2. decision at each step is based on improving **local** or **current** state in a myopic fashion without paying attention to the **global** situation
3. decisions often based on some fixed and simple **priority** rules
Pros and Cons of Greedy Algorithms

Pros:
1. Usually (too) easy to design greedy algorithms
2. Easy to implement and often run fast since they are simple
3. Several important cases where they are effective/optimal
4. Lead to a first-cut heuristic when problem not well understood

Cons:
- Very often greedy algorithms don’t work. Easy to lull oneself into believing they work
- Many greedy algorithms possible for a problem and no structured way to find effective ones

CS 374: Every greedy algorithm needs a proof of correctness
Pros and Cons of Greedy Algorithms

Pros:
1. Usually (too) easy to design greedy algorithms
2. Easy to implement and often run fast since they are simple
3. Several important cases where they are effective/optimal
4. Lead to a first-cut heuristic when problem not well understood

Cons:
1. **Very often** greedy algorithms don’t work. Easy to lull oneself into believing they work
2. Many greedy algorithms possible for a problem and no structured way to find effective ones

CS 374: Every greedy algorithm needs a proof of correctness
Pros and Cons of Greedy Algorithms

Pros:
1. Usually (too) easy to design greedy algorithms
2. Easy to implement and often run fast since they are simple
3. Several important cases where they are effective/optimal
4. Lead to a first-cut heuristic when problem not well understood

Cons:
1. Very often greedy algorithms don’t work. Easy to lull oneself into believing they work
2. Many greedy algorithms possible for a problem and no structured way to find effective ones

CS 374: Every greedy algorithm needs a proof of correctness
Greedy Algorithm Types

Crude classification:

1. **Non-adaptive**: fix some ordering of decisions a priori and stick with the order
2. **Adaptive**: make decisions adaptively but greedily/locally at each step

Plan:

1. See several examples
2. Pick up some proof techniques
Greedy Algorithm Types

Crude classification:

1. **Non-adaptive**: fix some ordering of decisions a priori and stick with the order
2. **Adaptive**: make decisions adaptively but greedily/locally at each step

Plan:

1. See several examples
2. Pick up some proof techniques
THE END

... (for now)
19.3 Scheduling Jobs to Minimize Average Waiting Time
The Problem

- \( n \) jobs \( J_1, J_2, \ldots, J_n \).
- Each \( J_i \) has non-negative processing time \( p_i \).
- One server/machine/person available to process jobs.
- Schedule/order jobs to min. total or average waiting time.
- Waiting time of \( J_i \) in schedule \( \sigma \): sum of processing times of all jobs scheduled before \( J_i \).

<table>
<thead>
<tr>
<th></th>
<th>( J_1 )</th>
<th>( J_2 )</th>
<th>( J_3 )</th>
<th>( J_4 )</th>
<th>( J_5 )</th>
<th>( J_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Example: schedule is \( J_1, J_2, J_3, J_4, J_5, J_6 \). Total waiting time is

\[
0 + 3 + (3 + 4) + (3 + 4 + 1) + (3 + 4 + 1 + 8) + \ldots =
\]

Optimal schedule: Shortest Job First. \( J_3, J_5, J_1, J_2, J_6, J_4 \).
The Problem

- $n$ jobs $J_1, J_2, \ldots, J_n$.
- Each $J_i$ has non-negative processing time $p_i$.
- One server/machine/person available to process jobs.
- Schedule/order jobs to min. total or average waiting time
- Waiting time of $J_i$ in schedule $\sigma$: sum of processing times of all jobs scheduled before $J_i$

<table>
<thead>
<tr>
<th>time</th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
<th>$J_5$</th>
<th>$J_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Example: schedule is $J_1, J_2, J_3, J_4, J_5, J_6$. Total waiting time is

$$0 + 3 + (3 + 4) + (3 + 4 + 1) + (3 + 4 + 1 + 8) + \ldots =$$

Optimal schedule: Shortest Job First. $J_3, J_5, J_1, J_2, J_6, J_4$. 
The Problem

- $n$ jobs $J_1, J_2, \ldots, J_n$.
- Each $J_i$ has non-negative processing time $p_i$.
- One server/machine/person available to process jobs.
- Schedule/order jobs to min. total or average waiting time.
- Waiting time of $J_i$ in schedule $\sigma$: sum of processing times of all jobs scheduled before $J_i$.

<table>
<thead>
<tr>
<th>time</th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
<th>$J_5$</th>
<th>$J_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Example: schedule is $J_1, J_2, J_3, J_4, J_5, J_6$. Total waiting time is

$$0 + 3 + (3 + 4) + (3 + 4 + 1) + (3 + 4 + 1 + 8) + \ldots =$$

Optimal schedule: Shortest Job First. $J_3, J_5, J_1, J_2, J_6, J_4$. 
The Problem

- \( n \) jobs \( J_1, J_2, \ldots, J_n \).
- Each \( J_i \) has non-negative processing time \( p_i \).
- One server/machine/person available to process jobs.
- Schedule/order jobs to min. total or average waiting time.
- Waiting time of \( J_i \) in schedule \( \sigma \): sum of processing times of all jobs scheduled before \( J_i \).

<table>
<thead>
<tr>
<th>time</th>
<th>( J_1 )</th>
<th>( J_2 )</th>
<th>( J_3 )</th>
<th>( J_4 )</th>
<th>( J_5 )</th>
<th>( J_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Example: schedule is \( J_1, J_2, J_3, J_4, J_5, J_6 \). Total waiting time is

\[
0 + 3 + (3 + 4) + (3 + 4 + 1) + (3 + 4 + 1 + 8) + \ldots =
\]

Optimal schedule: Shortest Job First. \( J_3, J_5, J_1, J_2, J_6, J_4 \).
Theorem 19.1.

Shortest Job First gives an optimum schedule for the problem of minimizing total waiting time.

Proof strategy: exchange argument

Assume without loss of generality that job sorted in increasing order of processing time and hence $p_1 \leq p_2 \leq \ldots \leq p_n$ and SJF order is $J_1, J_2, \ldots, J_n$. 
Theorem 19.1.

Shortest Job First gives an optimum schedule for the problem of minimizing total waiting time.

Proof strategy: exchange argument

Assume without loss of generality that job sorted in increasing order of processing time and hence $p_1 \leq p_2 \leq \ldots \leq p_n$ and SJF order is $J_1, J_2, \ldots, J_n$. 
Theorem 19.1.

Shortest Job First gives an optimum schedule for the problem of minimizing total waiting time.

Proof strategy: exchange argument

Assume without loss of generality that job sorted in increasing order of processing time and hence $p_1 \leq p_2 \leq \ldots \leq p_n$ and SJF order is $J_1, J_2, \ldots, J_n$. 
Optimality of SJF: Proof by picture
Optimality of SJF: Proof by picture
Optimality of SJF: Proof by picture
Optimality of **SJF**: Proof by picture
Optimality of $SJF$: Proof by picture
Inversions

**Definition 19.2.**
A schedule $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ has an inversion if there are jobs $J_a$ and $J_b$ such that $S$ schedules $J_a$ before $J_b$, but $p_a > p_b$.

**Claim 19.3.**
If a schedule has an inversion then there is an inversion between two adjacent scheduled jobs.

Proof: exercise.
Inversions

**Definition 19.2.**
A schedule $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ has an inversion if there are jobs $J_a$ and $J_b$ such that $S$ schedules $J_a$ before $J_b$, but $p_a > p_b$.

**Claim 19.3.**
*If a schedule has an inversion then there is an inversion between two adjacent scheduled jobs.*

Proof: exercise.
Proof of optimality of SJF

SJF = Shortest Job First

Recall SJF order is $J_1, J_2, \ldots, J_n$.

- Let $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ be an optimum schedule with fewest inversions.
- If schedule has no inversions then it is identical to SJF schedule and we are done.
- Otherwise there is an $1 \leq \ell < n$ such that $i_\ell > i_{\ell+1}$ since schedule has inversion among two adjacent scheduled jobs.

Claim 19.4.

The schedule obtained from $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ by exchanging/swapping positions of jobs $J_{i_\ell}$ and $J_{i_{\ell+1}}$ is also optimal and has one fewer inversion.

Assuming claim we obtain a contradiction and hence optimum schedule with fewest inversions must be the SJF schedule.
Proof of optimality of SJF

SJF = Shortest Job First

Recall SJF order is $J_1, J_2, \ldots, J_n$.

- Let $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ be an optimum schedule with fewest inversions.
- If schedule has no inversions then it is identical to SJF schedule and we are done.
- Otherwise there is an $1 \leq \ell < n$ such that $i_{\ell} > i_{\ell + 1}$ since schedule has inversion among two adjacent scheduled jobs.

Claim 19.4.

The schedule obtained from $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ by exchanging/swapping positions of jobs $J_{i_{\ell}}$ and $J_{i_{\ell + 1}}$ is also optimal and has one fewer inversion.

Assuming claim we obtain a contradiction and hence optimum schedule with fewest inversions must be the SJF schedule.
Exercise: A Weighted Version

- \( n \) jobs \( J_1, J_2, \ldots, J_n \). \( J_i \) has non-negative processing time \( p_i \) and a non-negative weight \( w_i \).
- One server/machine/person available to process jobs.
- Schedule/order the jobs to minimize total or average waiting time.
- Waiting time of \( J_i \) in schedule \( \sigma \): sum of processing times of all jobs scheduled before \( J_i \).
- Goal: minimize total weighted waiting time.
- Formally, compute a permutation \( \pi \) that minimizes \( \sum_{i=1}^{n} \left( \sum_{j=1}^{i-1} p_{\pi(j)} \right) w_{\pi(i)} \).

<table>
<thead>
<tr>
<th></th>
<th>( J_1 )</th>
<th>( J_2 )</th>
<th>( J_3 )</th>
<th>( J_4 )</th>
<th>( J_5 )</th>
<th>( J_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>weight</td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>100</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
THE END

... 

(for now)
19.3.1

Exercise: Scheduling Jobs to Minimize Weighted Average Waiting Time
Exercise: A Weighted Version

- **n** jobs $J_1, J_2, \ldots, J_n$. $J_i$ has non-negative processing time $p_i$ and a non-negative weight $w_i$.
- One server/machine/person available to process jobs.
- Schedule/order the jobs to minimize total or average waiting time.
- Waiting time of $J_i$ in schedule $\sigma$: sum of processing times of all jobs scheduled before $J_i$.
- Goal: minimize total weighted waiting time.
- Formally, compute a permutation $\pi$ that minimizes $\sum_{i=1}^{n} \left( \sum_{j=1}^{i-1} p_{\pi(j)} \right) w_{\pi(i)}$.

<table>
<thead>
<tr>
<th></th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
<th>$J_5$</th>
<th>$J_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>time</strong></td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td><strong>weight</strong></td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>100</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Exercise: A Weighted Version

Consider two jobs $p_1, p_2$ of weight $w_1$ and $w_2$. We have two possibilities:

<table>
<thead>
<tr>
<th>Job 1 first</th>
<th>Job 2 first</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise: A Weighted Version

Consider two jobs $p_1, p_2$ of weight $w_1$ and $w_2$. We have two possibilities:

<table>
<thead>
<tr>
<th>Pricing</th>
<th>Job 1 first</th>
<th>Job 2 first</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \cdot w_1 + p_1 w_2$</td>
<td>$0w_2 + p_2 w_1$</td>
<td></td>
</tr>
</tbody>
</table>
Exercise: A Weighted Version

Consider two jobs \( p_1, p_2 \) of weight \( w_1 \) and \( w_2 \). We have two possibilities:

<table>
<thead>
<tr>
<th></th>
<th>Job 1 first</th>
<th>Job 2 first</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pricing</td>
<td>0 \cdot w_1 + p_1 w_2</td>
<td>0w_2 + p_2 w_1</td>
</tr>
<tr>
<td>Equivalent to</td>
<td>( p_1 w_2 )</td>
<td>( p_2 w_1 )</td>
</tr>
</tbody>
</table>
Consider two jobs $p_1, p_2$ of weight $w_1$ and $w_2$. We have two possibilities:

<table>
<thead>
<tr>
<th></th>
<th>Job 1 first</th>
<th>Job 2 first</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pricing</td>
<td>$0 \cdot w_1 + p_1 w_2$</td>
<td>$0 w_2 + p_2 w_1$</td>
</tr>
<tr>
<td>Equivalent to</td>
<td>$p_1 w_2$</td>
<td>$p_2 w_1$</td>
</tr>
</tbody>
</table>
Exercise: A Weighted Version

Consider two jobs $p_1, p_2$ of weight $w_1$ and $w_2$. We have two possibilities:

<table>
<thead>
<tr>
<th></th>
<th>Job 1 first</th>
<th>Job 2 first</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pricing</td>
<td>$0 \cdot w_1 + p_1 w_2$</td>
<td>$0w_2 + p_2 w_1$</td>
</tr>
<tr>
<td>Equivalent to</td>
<td>$p_1 w_2$</td>
<td>$p_2 w_1$</td>
</tr>
</tbody>
</table>

need to compare $p_1 w_2 \overset{?}{=} p_2 w_1$
Exercise: A Weighted Version

Consider two jobs \( p_1, p_2 \) of weight \( w_1 \) and \( w_2 \). We have two possibilities:

<table>
<thead>
<tr>
<th></th>
<th>Job 1 first</th>
<th>Job 2 first</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pricing</td>
<td>( 0 \cdot w_1 + p_1 w_2 )</td>
<td>( 0w_2 + p_2 w_1 )</td>
</tr>
<tr>
<td>Equivalent to</td>
<td>( p_1 w_2 )</td>
<td>( p_2 w_1 )</td>
</tr>
</tbody>
</table>

need to compare \( p_1 w_2 = p_2 w_1 \)
Exercise: A Weighted Version

Consider two jobs $p_1, p_2$ of weight $w_1$ and $w_2$. We have two possibilities:

<table>
<thead>
<tr>
<th></th>
<th>Job 1 first</th>
<th>Job 2 first</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pricing</td>
<td>$0 \cdot w_1 + p_1 w_2$</td>
<td>$0w_2 + p_2 w_1$</td>
</tr>
<tr>
<td>Equivalent to</td>
<td>$p_1 w_2$</td>
<td>$p_2 w_1$</td>
</tr>
</tbody>
</table>

need to compare $p_1 w_2 \overset{?}{=} p_2 w_1$

dividing by $p_1 p_2$...
Exercise: A Weighted Version

Consider two jobs $p_1, p_2$ of weight $w_1$ and $w_2$. We have two possibilities:

<table>
<thead>
<tr>
<th></th>
<th>Job 1 first</th>
<th>Job 2 first</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pricing</td>
<td>$0 \cdot w_1 + p_1 w_2$</td>
<td>$0 w_2 + p_2 w_1$</td>
</tr>
<tr>
<td>Equivalent to</td>
<td>$p_1 w_2$</td>
<td>$p_2 w_1$</td>
</tr>
</tbody>
</table>

need to compare $p_1 w_2 \ ? \ p_2 w_1$

dividing by $p_1 p_2$...

equivalent to comparing $w_2/p_2 \ ? \ w_1/p_1$

$\omega_i = w_i/p_i$: Price per processing unit in dollars
Exercise: A Weighted Version

Consider two jobs $p_1, p_2$ of weight $w_1$ and $w_2$. We have two possibilities:

<table>
<thead>
<tr>
<th></th>
<th>Job 1 first</th>
<th>Job 2 first</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pricing</td>
<td>$0 \cdot w_1 + p_1 w_2$</td>
<td>$0 w_2 + p_2 w_1$</td>
</tr>
<tr>
<td>Equivalent to</td>
<td>$p_1 w_2$</td>
<td>$p_2 w_1$</td>
</tr>
</tbody>
</table>

need to compare $p_1 w_2 \ ? \ p_2 w_1$

dividing by $p_1 p_2$...

equivalent to comparing $w_2 / p_2 \ ? \ w_1 / p_1$

$\omega_i = w_i / p_i$: Price per processing unit in dollars
# Exercise: A Weighted Version

Consider two jobs $p_1$, $p_2$ of weight $w_1$ and $w_2$. We have two possibilities:

<table>
<thead>
<tr>
<th></th>
<th>Job 1 first</th>
<th>Job 2 first</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pricing</td>
<td>$0 \cdot w_1 + p_1 w_2$</td>
<td>$0w_2 + p_2 w_1$</td>
</tr>
<tr>
<td>Equivalent to</td>
<td>$p_1 w_2$</td>
<td>$p_2 w_1$</td>
</tr>
</tbody>
</table>

need to compare $p_1 w_2 \overset{?}{=} p_2 w_1$
dividing by $p_1 p_2$...
equivalent to comparing $w_2 / p_2 \overset{?}{=} w_1 / p_1$
$\omega_i = w_i / p_i$: Price per processing unit in dollars
Sort jobs in decreasing value of $\omega_i$. Schedule jobs by this value.
Exercise: A Weighted Version

Consider two jobs $p_1, p_2$ of weight $w_1$ and $w_2$. We have two possibilities:

<table>
<thead>
<tr>
<th></th>
<th>Job 1 first</th>
<th>Job 2 first</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pricing</td>
<td>$0 \cdot w_1 + p_1 w_2$</td>
<td>$0 w_2 + p_2 w_1$</td>
</tr>
<tr>
<td>Equivalent to</td>
<td>$p_1 w_2$</td>
<td>$p_2 w_1$</td>
</tr>
</tbody>
</table>

need to compare $p_1 w_2 \overset{?}{=} p_2 w_1$

dividing by $p_1 p_2$...

equivalent to comparing $w_2 / p_2 \overset{?}{=} w_1 / p_1$

$\omega_i = w_i / p_i$: Price per processing unit in dollars

Sort jobs in decreasing value of $\omega_i$. Schedule jobs by this value.

**Correctness proof:** Same as the unweighted case – if there is an inversion, then by the argument above, flip these jobs, and get a better schedule.
THE END

... (for now)
19.4
Scheduling to Minimize Lateness
Scheduling to Minimize Lateness

1. Given jobs \( J_1, J_2, \ldots, J_n \) with deadlines and processing times to be scheduled on a single resource.

2. If a job \( i \) starts at time \( s_i \) then it will finish at time \( f_i = s_i + t_i \), where \( t_i \) is its processing time. \( d_i \): deadline.

3. The lateness of a job is \( \ell_i = \max(0, f_i - d_i) \).

4. Schedule all jobs such that \( L = \max \ell_i \) is minimized.

<table>
<thead>
<tr>
<th></th>
<th>( J_1 )</th>
<th>( J_2 )</th>
<th>( J_3 )</th>
<th>( J_4 )</th>
<th>( J_5 )</th>
<th>( J_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_i )</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>( d_i )</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

\( \ell_1 = 2 \) \quad \ell_5 = 0 \quad \ell_4 = 6 \)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Scheduling to Minimize Lateness

1. Given jobs $J_1, J_2, \ldots, J_n$ with deadlines and processing times to be scheduled on a single resource.

2. If a job $i$ starts at time $s_i$ then it will finish at time $f_i = s_i + t_i$, where $t_i$ is its processing time. $d_i$: deadline.

3. The lateness of a job is $\ell_i = \max(0, f_i - d_i)$.

4. Schedule all jobs such that $L = \max \ell_i$ is minimized.

<table>
<thead>
<tr>
<th></th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
<th>$J_5$</th>
<th>$J_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_i$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$d_i$</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

$\ell_1 = 2$, $\ell_5 = 0$, $\ell_4 = 6$
Greedy Template

Initially $R$ is the set of all requests

$\textit{curr\_time} = 0$

$\textit{max\_lateness} = 0$

while $R$ is not empty do

choose $i \in R$

$\textit{curr\_time} = \textit{curr\_time} + t_i$

if ($\textit{curr\_time} > d_i$) then

$\textit{max\_lateness} = \max(\textit{curr\_time} - d_i, \textit{max\_lateness})$

return $\textit{max\_lateness}$

Main task: Decide the order in which to process jobs in $R$
Greedy Template

<table>
<thead>
<tr>
<th>Initially $R$ is the set of all requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{curr_time} = 0$</td>
</tr>
<tr>
<td>$\text{max_lateness} = 0$</td>
</tr>
<tr>
<td><strong>while</strong> $R$ is not empty <strong>do</strong></td>
</tr>
<tr>
<td>choose $i \in R$</td>
</tr>
<tr>
<td>$\text{curr_time} = \text{curr_time} + t_i$</td>
</tr>
<tr>
<td><strong>if</strong> ($\text{curr_time} &gt; d_i$) <strong>then</strong></td>
</tr>
<tr>
<td>$\text{max_lateness} = \max(\text{curr_time} - d_i, \text{max_lateness})$</td>
</tr>
</tbody>
</table>

| return $\text{max\_lateness}$ |

**Main task:** Decide the order in which to process jobs in $R$
Three Algorithms

1. Shortest job first — sort according to $t_i$.
2. Shortest slack first — sort according to $d_i - t_i$.
3. EDF = Earliest deadline first — sort according to $d_i$.

Counter examples for first two: exercise
Three Algorithms

1. Shortest job first — sort according to $t_i$.
2. Shortest slack first — sort according to $d_i - t_i$.
3. EDF = Earliest deadline first — sort according to $d_i$.

Counter examples for first two: exercise
Earliest Deadline First

**Theorem 19.1.**

*Greedy with EDF rule minimizes maximum lateness.*

Proof via an exchange argument.

Idle time: time during which machine is not working.

**Lemma 19.2.**

*If there is a feasible schedule then there is one with no idle time before all jobs are finished.*
Earliest Deadline First

**Theorem 19.1.**

*Greedy with EDF rule minimizes maximum lateness.*

Proof via an exchange argument.

Idle time: time during which machine is not working.

**Lemma 19.2.**

*If there is a feasible schedule then there is one with no idle time before all jobs are finished.*
Earliest Deadline First

**Theorem 19.1.**

*Greedy with EDF rule minimizes maximum lateness.*

Proof via an exchange argument.

Idle time: time during which machine is not working.

**Lemma 19.2.**

*If there is a feasible schedule then there is one with no idle time before all jobs are finished.*
Earliest Deadline First

**Theorem 19.1.**

*Greedy with EDF rule minimizes maximum lateness.*

Proof via an exchange argument.

*Idle time: time during which machine is not working.*

**Lemma 19.2.**

*If there is a feasible schedule then there is one with no idle time before all jobs are finished.*
Inversions

EDF = Earliest Deadline First

Assume jobs are sorted such that $d_1 \leq d_2 \leq \ldots \leq d_n$. Hence EDF schedules them in this order.

**Definition 19.3.**
A schedule $S$ is said to have an inversion if there are jobs $i$ and $j$ such that $S$ schedules $i$ before $j$, but $d_i > d_j$.

**Claim 19.4.**

If a schedule $S$ has an inversion then there is an inversion between two adjacent scheduled jobs.

Proof: exercise.
**Inversions**

**EDF** = Earliest Deadline First

Assume jobs are sorted such that $d_1 \leq d_2 \leq \ldots \leq d_n$. Hence **EDF** schedules them in this order.

**Definition 19.3.**

A schedule $S$ is said to have an inversion if there are jobs $i$ and $j$ such that $S$ schedules $i$ before $j$, but $d_i > d_j$.

**Claim 19.4.**

*If a schedule $S$ has an inversion then there is an inversion between two adjacent scheduled jobs.*

Proof: exercise.
Proof sketch of Optimality of EDF

- Let $S$ be an optimum schedule with smallest number of inversions.
- If $S$ has no inversions then this is same as **EDF** and we are done.
- Else $S$ has two adjacent jobs $i$ and $j$ with $d_i > d_j$.
- Swap positions of $i$ and $j$ to obtain a new schedule $S'$

**Claim 19.5.**

*Maximum lateness of $S'$ is no more than that of $S$. And $S'$ has strictly fewer inversions than $S$.***
THE END
...
(for now)
19.5

Maximum Weight Subset of Elements: Cardinality and Beyond
Picking k elements to maximize total weight

1. Given \( n \) items each with non-negative weights/profits and integer \( 1 \leq k \leq n \).
2. Goal: pick \( k \) elements to maximize total weight of items picked.

<table>
<thead>
<tr>
<th>weight</th>
<th>( e_1 )</th>
<th>( e_2 )</th>
<th>( e_3 )</th>
<th>( e_4 )</th>
<th>( e_5 )</th>
<th>( e_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

\( k = 2: \)

\( k = 3: \)

\( k = 4: \)
Greedy Template

\[ N \text{ is the set of all elements } \ X \leftarrow \emptyset \]

(* \( X \) will store all the elements that will be picked *)

\[ \text{while } |X| < k \text{ and } N \text{ is not empty do} \]

choose \( e_j \in N \) of maximum weight

add \( e_j \) to \( X \)

remove \( e_j \) from \( N \)

\[ \text{return the set } \ X \]

**Remark:** One can rephrase algorithm simply as sorting elements in decreasing weight order and picking the top \( k \) elements but the above template generalizes to other settings a bit more easily.

**Theorem 19.1.**

Greedy is optimal for picking \( k \) elements of maximum weight.
Greedy Template

\[ N \text{ is the set of all elements } X \leftarrow \emptyset \]
(* \( X \) will store all the elements that will be picked *)

\begin{algorithm}
\textbf{while} \( |X| < k \) and \( N \) is not empty \textbf{do}
\hspace{1em} choose \( e_j \in N \) of maximum weight
\hspace{1em} add \( e_j \) to \( X \)
\hspace{1em} remove \( e_j \) from \( N \)
\textbf{return} the set \( X \)
\end{algorithm}

**Remark:** One can rephrase algorithm simply as sorting elements in decreasing weight order and picking the top \( k \) elements but the above template generalizes to other settings a bit more easily.

**Theorem 19.1.**

Greedy is optimal for picking \( k \) elements of maximum weight.
A more interesting problem

1. Given $n$ items $N = \{e_1, e_2, \ldots, e_n\}$. Each item $e_i$ has a non-negative weight $w_i$.

2. Items partitioned into $h$ sets $N_1, N_2, \ldots, N_h$. Think of each item having one of $h$ colors.

3. Given integers $k_1, k_2, \ldots, k_h$ and another integer $k$

4. Goal: pick $k$ elements such that no more than $k_i$ from $N_i$ to maximize total weight of items picked.

<table>
<thead>
<tr>
<th></th>
<th>$e_1$</th>
<th>$e_2$</th>
<th>$e_3$</th>
<th>$e_4$</th>
<th>$e_5$</th>
<th>$e_6$</th>
<th>$e_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>weight</strong></td>
<td>9</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

$N_1 = \{e_1, e_2, e_3\}$, $N_2 = \{e_4, e_5\}$, $N_3 = \{e_6, e_7\}$
$k = 4$, $k_1 = 2$, $k_2 = 1$, $k_3 = 2$
Theorem 19.2.

Greedy is optimal for the problem on previous slide.

Proof: exercise after class.

Special case of general phenomenon of Greedy working for maximum weight independent set in a matroid. Beyond scope of course.
Greedy Template

\[ N \text{ is the set of all elements } X \leftarrow \emptyset \]
\((* X \text{ will store all the elements that will be picked } *)\)
\[ \text{while } N \text{ is not empty do} \]
\[ N' = \{ e_i \in N \mid X \cup \{ e_i \} \text{ is feasible} \} \]
\[ \text{if } N' = \emptyset \text{ then break} \]
\[ \text{choose } e_j \in N' \text{ of maximum weight} \]
\[ \text{add } e_j \text{ to } X \]
\[ \text{remove } e_j \text{ from } N \]
\[ \text{return the set } X \]

**Theorem 19.2.**

*Greedy is optimal for the problem on previous slide.*

Proof: exercise after class.

Special case of general phenomenon of Greedy working for maximum weight independent set in a matroid. Beyond scope of course.
THE END

... (for now)
19.6
Interval Scheduling
19.6.1
Problem statement, and a few greedy algorithms that do not work
Interval Scheduling

Problem 19.1 (Interval Scheduling).

Input: A set of jobs with start and finish times to be scheduled on a resource (example: classes and class rooms).

Goal: Schedule as many jobs as possible

Two jobs with overlapping intervals cannot both be scheduled!
Problem 19.1 (Interval Scheduling).

**Input:** A set of jobs with start and finish times to be scheduled on a resource (example: classes and class rooms).

**Goal:** Schedule as many jobs as possible

- Two jobs with overlapping intervals cannot both be scheduled!
Greedy Template

\( R \) is the set of all requests
\( X \leftarrow \emptyset \) (* \( X \) will store all the jobs that will be scheduled *)

while \( R \) is not empty do
  choose \( i \in R \)
  add \( i \) to \( X \)
  remove from \( R \) all requests that overlap with \( i \)
return the set \( X \)

Main task: Decide the order in which to process requests in \( R \)
Greedy Template

\begin{itemize}
\item $R$ is the set of all requests
\item $X \leftarrow \emptyset$ (* $X$ will store all the jobs that will be scheduled *)
\item \textbf{while} $R$ is not empty \textbf{do}
\begin{itemize}
\item choose $i \in R$
\item add $i$ to $X$
\item remove from $R$ all requests that overlap with $i$
\end{itemize}
\item \textbf{return} the set $X$
\end{itemize}

\textbf{Main task:} Decide the order in which to process requests in $R$
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.

Figure: Counter example for earliest start time
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.

Figure: Counter example for earliest start time
Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.

Figure: Counter example for earliest start time
Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.

___   ___   ___   ___  
_____________________
_____________________

Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.
Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.
Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.

---

---
Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.
Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.

Figure: Counter example for smallest processing time
Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.

Figure: Counter example for smallest processing time
Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.

Figure: Counter example for smallest processing time
Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.

_____ _____  _____  _____

_____  _____

____________________
Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.

________  ______  ______

____     ____

_________________________
Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.

[Red bar] _____  _____  _____  _____

______________________________
Fewest Conflicts

Process jobs in that have the fewest "conflicts" first.
Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.
Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.

[Diagram of job sequence]

**Figure**: Counter example for fewest conflicts
Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.

Figure: Counter example for fewest conflicts
Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.

Figure: Counter example for fewest conflicts
Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.

Figure: Counter example for fewest conflicts
THE END

... (for now)
19.6.2
Interval Scheduling: Earliest finish time
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.
Theorem 19.2.
The greedy algorithm that picks jobs in the order of their finishing times is optimal.
### Implementation and Running Time

Initially $R$ is the set of all requests

$X \leftarrow \emptyset$ (* $X$ stores the jobs that will be scheduled *)

while $R$ is not empty

choose $i \in R$ such that finishing time of $i$ is least

if $i$ does not overlap with requests in $X$

add $i$ to $X$

remove $i$ from $R$

return the set $X$

- Presort all requests based on finishing time. $O(n \log n)$ time
- Now choosing least finishing time is $O(1)$
- Keep track of the finishing time of the last request added to $A$. Then check if starting time of $i$ later than that
- Thus, checking non-overlapping is $O(1)$
- Total time $O(n \log n + n) = O(n \log n)$
Comments

1. Interesting Exercise: smallest interval first picks at least half the optimum number of intervals.

2. All requests need not be known at the beginning. Such **online** algorithms are a subject of research.
Weighted Interval Scheduling

Suppose we are given \( n \) jobs. Each job \( i \) has a start time \( s_i \), a finish time \( f_i \), and a weight \( w_i \). We would like to find a set \( S \) of compatible jobs whose total weight is maximized. Which of the following greedy algorithms finds the optimum schedule?

- Earliest start time first.
- Earliest finish time first.
- Highest weight first.
- None of the above.
- IDK.

Weighted problem can be solved via dynamic programming. See notes.
Weighted Interval Scheduling

Suppose we are given \( n \) jobs. Each job \( i \) has a start time \( s_i \), a finish time \( f_i \), and a weight \( w_i \). We would like to find a set \( S \) of compatible jobs whose total weight is maximized. Which of the following greedy algorithms finds the optimum schedule?

- (A) Earliest start time first.
- (B) Earliest finish time first.
- (C) Highest weight first.
- (D) None of the above.
- (E) IDK.

Weighted problem can be solved via dynamic programming. See notes.
THE END

... (for now)
19.6.3
Proving optimality of earliest finish time
Earliest finish time: A quick recall
Earliest finish time: A quick recall
Earliest finish time: A quick recall
Earliest finish time: A quick recall

Time
Earliest finish time: A quick recall
Earliest finish time: A quick recall

Time
Earliest finish time: A quick recall

Time
Earliest finish time: A quick recall
Earliest finish time: A quick recall
Earliest finish time: A quick recall

Time
Earliest finish time: A quick recall
Earliest finish time: A quick recall
Earliest finish time: A quick recall
Earliest finish time: A quick recall

Time
Earliest finish time: A quick recall
Earliest finish time: A quick recall
Earliest finish time: A quick recall
Earliest finish time: A quick recall
Earliest finish time: A quick recall
Earliest finish time: A quick recall
Earliest finish time: A quick recall
Proving Optimality

1 Correctness: Clearly the algorithm returns a set of jobs that does not have any conflicts

2 For a set of requests \( R \), let \( O \) be an optimal set and let \( X \) be the set returned by the greedy algorithm. Then \( O = X \)? Not likely!

Instead we will show that \( |O| = |X| \)
Proving Optimality

1. **Correctness:** Clearly the algorithm returns a set of jobs that does not have any conflicts.

2. For a set of requests $R$, let $O$ be an optimal set and let $X$ be the set returned by the greedy algorithm. Then $O = X$? Not likely!

Instead we will show that $|O| = |X|$.
Proving Optimality

1. Correctness: Clearly the algorithm returns a set of jobs that does not have any conflicts.

2. For a set of requests $R$, let $O$ be an optimal set and let $X$ be the set returned by the greedy algorithm. Then $O = X$? Not likely!

Instead we will show that $|O| = |X|$
Proving Optimality

1. **Correctness:** Clearly the algorithm returns a set of jobs that does not have any conflicts.

2. For a set of requests $R$, let $O$ be an optimal set and let $X$ be the set returned by the greedy algorithm. Then $O = X$? Not likely!

Instead we will show that $|O| = |X|$
Proving Optimality

Correctness: Clearly the algorithm returns a set of jobs that does not have any conflicts.

For a set of requests $R$, let $O$ be an optimal set and let $X$ be the set returned by the greedy algorithm. Then $O = X$? Not likely!

Instead we will show that $|O| = |X|$.
Proving Optimality

1. **Correctness**: Clearly the algorithm returns a set of jobs that does not have any conflicts.

2. For a set of requests \( R \), let \( O \) be an optimal set and let \( X \) be the set returned by the greedy algorithm. Then \( O = X \)? Not likely!

Instead we will show that \( |O| = |X| \)
Claim 19.3.

\( i \) be first interval picked by Greedy into solution.  
\( O \): Optimal solution.  
If \( i \not\in O \), there is exactly one interval \( j_1 \in O \) that conflicts with \( i \).

Proof.

1. No \( j \in O \) conflicts \( i \) \( \implies \) \( O \) is not opt!
2. Suppose \( j_1, j_2 \in O \) such that \( j_1 \neq j_2 \) and both \( j_1 \) and \( j_2 \) conflict with \( i \).
3. Since \( i \) has earliest finish time, \( j_1 \) and \( i \) overlap at \( f(i) \).
4. For same reason \( j_2 \) also overlaps with \( i \) at \( f(i) \).
5. Implies that \( j_1, j_2 \) overlap at \( f(i) \) but intervals in \( O \) cannot overlap.
Proof of Optimality: Key Lemma

**Lemma 19.4.**

$i_1$ be first interval picked by Greedy. There exists an optimum solution that contains $i_1$.

**Proof.**

Let $O$ be an arbitrary optimum solution. If $i_1 \in O$ we are done.

By Claim 19.3 ...

1. Exists exactly one $j_1 \in O$ conflicting with $i_1$.
2. Form a new set $O'$ by removing $j_1$ from $O$ and adding $i_1$, that is $O' = (O - \{j_1\}) \cup \{i_1\}$.
3. From claim, $O'$ is a feasible solution (no conflicts).
4. Since $|O'| = |O|$, $O'$ is also an optimum solution and it contains $i_1$. 


Lemma 19.4.

\( i_1 \) be first interval picked by Greedy. There exists an optimum solution that contains \( i_1 \).

Proof.

Let \( O \) be an arbitrary optimum solution. If \( i_1 \in O \) we are done.

By Claim 19.3...

1. Exists exactly one \( j_1 \in O \) conflicting with \( i_1 \).
2. Form a new set \( O' \) by removing \( j_1 \) from \( O \) and adding \( i_1 \), that is \( O' = (O - \{j_1\}) \cup \{i_1\} \).
3. From claim, \( O' \) is a feasible solution (no conflicts).
4. Since \( |O'| = |O| \), \( O' \) is also an optimum solution and it contains \( i_1 \).
Lemma 19.4.

\( i_1 \) be first interval picked by Greedy. There exists an optimum solution that contains \( i_1 \).

Proof.

Let \( O \) be an arbitrary optimum solution. If \( i_1 \in O \) we are done. By Claim 19.3 ...

1. Exists exactly one \( j_1 \in O \) conflicting with \( i_1 \).
2. Form a new set \( O' \) by removing \( j_1 \) from \( O \) and adding \( i_1 \), that is \( O' = (O - \{j_1\}) \cup \{i_1\} \).
3. From claim, \( O' \) is a feasible solution (no conflicts).
4. Since \( |O'| = |O| \), \( O' \) is also an optimum solution and it contains \( i_1 \).
Proof of Optimality of Earliest Finish Time First

Proof by Induction on number of intervals.

**Base Case:** $n = 1$. Trivial since Greedy picks one interval.

**Induction Step:** Assume theorem holds for $i < n$.

Let $K$ be an input (i.e., instance) with $n$ intervals.

$i_1 \leftarrow$ First interval picked by greedy algorithm.

$K' \leftarrow$ The result of removing $i_1$ and all conflicting intervals from $K$.

$|K'| = |K| - 1$.

$G(K), G(K')$: Solution produced by Greedy on $K$ and $K'$, respectively.

**Lemma 19.4** $\implies$ optimum solution $O$ to $K$ with $i_1 \in O$.

Let $O' = O - \{i_1\}$. $O'$ is a solution to $K'$.

\[
|G(K)| = 1 + |G(K')| \quad \text{from Greedy description}
\geq 1 + |O'| \quad \text{By induction, $G(I')$ is optimum for $I'$}
= |O|
\]
Proof of Optimality of Earliest Finish Time First

Proof by Induction on number of intervals.

**Base Case:** \( n = 1 \). Trivial since Greedy picks one interval.

**Induction Step:** Assume theorem holds for \( i < n \).

Let \( K \) be an input (i.e., instance) with \( n \) intervals

\( i_1 \leftarrow \) First interval picked by greedy algorithm.

\( K' \leftarrow \) The result of removing \( i_1 \) and all conflicting intervals from \( K \).

\[ |K'| = |K| - 1. \]

\( G(K), G(K') \): Solution produced by Greedy on \( K \) and \( K' \), respectively.

**Lemma 19.4** \( \Rightarrow \) optimum solution \( O \) to \( K \) with \( i_1 \in O \).

Let \( O' = O - \{i_1\} \). \( O' \) is a solution to \( K' \).

\[
|G(K)| = 1 + |G(K')| \\
\geq 1 + |O'| \\
= |O|
\]

from Greedy description

By induction, \( G(I') \) is optimum for \( I' \)
Proof of Optimality of Earliest Finish Time First

Proof by Induction on number of intervals.

**Base Case:** \( n = 1 \). Trivial since Greedy picks one interval.

**Induction Step:** Assume theorem holds for \( i < n \).

Let \( K \) be an input (i.e., instance) with \( n \) intervals

\( i_1 \leftarrow \) First interval picked by greedy algorithm.

\( K' \leftarrow \) The result of removing \( i_1 \) and all conflicting intervals from \( K \).

\(|K'| = |K| - 1.\)

\( G(K), G(K') \): Solution produced by Greedy on \( K \) and \( K' \), respectively.

**Lemma 19.4** \( \implies \) optimum solution \( O \) to \( K \) with \( i_1 \in O \).

Let \( O' = O - \{i_1\} \). \( O' \) is a solution to \( K' \).

\[ |G(K)| = 1 + |G(K')| \]

\[ \geq 1 + |O'| \]

\[ = |O| \]

from Greedy description

By induction, \( G(I') \) is optimum for \( I' \)
Proof of Optimality of Earliest Finish Time First

Proof by Induction on number of intervals.

**Base Case:** \( n = 1 \). Trivial since Greedy picks one interval.

**Induction Step:** Assume theorem holds for \( i < n \).

Let \( K \) be an input (i.e., instance) with \( n \) intervals

- \( i_1 \) \( \leftarrow \) First interval picked by greedy algorithm.
- \( K' \) \( \leftarrow \) The result of removing \( i_1 \) and all conflicting intervals from \( K \).

\[ |K'| = |K| - 1. \]

**Lemma 19.4** \( \implies \) optimum solution \( O \) to \( K \) with \( i_1 \in O \).

Let \( O' = O - \{i_1\} \). \( O' \) is a solution to \( K' \).

\[
|G(K)| = 1 + |G(K')| \\
\geq 1 + |O'| \\
= |O|
\]

from Greedy description

By induction, \( G(I') \) is optimum for \( I' \)
Proof of Optimality of Earliest Finish Time First

Proof by Induction on number of intervals.

**Base Case:** \( n = 1 \). Trivial since Greedy picks one interval.

**Induction Step:** Assume theorem holds for \( i < n \).

Let \( K \) be an input (i.e., instance) with \( n \) intervals

- \( i_1 \leftarrow \) First interval picked by greedy algorithm.
- \( K' \leftarrow \) The result of removing \( i_1 \) and all conflicting intervals from \( K \).

\[ |K'| = |K| - 1. \]

**Lemma 19.4** \( \implies \) optimum solution \( O \) to \( K \) with \( i_1 \in O \).

Let \( O' = O - \{i_1\} \). \( O' \) is a solution to \( K' \).

\[
|G(K)| = 1 + |G(K')| \\
\geq 1 + |O'| \\
= |O|
\]

from Greedy description

By induction, \( G(I') \) is optimum for \( I' \)
Proof of Optimality of Earliest Finish Time First

Proof by Induction on number of intervals.

**Base Case:** \( n = 1 \). Trivial since Greedy picks one interval.

**Induction Step:** Assume theorem holds for \( i < n \).

Let \( K \) be an input (i.e., instance) with \( n \) intervals

\( i_1 \leftarrow \) First interval picked by greedy algorithm.

\( K' \leftarrow \) The result of removing \( i_1 \) and all conflicting intervals from \( K \).

\(|K'| = |K| - 1.\)

\( G(K), G(K') \): Solution produced by Greedy on \( K \) and \( K' \), respectively.

**Lemma 19.4** \( \implies \) optimum solution \( O \) to \( K \) with \( i_1 \in O \).

Let \( O' = O - \{i_1\} \). \( O' \) is a solution to \( K' \).

\[
|G(K)| = 1 + |G(K')| \quad \text{from Greedy description}
\]
\[
\geq 1 + |O'| \quad \text{By induction, } G(I') \text{ is optimum for } I'
\]
\[
= |O|
\]
Proof of Optimality of Earliest Finish Time First

Proof by Induction on number of intervals.

**Base Case:** $n = 1$. Trivial since Greedy picks one interval.

**Induction Step:** Assume theorem holds for $i < n$.

Let $K$ be an input (i.e., instance) with $n$ intervals

$i_1 \leftarrow$ First interval picked by greedy algorithm.

$K' \leftarrow$ The result of removing $i_1$ and all conflicting intervals from $K$.

$|K'| = |K| - 1$.

$G(K), G(K')$: Solution produced by Greedy on $K$ and $K'$, respectively.

**Lemma 19.4** $\implies$ optimum solution $O$ to $K$ with $i_1 \in O$.

Let $O' = O - \{i_1\}$. $O'$ is a solution to $K'$.

\[
|G(K)| = 1 + |G(K')| \\
\geq 1 + |O'| \\
= |O|
\]

from Greedy description

By induction, $G(I')$ is optimum for $I'$)
Proof of Optimality of Earliest Finish Time First

Proof by Induction on number of intervals.

**Base Case:** \( n = 1 \). Trivial since Greedy picks one interval.

**Induction Step:** Assume theorem holds for \( i < n \).

Let \( K \) be an input (i.e., instance) with \( n \) intervals

\( i_1 \leftarrow \) First interval picked by greedy algorithm.

\( K' \leftarrow \) The result of removing \( i_1 \) and all conflicting intervals from \( K \).

\(|K'| = |K| - 1\).

\( G(K), G(K') \): Solution produced by Greedy on \( K \) and \( K' \), respectively.

**Lemma 19.4** \( \implies \) optimum solution \( O \) to \( K \) with \( i_1 \in O \).

Let \( O' = O - \{i_1\} \). \( O' \) is a solution to \( K' \).

\[
|G(K)| = 1 + |G(K')|
\]

from Greedy description

\[
\geq 1 + |O'|
\]

By induction, \( G(I') \) is optimum for \( I' \)

\[
= |O|
\]
Proof of Optimality of Earliest Finish Time First

Proof by Induction on number of intervals.

**Base Case:** \( n = 1 \). Trivial since Greedy picks one interval.

**Induction Step:** Assume theorem holds for \( i < n \).

Let \( K \) be an input (i.e., instance) with \( n \) intervals

\[ i_1 \leftarrow \text{First interval picked by greedy algorithm.} \]

\( K' \leftarrow \text{The result of removing} \ i_1 \ \text{and all conflicting intervals from} \ K \).

\[ |K'| = |K| - 1. \]

\( G(K), G(K') \): Solution produced by Greedy on \( K \) and \( K' \), respectively.

**Lemma 19.4** \( \implies \) optimum solution \( O \) to \( K \) with \( i_1 \in O \).

Let \( O' = O - \{i_1\} \). \( O' \) is a solution to \( K' \).

\[
|G(K)| = 1 + |G(K')| \quad \text{from Greedy description}
\]

\[
\geq 1 + |O'| \quad \text{By induction,} \ G(I') \text{ is optimum for} \ I'
\]

\[ = |O| \]
Proof of Optimality of Earliest Finish Time First

Proof by Induction on number of intervals.

**Base Case:** \( n = 1 \). Trivial since Greedy picks one interval.

**Induction Step:** Assume theorem holds for \( i < n \).

Let \( K \) be an input (i.e., instance) with \( n \) intervals.

\( i_1 \leftarrow \) First interval picked by greedy algorithm.

\( K' \leftarrow \) The result of removing \( i_1 \) and all conflicting intervals from \( K \).

\[ |K'| = |K| - 1. \]

\( G(K), G(K') \): Solution produced by Greedy on \( K \) and \( K' \), respectively.

**Lemma 19.4** \( \implies \) optimum solution \( O \) to \( K \) with \( i_1 \in O \).

Let \( O' = O - \{i_1\}. \) \( O' \) is a solution to \( K' \).

\[
\begin{align*}
|G(K)| &= 1 + |G(K')| \\
&\geq 1 + |O'| \\
&= |O|
\end{align*}
\]

from Greedy description

By induction, \( G(I') \) is optimum for \( I' \)
THE END

...  

(for now)
Greedy algorithms – an epilogue
Greedy proof techniques: Overview

1. **Greedy’s first step leads to an optimum solution.** Show that optimal solution can be modified to agree with greedy after first step. Then use induction. Example, Interval Scheduling.

2. **Greedy algorithm stays ahead.** Show that after each step the solution of the greedy algorithm is at least as good as the solution of any other algorithm. Example, Interval scheduling.

3. **Structural property of solution.** Observe some structural bound of every solution to the problem, and show that greedy algorithm achieves this bound. Example, Interval Partitioning (see Kleinberg-Tardos book).

4. **Exchange argument.** Gradually transform any optimal solution to the one produced by the greedy algorithm, without hurting its optimality. Example: Minimizing lateness, and Interval scheduling.
Takeaway Points

1. **Greedy algorithms** come naturally but often are incorrect. A proof of correctness is an absolute necessity.

2. **Exchange arguments** are often the key proof ingredient. Focus on why the first step of the algorithm is correct: need to show that there is an optimum/correct solution with the first step of the algorithm.

3. Thinking about correctness is also a good way to figure out which of the many greedy strategies is likely to work.