Recall—scheduling jobs to minimize max. lateness

Greedy Algorithm: order jobs by deadline,
so \( d_1 \leq d_2 \leq \ldots \leq d_n \)

we will show that the greedy alg. minimizes max. lateness

Advice about proofs:
Don't be general at first! Try special cases!

What is a good special case here?

\( n = 2 \quad d_1 < d_2 \)

the \( \exists \) wrong \( \exists \) solution, \( \Omega \):

\[
\begin{array}{cc}
2 & 1 \\
\hline
1 & 2
\end{array}
\]

the greedy solution, \( G \):

\[
\begin{array}{cc}
2 & 1 \\
\hline
1 & 2
\end{array}
\]

\( G \) has job 1 before job 2

\( G \) has job 2 before job 1

\( l_0(1) \) = lateness of job 1 in \( \Omega \)

etc., for \( l_0(2), \ l_G(1), \ l_G(2) \)

\( l_G \) = max. lateness of greedy schedule = \( \max \{l_G(1), \ l_G(2)\} \)

\( l_0 \) = max. lateness of other schedule

\( l_G(1) \leq l_0(1) \) because we moved \( 1 \) earlier

\( l_G(2) \leq l_0(1) \) because \( d_1 \leq d_2 \)

Therefore \( l_G \leq l_0(1) \leq l_0 \)
Can we generalize?
This idea allows us to swap a pair of consecutive jobs if their deadlines are out of order, making the solution better (or at least not worse).
Next: a proof that greedy gives optimal using "exchange proof".

**Theorem.** The greedy alg. gives an optimal solution, i.e., one that minimizes the maximum lateness.

**Proof.** An "exchange proof" that converts any solution to the greedy one without increasing max. lateness.
Let 1, ..., n be ordering of jobs by greedy alg., i.e.,
\( d_1 \leq d_2 \leq ... \leq d_n \).
Consider an optimal ordering of jobs. If it matches greedy, fine. Otherwise there must be two jobs that are consecutive in this ordering but in wrong order for greedy: \( i, j \) with \( d_j \leq d_i \).

**Claim:** Swapping \( i \) and \( j \) gives a new optimal ordering.
(Proof below)
Furthermore, the new optimal ordering has fewer inversions. So repeated swaps will eventually give us the greedy ordering, which must then be optimal.
Aside: recall that an inversion is a pair out of order. Doing a swap of two consecutive elements that are out of order decreases the # of inversions.

\[ \text{e.g.} \quad 2 \underline{5} \, 3 \, 1 \, 4 \quad \text{# inversions} \quad 5 \]

\[ 2 \, 3 \, 5 \, 1 \, 4 \quad 4 \]

\[ 2 \, 3 \, 1 \, 5 \, 4 \quad 3 \]

... eventually get sorted order

---

**Proof of claim**

Consider swapping jobs \( i \) and \( j \).

<table>
<thead>
<tr>
<th>( i )</th>
<th>( j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>old</td>
<td>new</td>
</tr>
</tbody>
</table>

\[ \ell_N(j) \leq \ell_O(j) \quad \text{because now we do } j \text{ first} \]

\[ \ell_N(i) \leq \ell_O(j) \quad \text{because } d_j \leq d_i \]

And all other jobs have some listeness.

Thus \( \ell_N \leq \ell_O \). But \( \ell_O \) was min. So \( \ell_N = \ell_O \).

So we can swap until we get the greedy soln, \( \ell \) unchanged.
Knapsack Problem

You're going on a 5 day camping trip to Algonquin Park. You want to pack your knapsack to maximize value and minimize weight.

Given n items, item i has weight \( w_i \) and value \( v_i \).

Weight limit of knapsack is \( W \). Put items in knapsack, sum of weights \( \leq W \), maximize sum of values.

[Notation: \( \text{find } S \subseteq \{1, \ldots, n\}, \sum v_i : i \in S \leq W \) and maximize \( \sum v_i : i \in S \) ]

Two versions of the problem:

- 0-1 knapsack. Items are indivisible (tent, flashlight)
- Fractional knapsack. Can use fractions of items (oatmeal, cheese)

We'll see a dynamic programming algorithm for 0-1 knapsack, but (in some sense) the alg. is not efficient and the problem is hard.

Today: a greedy algorithm for the fractional knapsack

Example:

<table>
<thead>
<tr>
<th></th>
<th>( v_i )</th>
<th>( w_i )</th>
<th>( v_i/w_i )</th>
<th>Note: it makes sense to order items by value per weight.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>3</td>
<td>2 1/3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

\( W = 6 \)
For the 0-1 case, greedy gives item 1, value 12 (nothing else fits) but taking items 2 and 3 gives value 13.

For fractional case, greedy takes item 1, leaving weight of 2 free, so take $2/3$ of item 2. Value: $12 + \frac{2}{3} \cdot 7$.

Greedy Algorithm

$x_i$ - weight of item $i$ that we take.

Free-W $\leftarrow$ W

for $i = 1 \ldots n$ (items ordered by $v_i/w_i$)

$\hat{x}_i \leftarrow \min \{x_i \leq w_i, \text{free-W} \}$

Free-W $\leftarrow$ Free-W $-$ $\hat{x}_i$

Note that the solution will look like:

<table>
<thead>
<tr>
<th>Item</th>
<th>1</th>
<th>2</th>
<th>$\ldots$</th>
<th>$j$</th>
<th>$j+1$</th>
<th>$\ldots$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$</td>
<td>$\overbrace{\tilde{x}_i \tilde{x}_2 \ldots \tilde{x}_j 0 \ldots 0}$</td>
<td>\underline{use none of items $j+1 \ldots n$}</td>
<td>use all of items $1 \ldots j-1$</td>
<td>\underline{use fraction of item $j$ $0 &lt; x_j &lt; w_j$}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Final weight: $\sum x_i = W$ (if $\sum w_i \geq W$)

Final value: $\sum (\frac{v_i}{w_i}) \hat{x}_i$

Running time $O(n \log n)$ to sort by $v_i/w_i$. 
Claim: The greedy alg. gives the opt. soln. to the fractional knapsack problem.

Proof: greedy solution $x_1, x_2, \ldots, x_{k-1}, x_k, \ldots, x_n$

opt. solution $y_1, y_2, \ldots, y_{k-1}, y_k, y_e, y_n$

Suppose $y$ is an opt. soln. that matches $x$ on max # indices.

If $x = y$ done. Let $k =$ first index where $x_k \neq y_k$.

Then $x_k > y_k$ since greedy maximizes $x_k$.

Since $\sum y = \sum x = W$, there is a later index $l > k$ with $y_l > x_l$.

Exchange weight $\Delta$ of item $l$ for equal weight of item $k$.

$y_k' = y_k + \Delta$

$y_l' = y_l - \Delta$

choose $\Delta$ so $x_k = y_k'$ or $x_l = y_l'$.

so $\Delta \leq \min \{ y_l - x_l, x_k - y_k \}$.

So $\Delta > 0$.

(amount we can move from $l$) - (amount we can add to $k$).

Change in value

$\Delta \left( \frac{v_k}{w_k} \right) - \Delta \left( \frac{v_l}{w_l} \right) = \Delta \left( \frac{v_k}{w_k} - \frac{v_l}{w_l} \right)$

This is non-neg. because $\frac{v_k}{w_k} \geq \frac{v_l}{w_l}$ (we sorted this way).

But $y$ was an opt. soln., so this can’t be better.

Therefore it’s a new opt. soln. that matches $x$ on one more index ($k$ or $l$). Contradiction to choice of $y$.

We will see more greedy algs. for graph problems.
Not covered in class
Some other greedy algorithms you have seen

- Huffman coding in CS 240
  - For more info, see http://jeffe.cs.illinois.edu/teaching/algorithms/book/ch4-greedy.pdf
- Caching algorithms
  LRU and LRU and FIFO are all greedy
  as is the optimal offline caching alg.