CS 341: ALGORITHMS

Trevor Brown

trevor.brown@uwaterloo.ca
THIS TIME

• “Slick” proof of greedy interval selection algorithm
• More greedy algorithms
  • Interval colouring
  • Rational knapsack
RECALL: INTERVAL SELECTION PROBLEM

- **Input:** a set $A = \{A_1, \ldots, A_n\}$ of time intervals
  - Each interval $A_i$ has a start time $s_i$ and a finish time $f_i$
- **Feasible solution:** a subset $B$ of $A$ containing pairwise disjoint intervals
- **Output:** a feasible solution of maximum size
  - i.e., one that maximizes $|B|

Where $s_i$ and $f_i$ are positive integers

Bad solution: only chose 3, but could choose 5!
A CORRECT GREEDY ALGORITHM

Sort the intervals in increasing order of **finishing times**. At any stage, choose the **earliest finishing** interval that is disjoint from all previously chosen intervals (i.e., the local evaluation criterion is $f_i$).
Induction is a standard way to prove correctness of greedy algorithms; however, sometimes shorter “slick” proofs are possible.

Let \( F = \{f_{i_1}, \ldots, f_{i_k}\} \) be the finishing times of the intervals in \( \mathcal{B} \).

No interval finishes strictly to the left

No interval starts strictly to the right

No interval is strictly between these points!

So, every interval chosen by optimal contains a point in \( F \)

No two intervals in \( O \) contain the same point in \( F \) (by disjointness)

So, there is an injective mapping from \( O \) to \( F \), so \( |O| \leq |F| \)
**PROBLEM: INTERVAL COLOURING**

**Instance:** A set $A = \{A_1, \ldots, A_n\}$ of intervals. For $1 \leq i \leq n$, $A_i = [s_i, f_i]$, where $s_i$ is the **start time** of interval $A_i$ and $f_i$ is the **finish time** of $A_i$.

**Feasible solution:** A $c$-colouring is a mapping $\text{col} : A \rightarrow \{1, \ldots, c\}$ that assigns each interval a **colour** such that two intervals receiving the same colour are always disjoint.

**Find:** A $c$-colouring of $A$ with the **minimum number of colours**.

---

**Example**

7 intervals, 7 colours. Feasible, but not optimal.
MORE EXAMPLES

Example

Not feasible!

Example

Same color, but disjoint. OK!

Example

Same color, but not disjoint... BAD

Example

7 intervals, 6 colours. Feasible, but not optimal

Example

7 intervals, 2 colours. Optimal
Greedy Strategies for Interval Colouring

As usual, we consider the intervals one at a time.

At a given point in time, suppose we have coloured the first $i < n$ intervals using $d$ colours.

We will colour the $(i + 1)$st interval with any permissible colour. If it cannot be coloured using any of the existing $d$ colours, then we introduce a new colour and $d$ is increased by 1.

Question: In what order should we consider the intervals?
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**

Consider intervals in the order they are given in the input: \(A_1\ldots A_{10}\)
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**Example:**

**Order Matters!**

---

| \(A_1\) | 1  |
| \(A_2\) |    |
| \(A_3\) |    |
| \(A_4\) |    |
| \(A_5\) |    |
| \(A_6\) |    |
| \(A_7\) |    |
| \(A_8\) |    |
| \(A_9\) |    |
| \(A_{10}\) |   |

**x-axis**

0 2 4 6 8 10 12 14 16 18 20
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**Example:**

**Order Matters!**
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**Example:**

**Order Matters!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**Example:**

ORDER MATTERS!
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

### Example: Order Matters!

<table>
<thead>
<tr>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
<th>(A_4)</th>
<th>(A_5)</th>
<th>(A_6)</th>
<th>(A_7)</th>
<th>(A_8)</th>
<th>(A_9)</th>
<th>(A_{10})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**x-axis**

0 2 4 6 8 10 12 14 16 18 20
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**Example:**

ORDER MATTERS!
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**Example:**

**Order matters!**

Used 4 colours

Can we do better?
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**Example:**

Order matters!

Pre-sort intervals by increasing start time!
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**
**ORDER MATTERS!**

Pre-sort intervals by increasing start time!
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**

![Graph showing intervals and colours]
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**Example:**

\[
\begin{array}{c|cccccc}
A_1 & 1 & & & & & \\
A_2 & 2 & & & & & \\
A_3 & 3 & & & & & \\
A_4 & & & & & & \\
A_5 & & & & & & \\
A_6 & & & & & & \\
A_7 & & & & & & \\
A_8 & & & & & & \\
A_9 & & & & & & \\
A_{10} & & & & & & \\
\end{array}
\]
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

<table>
<thead>
<tr>
<th>(A_i)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**Example:**

ORDER MATTERS!

Used 3 colours

Can we do better?
**Algorithm: GreedyIntervalColouring** $(A)$

1. sort the intervals so that $s_1 \leq \cdots \leq s_n$
2. $d \leftarrow 1$
3. $\text{finish}[1] \leftarrow f_1$
4. for $i \leftarrow 2$ to $n$
   - $\text{flag} \leftarrow \text{false}$
   - $c \leftarrow 1$
   - while $c \leq d$ and (not $\text{flag}$)
     - do
       - if $\text{finish}[c] \leq s_i$ then
         - $\text{flag} \leftarrow \text{true}$
         - $\text{colour}[i] \leftarrow c$
         - $\text{finish}[c] \leftarrow f_i$
       - else $c \leftarrow c + 1$
     - if not $\text{flag}$ then
       - $d \leftarrow d + 1$
       - $\text{colour}[i] \leftarrow d$
       - $\text{finish}[d] \leftarrow f_i$
   - return $(d, \text{colour})$

$d = \# \text{ of colours used so far}$

Interval 1 gets colour 1

For each interval $A_i$, search for an appropriate colour $c$

If $s_i \geq \text{finish}[c]$, then we can give $A_i$ colour $c$ without breaking feasibility

Consider interval $A_i = (s_i, f_i)$. If $s_i \geq \text{finish}[c]$, then we can give $A_i$ colour $c$ without breaking feasibility

If we didn’t reuse a colour, use a new colour

Check if we can use any colour $c$ in $1..d$
EXAMPLE: RUNNING GREEDY
<table>
<thead>
<tr>
<th>i=1</th>
<th>d=1</th>
<th>finish[1]=</th>
</tr>
</thead>
</table>

Code **before** the loop: just assign colour 1

**EXAMPLE:**
(RUNNING GREEDY)
While loop over $c$. Check if we can reuse a color in $1..d$

**EXAMPLE:**
**RUNNING GREEDY**

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$1$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_5$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_6$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_7$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_8$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_9$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_{10}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Is $\text{finish}[1] \leq s_2$?
- No. We cannot reuse colour 1.
- Cannot reuse any colour. Create a new one!

$x$-axis
i = 2

While loop over c.
Check if we can reuse a color in 1..d

Is $finish[1] \leq s_2$?
No. We cannot reuse colour 1.

d = 2

finish[1] =

Cannot reuse any colour. Create a new one!

finish[2] =
While loop over $c$. Check if we can reuse a color in $1..d$

**EXAMPLE: RUNNING GREEDY**

<table>
<thead>
<tr>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
<th>A_7</th>
<th>A_8</th>
<th>A_9</th>
<th>A_10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Is $\text{finish}[1] \leq s_3$?

- No. We cannot reuse colour 1.

Is $\text{finish}[2] \leq s_3$?

- No. We cannot reuse colour 2.

Cannot reuse any colour. Create new one.
EXAMPLE: RUNNING GREEDY

While loop over c. Check if we can reuse a color in 1..d

Is \( f_1 \leq s_3 \)?
No. We cannot reuse colour 1.

Is \( f_2 \leq s_3 \)?
No. We cannot reuse colour 2.

Cannot reuse any colour. Create new one.
EXAMPLE: RUNNING GREEDY

While loop over $c$. Check if we can reuse a color in $1..d$

Is $f_{\text{finish}[1]} \leq s_4$?

Yes. We can reuse colour 1.

While loop over $c$. Check if we can reuse a color in $1..d$

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
</tr>
<tr>
<td>$A_4$</td>
<td></td>
</tr>
<tr>
<td>$A_5$</td>
<td></td>
</tr>
<tr>
<td>$A_6$</td>
<td></td>
</tr>
<tr>
<td>$A_7$</td>
<td></td>
</tr>
<tr>
<td>$A_8$</td>
<td></td>
</tr>
<tr>
<td>$A_9$</td>
<td></td>
</tr>
<tr>
<td>$A_{10}$</td>
<td></td>
</tr>
</tbody>
</table>

i=4, d=3

Check if we can reuse a color in $1..d$

finish[1]=
finish[2]=
finish[3]=

x-axis
EXAMPLE: RUNNING GREEDY

While loop over \( c \). Check if we can reuse a color in 1..\( d \).

Is \( \text{finish}[1] \leq s_4 \)?

Yes. We can reuse colour 1.

\[
i = 4
\]

\[
d = 3
\]

\[
\text{finish}[1] =
\]

\[
\text{finish}[2] =
\]

\[
\text{finish}[3] =
\]
While loop over $c$. Check if we can reuse a color in $1..d$

**EXAMPLE:**
**RUNNING GREEDY**

Is $\text{finish}[1] \leq s_5$?
No. We **cannot** reuse colour 1.

Is $\text{finish}[2] \leq s_5$?
No. We **cannot** reuse colour 2.

Is $\text{finish}[3] \leq s_5$?
Yes. We **can** reuse colour 3.

<table>
<thead>
<tr>
<th>$A_i$</th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_2$</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_4$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_5$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_6$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_7$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_8$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_9$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_{10}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EXAMPLE: RUNNING GREEDY

While loop over c. Check if we can reuse a color in 1..d


Is \( f_1 \leq s_5 \)?
No. We cannot reuse colour 1.

Is \( f_2 \leq s_5 \)?
No. We cannot reuse colour 2.

Is \( f_3 \leq s_5 \)?
Yes. We can reuse colour 3.
EXAMPLE: RUNNING GREEDY

While loop over c. Check if we can reuse a color in 1..d

Is $finish[1] \leq s_6$?

No. We cannot reuse colour 1.

Is $finish[2] \leq s_6$?

Yes. We can reuse colour 2.

While loop over $c$. Check if we can reuse a color in 1..d

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
<th>$A_6$</th>
<th>$A_7$</th>
<th>$A_8$</th>
<th>$A_9$</th>
<th>$A_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Is $finish[1] \leq s_6$?

No. We cannot reuse colour 1.

Is $finish[2] \leq s_6$?

Yes. We can reuse colour 2.
While loop over $c$. Check if we can reuse a color in $1..d$

**EXAMPLE:**
**RUNNING GREEDY**

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$1$</th>
<th>Finish[1] =</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>$2$</td>
<td>Finish[2] =</td>
</tr>
<tr>
<td>$A_3$</td>
<td>$3$</td>
<td>Finish[3] =</td>
</tr>
</tbody>
</table>

Is $finish[1] \leq s_6$?
No. We **cannot** reuse colour 1.

Is $finish[2] \leq s_6$?
Yes. We **can** reuse colour 2.

And so on, and so forth…
Correctness of the Algorithm

The correctness of this greedy algorithm can be proven inductively as well as by a “slick” method—we give the “slick” proof:

Let $D$ denote the number of colours used by the algorithm.

Suppose $A_i = [s_i, f_i]$ is the first interval to receive the last colour, $D$.

For every colour $c < D$, there is an interval $A_c = [s_c, f_c]$ such that $s_c \leq s_i < f_c$ (i.e., $A_c$ overlaps $A_i$).
A₁ is the last interval that received colour 1 at the time A₃ received its colour.

s₁ ≤ s₃ because of the pre-sorting step.

First interval with colour D=3 is A₃.

Want to argue: for each colour c < D, there exists an interval Aᵞ with colour c that intersects A₃.

So f₁ is finish[1], and of course finish[1] > s₃ (or we would have reused colour 1).

Since s₁ ≤ s₃ ≤ f₁, A₁ must intersect A₃.

Same argument for A₂ intersecting A₃...

Intersection of three intervals implies three colours are required.
General Argument

We know \( \text{finish}[1] > s_D \), because if \( \text{finish}[1] \leq s_D \) then the greedy algorithm reuses colour 1!

Similarly, we know \( \text{finish}[2] > s_D \), \( \text{finish}[3] > s_D \), ...

And, we know these intervals all start at or before time \( s_D \)

So they must all intersect \( A_D \).

\[ \begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\text{colour} & 1 & 2 & 3 & 4 & 5 & \ldots & \text{colour } D-1 & \text{colour } D \\
\hline
\text{finish[1]} & & & \times & \times & \times & \ldots & &
\hline
\text{finish[2]} & & & & & & \ldots & &
\hline
\text{finish[3]} & & & & & & \ldots & &
\hline
\end{array} \]
Correctness of the Algorithm

The correctness of this greedy algorithm can be proven inductively as well as by a "slick" method—we give the "slick" proof:

Let $D$ denote the number of colours used by the algorithm.

Suppose $A_i = [s_i, f_i]$ is the first interval to receive the last colour, $D$.

For every colour $c < D$, there is an interval $A_c = [s_c, f_c]$ such that $s_c \leq s_i < f_c$ (i.e., $A_c$ overlaps $A_i$).

Therefore we have $D$ intervals, all of which contain the point $s_i$.

These $D$ intervals must all receive different colours, so there is no colouring with fewer than $D$ colours.
The complexity of the algorithm is $O(nD)$, where $D$ is the value of $d$ returned by the algorithm.

We don’t know the value of $D$ ahead of time; all we know is that $1 \leq D \leq n$.

If it turns out that $D \in \Omega(n)$, then the best we can say is that the complexity is $O(n^2)$.

What inefficiencies exist in this algorithm?

What data structure would allow a more efficient algorithm to be designed?

What would be the complexity of an algorithm making use of an appropriate data structure?
IMPROVING THIS ALGORITHM

• Current greedy algorithm:
  • For each interval \( A_i \), compare its start time \( s_i \) with the
    \textbf{finish}[c] times of \textbf{all} colours introduced so-far
  • Why? Looking for a \textbf{finish}[c] time that is earlier than \( s_i \)
  • We are doing \textbf{linear search}... Can we do better?
  • Use a priority queue to keep track of the \textbf{earliest \textbf{finish}[c]} at all times in the algorithm
    • Then we only need to look at \textbf{minimum element}
EXAMPLE: HEAP-BASED ALGORITHM

Min element: NULL

Heap
EXAMPLE: HEAP-BASED ALGORITHM

Min element: NULL

Heap

Iteration i=1  Check heap minimum  Empty, so a new colour is needed

A_1  A_2  A_3  A_4  A_5  A_6  A_7  A_8  A_9  A_{10}

x-axis
EXAMPLE: HEAP-BASED ALGORITHM

<table>
<thead>
<tr>
<th>Min element:</th>
<th>finish at time 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>finish at time 3</td>
</tr>
</tbody>
</table>

Iteration i=1
Check heap minimum
Empty, so a new colour is needed

<table>
<thead>
<tr>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
<th>A_7</th>
<th>A_8</th>
<th>A_9</th>
<th>A_{10}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x-axis

Finish at time 3
EXAMPLE: HEAP-BASED ALGORITHM

Min element: finish at time 3
Heap finish at time 3

Iteration i=2
Check heap minimum
Check if finish time 3 is before $s_2$
No. New colour!

<table>
<thead>
<tr>
<th></th>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
<th>A_7</th>
<th>A_8</th>
<th>A_9</th>
<th>A_10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x-axis

Check if finish time 3 is before $s_2$
EXAMPLE: HEAP-BASED ALGORITHM

Check heap minimum
Check if finish time 3 is before \( s_2 \)
No. New colour!

Iteration i=2

| A_1 | 1 |
| A_2 | 2 |
| A_3 |
| A_4 |
| A_5 |
| A_6 |
| A_7 |
| A_8 |
| A_9 |
| A_{10} |

Min element: finish at time 3
Heap: finish at time 3
finish at time 7

x-axis
### Example: Heap-Based Algorithm

<table>
<thead>
<tr>
<th>Iteration i=3</th>
<th>Check heap minimum</th>
<th>Check if finish time 3 is before $s_3$</th>
<th>No. New colour!</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A_1</strong></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A_2</strong></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A_3</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A_4</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A_5</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A_6</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A_7</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A_8</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A_9</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A_{10}</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Min element:** finish at time 3

**Heap:** finish at time 3

Finish at time 7
EXAMPLE: HEAP-BASED ALGORITHM

Min element: finish at time 3
Heap finish at time 3
finish at time 7 finish at time 5

Iteration i=3  Check heap minimum  Check if finish time 3 is before $s_3$  No. New colour!

<table>
<thead>
<tr>
<th>A_1</th>
<th>1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A_2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_{10}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x-axis

Finish at time 3

Check if finish time 3 is before $s_3$
EXAMPLE: HEAP-BASED ALGORITHM

Min element: finish at time 3

Heap
finish at time 3
finish at time 7
finish at time 5

Iteration i=4
Check heap minimum
Check if finish time 3 is before $s_4$

Yes. Reuse colour, deleteMin and insert new finish time into heap!
### Example: Heap-Based Algorithm

**Iteration i=4**: Check heap minimum. Check if finish time 3 is before $s_4$.

**Yes. Reuse colour, deleteMin and insert new finish time into heap!**

| A_1 | 1  |
| A_2 | 2  |
| A_3 | 3  |
| A_4 |     |
| A_5 |     |
| A_6 |     |
| A_7 |     |
| A_8 |     |
| A_9 |     |
| A_{10} |     |

**Min element:**
- Finish at time 5

**Heap**
- Finish at time 7
- Finish at time 5

**x-axis**
- 0 2 4 6 8 10 12 14 16 18 20
EXAMPLE: HEAP-BASED ALGORITHM

<table>
<thead>
<tr>
<th>A_1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_2</td>
<td>2</td>
</tr>
<tr>
<td>A_3</td>
<td>3</td>
</tr>
<tr>
<td>A_4</td>
<td></td>
</tr>
<tr>
<td>A_5</td>
<td></td>
</tr>
<tr>
<td>A_6</td>
<td></td>
</tr>
<tr>
<td>A_7</td>
<td></td>
</tr>
<tr>
<td>A_8</td>
<td></td>
</tr>
<tr>
<td>A_9</td>
<td></td>
</tr>
<tr>
<td>A_{10}</td>
<td></td>
</tr>
</tbody>
</table>

Iteration i=4
Check heap minimum
Check if finish time 3 is before s_4
Yes. Reuse colour, deleteMin and insert new finish time into heap!

Min element: finish at time 5
Heap: finish at time 9
finish at time 7
finish at time 5

x-axis
example: heap-based algorithm

iteration i=5
check heap minimum
check if finish time 5 is before s5
yes. reuse colour, deleteMin and insert new finish time into heap!

min element: finish at time 5

heap

| A1 | 1 |
| A2 | 2 |
| A3 | 3 |
| A4 |  |  |
| A5 |  |  |
| A6 |  |  |
| A7 |  |  |
| A8 |  |  |
| A9 |  |  |
| A10|  |  |

x-axis

0  2  4  6  8  10  12  14  16  18  20
**EXAMPLE: HEAP-BASED ALGORITHM**

Min element: **finish at time 7**

Heap: finish at **time 9**

**Iteration i=5**

Check heap minimum

Check if finish time 5 is before $s_5$

Yes. **Reuse colour**, **deleteMin** and insert new finish time into heap!
**EXAMPLE: HEAP-BASED ALGORITHM**

- **Iteration i=5**
- **Check heap minimum**
- **Check if finish time 5 is before s₅**

Yes. **Reuse colour, deleteMin** and insert new finish time into heap!

<table>
<thead>
<tr>
<th>A₁</th>
<th>A₂</th>
<th>A₃</th>
<th>A₄</th>
<th>A₅</th>
<th>A₆</th>
<th>A₇</th>
<th>A₈</th>
<th>A₉</th>
<th>A₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Min element:**
- Finish at time 7

**Heap:**
- Finish at time 9
- Finish at time 7
- Finish at time 13

**x-axis**
0 2 4 6 8 10 12 14 16 18 20
**EXAMPLE: HEAP-BASED ALGORITHM**

Iteration $i=6$

Check heap minimum

Check if finish time 5 is before $s_6$

Yes. **Reuse** colour, **deleteMin** and insert new finish time into heap!

Min element: finish at time 7

Heap finish at time 9

finish at time 7 finish at time 13

---

**Check heap minimum**

**Check if finish time 5 is before $s_6$**

Yes. **Reuse** colour, **deleteMin** and insert new finish time into heap!
EXAMPLE: HEAP-BASED ALGORITHM

Iteration i=6
Check heap minimum
Check if finish time 5 is before $s_6$

Yes. **Reuse** colour, **deleteMin** and insert new finish time into heap!

| A₁ | 1 |   |   |   |   |   |
| A₂ | 2 |   |   |   |   |   |
| A₃ | 3 |   |   |   |   |   |
| A₄ |   |   |   |   |   |   |
| A₅ |   |   |   |   |   |   |
| A₆ |   |   |   |   |   |   |
| A₇ |   |   |   |   |   |   |
| A₈ |   |   |   |   |   |   |
| A₉ |   |   |   |   |   |   |
| A₁₀|   |   |   |   |   |   |

Min element: finish at time 9

Heap

finish at time 9

finish at time 13

x-axis
EXAMPLE: HEAP-BASED ALGORITHM

Min element: finish at time 9

Heap finish at time 9

A1 finish at time 11

A2 finish at time 13

Iteration i=6
Check heap minimum
Check if finish time 5 is before s6
Yes. Reuse colour, deleteMin and insert new finish time into heap!

And so on, and so forth…
A modification is to use the colour of the interval having the earliest finishing time among the most recently chosen intervals of each colour. We can use a priority queue to keep track of these finishing times. Whenever we colour interval $A_i$ with colour $c$, we insert $(f_i, c)$ into the priority queue (here $f_i$ is the "key").

When we want to want to colour the next interval $A_i$, we look at the minimum key $f$ in the priority queue. If $f \leq s_i$, then we do a deleteMin operation, yielding the pair $(f, c)$ and we use colour $c$ for interval $A_i$. If $f > s_i$, we introduce a new colour.

Note that each interval is inserted once and deleted once from the priority queue. Therefore, the complexity of this approach is $O(n \log D)$. Since $D \leq n$, it is $O(n \log n)$. 

Time complexity?

Time?

$O(\log D)$

$O(1)$

$O(\log D)$
Knapsack Problems

Problem 4.4

Knapsack

Instance: Profits $P = [p_1, \ldots, p_n]$; weights $W = [w_1, \ldots, w_n]$; and a capacity, $M$. These are all positive integers.

Feasible solution: An $n$-tuple $X = [x_1, \ldots, x_n]$ where $\sum_{i=1}^{n} w_i x_i < M$.

In the 0-1 Knapsack problem (often denoted just as Knapsack), we require that $x_i \in \{0, 1\}$, $1 \leq i \leq n$.

In the Rational Knapsack problem, we require that $x_i \in \mathbb{Q}$ and $0 \leq x_i \leq 1$, $1 \leq i \leq n$.

Find: A feasible solution $X$ that maximizes $\sum_{i=1}^{n} p_i x_i$. 

NP Hard. Probably requires exponential time to solve...

Can be solved in polynomial time by a greedy alg!
Possible Greedy Strategies for Knapsack Problems

Consider the items in decreasing order of profit (i.e., the local evaluation criterion is $p_i$).

Consider the items in increasing order of weight (i.e., the local evaluation criterion is $w_i$).

Consider the items in decreasing order of profit divided by weight (i.e., the local evaluation criterion is $p_i/w_i$).

Does one of these strategies yield a correct greedy algorithm for the Rational Knapsack problem?
Consider the following instance of the **Rational Knapsack Problem**:

<table>
<thead>
<tr>
<th>profits</th>
<th>50</th>
<th>90</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>weights</td>
<td>50</td>
<td>60</td>
<td>10</td>
</tr>
<tr>
<td>capacity</td>
<td>100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If we consider the objects in order of decreasing profit, we obtain the solution \((4/5, 1, 0)\), yielding profit 130.

If we consider the objects in order of increasing weight, we obtain the solution \((1, 2/3, 1)\), yielding profit 130.

If we consider the objects in order of decreasing profit / weight ratio, then we obtain the solution \((3/5, 1, 1)\), yielding profit 140.

At least for this instance, the third strategy is better than the other two strategies.

It turns out strategy #3 is optimal...
Algorithm: GreedyRationalKnapsack($P, W : array; M : integer$)

sort the items so that $p_1/w_1 \geq \cdots \geq p_n/w_n$

$X \leftarrow [0, \ldots, 0]$  \hspace{2cm} No items are chosen

$i \leftarrow 1$

$CurW \leftarrow 0$  \hspace{2cm} Current weight of knapsack

while $(CurW < M)$ and $(i \leq n)$

\[ \begin{align*}
\text{if } CurW + w_i & \leq M \\
\text{then } & \begin{cases} 
  x_i \leftarrow 1 \\
  CurW \leftarrow CurW + w_i \\
  i \leftarrow i + 1 
\end{cases} \\
\text{else } & \begin{cases} 
  x_i \leftarrow (M - CurW)/w_i \\
  CurW := M 
\end{cases}
\end{align*} \]

\hspace{2cm} Until full, or no more items

\hspace{2cm} If whole item fits

\hspace{2cm} Put it in the knapsack

\hspace{2cm} Else put in as much of the item as you can, to exactly fill the knapsack

\hspace{1cm} Either $X=(1,1,\ldots,1,0,\ldots,0)$ or $X=(1,1,\ldots,1,x_i,0,\ldots,0)$ where $x_i \in (0,1)$

\hspace{2cm} Time complexity?
Correctness Proof

For simplicity, assume that the profit / weight ratios are all distinct, so

\[ \frac{p_1}{w_1} > \frac{p_2}{w_2} > \cdots > \frac{p_n}{w_n}. \]

Suppose the greedy solution is \( X = (x_1, \ldots, x_n) \) and the optimal solution is \( Y = (y_1, \ldots, y_n) \).

We will prove that \( X = Y \), i.e., \( x_j = y_j \) for \( j = 1, \ldots, n \). Therefore there is a unique optimal solution and it is equal to the greedy solution.

Suppose \( X \neq Y \). To obtain a contradiction

Pick the smallest integer \( j \) such that \( x_j \neq y_j \).

It is impossible that \( x_j < y_j \), so we have \( x_j > y_j \).

There exists an index \( k > j \) such that \( y_k > 0 \) (otherwise \( Y \) is not optimal).
Greedy solution $X$ and Optimal solution $Y$ differ at index $j$, where $y_j \neq x_j$. Fraction of item in knapsack.
Correctness Proof

For simplicity, assume that the profit / weight ratios are all distinct, so

\[ \frac{p_1}{w_1} > \frac{p_2}{w_2} > \cdots > \frac{p_n}{w_n}. \]

Suppose the greedy solution is \( X = (x_1, \ldots, x_n) \) and the optimal solution is \( Y = (y_1, \ldots, y_n) \).

We will prove that \( X = Y \), i.e., \( x_j = y_j \) for \( j = 1, \ldots, n \). Therefore there is a unique optimal solution and it is equal to the greedy solution.

Suppose \( X \neq Y \).

Pick the smallest integer \( j \) such that \( x_j \neq y_j \).

It is impossible that \( x_j < y_j \), so we have \( x_j > y_j \).

There exists an index \( k > j \) such that \( y_k > 0 \) (otherwise \( Y \) is not optimal).
Can we have $y_j > x_j$?

No! Greedy would take more item $j$ if it could...

$j = \text{first index where the solutions differ}$
Greedy solution $X$ and Optimal solution $Y$ differ at index $j$. Must have $y_j < x_j$ where $(x_j - y_j)$.
Correctness Proof

For simplicity, assume that the profit / weight ratios are all distinct, so

\[ \frac{p_1}{w_1} > \frac{p_2}{w_2} > \cdots > \frac{p_n}{w_n}. \]

Suppose the greedy solution is \( X = (x_1, \ldots, x_n) \) and the optimal solution is \( Y = (y_1, \ldots, y_n) \).

We will prove that \( X = Y \), i.e., \( x_j = y_j \) for \( j = 1, \ldots, n \). Therefore there is a unique optimal solution and it is equal to the greedy solution.

Suppose \( X \neq Y \).

Pick the smallest integer \( j \) such that \( x_j \neq y_j \).

- It is impossible that \( x_j < y_j \), so we have \( x_j > y_j \).
- There exists an index \( k > j \) such that \( y_k > 0 \) (otherwise \( Y \) is not optimal).
Greedy solution $X$ \hspace{0.5cm} Optimal solution $Y$ \hspace{0.5cm} $j = \text{first index where the solutions differ}$

Can $Y$ be all zeros after $y_j$? \hspace{0.5cm} No! It would be worth less than $X$ \hspace{0.5cm} Must exist $k > j$ such that $y_k > 0$
Greedy solution $X$

Optimal solution $Y$

$j$ = first index where the solutions differ

Must exist $k > j$ such that $y_k > 0$

But, by our sort order, item $j$ is worth more (per unit of weight) than item $k$!

Remove some of item $k$ and replace it with some of item $j$?
\[
\begin{align*}
\text{Let } \delta &= \min\{w_j(x_j - x_i), w_k y_k\} \\
\text{Max weight we can add of this} &= w_j(x_j - y_j) \\
\text{Max weight we can remove of this} &= w_k y_k \\
\delta &= \text{how much weight we will move from } k \to j. \\
\text{Observe that } \delta &> 0 \\
j &= \text{first index where the solutions differ}
\end{align*}
\]
Greedy solution $X$

Optimal solution $Y$

fraction of item in knapsack

$j = \text{first index where the solutions differ}$

Modified optimal solution $Y'$

To move $\delta$ weight from item $j$ to item $k$...

What fraction of item $j$ are we removing?

What fraction of item $k$ are we adding?

What fraction of item $j$ are we adding?

$y_j' = y_j + \frac{\delta}{w_j}$

$y_k' = y_k - \frac{\delta}{w_k}$

$\frac{\delta}{w_k}$
The idea is to show that 

\[ Y' \text{ is feasible, and } \text{profit}(Y') > \text{profit}(Y). \]

This contradicts the optimality of \( Y \) and proves that \( X = Y \).
PROOF CONTINUED

• To show $Y'$ is **feasible**, we show $y'_k \geq 0, y'_j \leq 1$ and $\text{weight}(Y') \leq M$

First, we have $y'_k = y_k - \frac{\delta}{w_k} \geq y_k - \frac{w_k y_k}{w_k} = 0$

Second, we have $y'_j = y_j + \frac{\delta}{w_j} \leq y_j + \frac{w_j (x_j - y_j)}{w_j} = x_j \leq 1$

Third, $\text{weight}(Y') = \text{weight}(Y) + \frac{\delta}{w_j} w_j - \frac{\delta}{w_k} w_k = \text{weight}(Y) \leq M$

Since $\delta = \min\{w_j(x_j - x_i), w_k y_k\}$
PROOF CONTINUED

• Finally we compute \( \text{profit}(Y') \)

• \( \text{profit}(Y') = \text{profit}(Y) + \frac{\delta}{w_j} p_j - \frac{\delta}{w_k} p_k \)

• \( = \text{profit}(Y) + \delta \left( \frac{p_j}{w_j} - \frac{p_k}{w_k} \right) \)

• Since \( j \) is before \( k \), and we consider items with more profit per unit weight first, we have \( \frac{p_j}{w_j} > \frac{p_k}{w_k} \).

• So, \( \delta \left( \frac{p_j}{w_j} - \frac{p_k}{w_k} \right) > 0 \), so \( \text{profit}(Y') > \text{profit}(Y) \).