CS 341: ALGORITHMS

Lecture 6: greedy algorithms II

Readings: see website

Trevor Brown

https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca
OPTIMALITY PROOF
for greedy interval selection
Goal: choose as many disjoint intervals as possible, (i.e., without any overlap)

Algorithm:

3 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 $f_i$). 

![Diagram showing intervals and their selection process.](image-url)
PROVING OPTIMALITY

- Consider an input $A[1..n]$
- Let $G$ be the greedy solution
- Let $O$ be an optimal solution
- “Greedy stays ahead” argument
  - Intuition: out of the a given set of intervals, greedy picks **as many as optimal**
How to compare $G$ and $O$? Imagine reordering $O$ to match $G$!
CRUCIAL: We are NOT assuming the optimal algorithm uses the same sort order!

We are merely imagining reordering the intervals chosen by the optimal algorithm so we can easily compare their finish times to intervals in G.
REORDERING O BY INCREASING FINISH TIME

Now O' and G are both ordered by increasing finish time.
This ordering helps us leverage what we know about G in our comparison with O'.

Argue for a prefix of the intervals sorted this way, G chooses as many as O'.
Looks like $f(G_1) \leq f(O'_1)$ and $f(G_2) \leq f(O'_2)$ \ldots Is $f(G_i) \leq f(O'_i)$ for all $i$?

**If** this trend holds in general, then

out of the intervals with finish time $\leq f(O'_i)$

G chooses **as many** intervals as O!
PROVING LEMMA: \( f(G_i) \leq f(O'_i) \) FOR ALL \( i \)

Base case: \( f(G_1) \leq f(O'_1) \) since \( G \) chooses the interval with the earliest finish time first.
**PROVING LEMA**: \( f(G_i) \leq f(O'_i) \) FOR ALL \( i \)

<table>
<thead>
<tr>
<th>O'</th>
<th>( O'_1 )</th>
<th>( O'_2 )</th>
<th>...</th>
<th>( O'_{i-1} )</th>
<th>( O'_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>( G_1 )</td>
<td>( G_2 )</td>
<td>...</td>
<td>( G_{i-1} )</td>
<td></td>
</tr>
</tbody>
</table>

Inductive step: assume \( f(G_{i-1}) \leq f(O'_{i-1}) \). Show \( f(G_i) \leq f(O'_i) \).

- Since \( O' \) is feasible, \( f(O'_{i-1}) \leq s(O'_i) \)
- So \( f(G_{i-1}) \leq s(O'_i) \)
- So \( G \) can choose \( O'_i \) if it has the smallest finish time
- So \( f(G_i) \leq f(O'_i) \)
Suppose $|O'| > |G|$ to obtain a contradiction

- So if $G$ chooses $k$ intervals, $O'$ chooses at least $k + 1$
- By the lemma, $f(G_k) \leq f(O_k)$
- Since $O'$ is feasible, $f(O'_k) \leq s(O'_{k+1})$
- But then $G$ can, and would, pick $O'_{k+1}$.
  - Contradiction!

So assumption $|O'| > |G|$ is wrong!
A DIFFERENT PROOF

“Slick” ad-hoc approaches are sometimes possible...
Let $F = \{f_{i_1}, \ldots, f_{i_k}\}$ be the finishing times of the intervals in $X$

No interval finishes strictly to the left

No interval in is strictly between these points!

No interval starts strictly to the right

Would be chosen by greedy! (contradiction)

So, in addition to the intervals in $X$, only the following types of intervals are possible:

Contains $f_{i_1}$

Contains $f_{i_2}$

Contains $f_{i_1}$ and $f_{i_2}$

Thus, every interval contains some finishing time in $F$

And, two intervals in $O$ cannot contain the same element of $F$

So, there must be as many finishing times in $F$ as there are intervals in $O$. QED
KNAPSACK PROBLEMS
Gotta respect the weight limit $M$...
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 $\sum_{i=1}^{n} w_i x_i \leq M$.

*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 $\sum_{i=1}^{n} p_i x_i$.

---

0-1 Knapsack: NP Hard. Probably requires exponential time to solve...

Rational knapsack: Can be solved in polynomial time by a greedy alg!

Lets discuss this now… other one later
POSSIBLE GREEDY STRATEGIES FOR KNAPSACK PROBLEMS

- **Strategy 1**: consider items in **decreasing** order of profit (i.e., we maximize the local evaluation criterion \( p_i \))

- Let’s try an example input
  - Profits \( P = [20, 50, 100] \)
  - Weights \( W = [10, 20, 10] \)
  - Weight limit \( M = 10 \)

- Algorithm selects last item for 100 profit
  - Looks optimal in this example
POSSIBLE GREEDY STRATEGIES FOR KNAPSACK PROBLEMS

- **Strategy 1**: consider items in **decreasing** order of **profit** (i.e., we maximize the local evaluation criterion $p_i$)

- How about a **second example input**
  - Profits $P = [20, 50, 100]$
  - Weights $W = [10, 20, 100]$
  - Weight limit $M = 10$
  - Algorithm selects last item for **10** profit
    - **Not optimal!**
POSSIBLE GREEDY STRATEGIES FOR KNAPSACK PROBLEMS

- **Strategy 2:** consider items in **increasing** order of **weight**
  (i.e., we minimize the local evaluation criterion $w_i$)

- **Counterexample**
  - Profits $P = [20,50,100]$
  - Weights $W = [10,20,100]$
  - Weight limit $M = 10$

  - Algorithm selects first item for 20 profit
    - It **could** select half of second item, for 25 profit!
POSSIBLE GREEDY STRATEGIES FOR KNAPSACK PROBLEMS

- **Strategy 3**: consider items in **decreasing** order of profit divided by weight (i.e., we maximize local evaluation criterion $p_i/w_i$)

- Let’s try our first example input
  - Profits $P = [20, 50, 100]$
  - Weights $W = [10, 20, 10]$
  - Weight limit $M = 10$
  - Profit divided by weight
    - $P/W = [2, 2.5, 10]$
  - Algorithm selects last item for 100 profit (optimal)
POSSIBLE GREEDY STRATEGIES FOR KNAPSACK PROBLEMS

- **Strategy 3**: consider items in **decreasing** order of **profit divided by weight** (i.e., we maximize local evaluation criterion \( p_i/w_i \))

- Let’s try our second example input
  - Profits \( P = [20, 50, 100] \)
  - Weights \( W = [10, 20, 100] \)
  - Weight limit \( M = 10 \)
  - Profit divided by weight
    - \( P/W = [2, 2.5, 1] \)
  - Algorithm selects second item for 25 profit (optimal)

It turns out strategy #3 is optimal...
No items are chosen yet

Current weight of knapsack

For all items

If we cannot fit the entire item

Put in as much of the item as you can, to exactly fill the knapsack

Otherwise take the entire item

Either \( X = (1, 1, \ldots, 1, 0, \ldots, 0) \) or \( X = (1, 1, \ldots, 1, x_i, 0, \ldots, 0) \) where \( x_i \in (0, 1) \)
Preprocess(A[1..n], M) // A[i] = (p_i, w_i)
  sort A by decreasing profit divided by weight
  let p[1..n] be the profits in A
  let w[1..n] be the weights in A
  return GreedyRationalKnapsack(p, w, M)

GreedyRationalKnapsack(p[1..n], w[1..n], M)
  X = [0, ..., 0]
  weight = 0
  for i = 1..n
      if weight + w[i] > M then
          X[i] = (M - weight) / w[i]
          break
      else
          X[i] = 1
          weight = weight + w[i]
  return X

Running time complexity?
Can do preprocessing in \( \Theta(n \log n) \)
Create array in \( \Theta(n) \) time
\( \Theta(n) \) iterations each doing \( \Theta(1) \) work
Total \( \Theta(n \log n) \) (or \( \Theta(n) \) if input is already sorted)
**INFORMAL FEASIBILITY ARGUMENT**

*(SHOULD BE GOOD ENOUGH TO SHOW FEASIBILITY ON ASSESSMENTS)*

- **Feasibility:** all $x_i$ are in $[0, 1]$ and total weight is $\leq M$
- Either everything fits in the knapsack, or:
- When we exit the loop, **weight is exactly** $M$
- Every time we write to $x_i$ it’s either 0, 1 or $(M - \text{weight})/w_i$ where $\text{weight} + w[i] > M$
  - Rearranging the latter we get $(M - \text{weight})/w_i < 1$
  - And $\text{weight} \leq M$, so $(M - \text{weight})/w_i \geq 0$
- **So, we have** $x_i \in [0, 1]$

```
for i = 1..n
  if weight + w[i] > M then
    X[i] = (M - weight) / w[i]
    break
  else
    X[i] = 1
    weight = weight + w[i]
```

24
MINOR MODIFICATION TO FACILITATE **FORMAL PROOF**

```plaintext
GreedyRationalKnapsack(p[1..n], w[1..n], M)

X = [0, ..., 0]
weight = 0
for i = 1..n
  if weight + w[i] > M then
    X[i] = (M - weight) / w[i]
    weight = M
    break
  else
    X[i] = 1
    weight = weight + w[i]
return X
```

Optional slide, just for your notes

Does NOT change behaviour of the algorithm at all!
FORMAL FEASIBILITY ARG

- Loop invariant: $\forall i : x_i \in [0,1]$ and $\text{weight} = \sum_{i=1}^{n} w_i x_i \leq M$

- Base case. Initially weight = 0 and $\forall i : x_i = 0$.
  - So $0 = \text{weight} = \sum_{i=1}^{n} w_i \cdot 0 = \sum_{i=1}^{n} w_i x_i \leq M$

- Inductive step.
  - Suppose invariant holds at start of iteration $i$
  - Let $\text{weight}', x_i'$ denote values of $\text{weight}, x_i$ at end of iteration $i$
  - Prove invariant holds at end of iteration $i$
  - i.e., $\forall i : x_i' \in [0,1]$ and $\text{weight}' = \sum_{i=1}^{n} w_i x_i' \leq M$

```plaintext
for i = 1..n
  if weight + w[i] > M then
    X[i] = (M - weight) / w[i]
    weight = M
    break
  else
    X[i] = 1
    weight = weight + w[i]
```

Optional slide, just for your notes
FORMAL FEASIBILITY ARG

- WTP: \( \forall i : x'_i \in [0, 1] \)
  and weight' = \( \sum_{i=1}^{n} w_i x'_i \leq M \)

- Case 1: weight + \( w_i \) \( \leq M \)
  - \( x'_i = 1 \) which is in \([0, 1]\) (by line 11)
  - weight' = weight + \( w_i \) (by line 12)
    and this is \( \leq M \) by the case
  - weight' = \( \sum_{k=1}^{n} x_k w_k + w_i \) (by invariant)
  - weight' = \( \sum_{k=1}^{n} x_k w_k + x'_i w_i \) (since \( x'_i = 1 \))
  - And \( x'_k = x_k \) for all \( k \neq i \) and \( x_i = 0 \) so \( \sum_{k=1}^{n} x'_k w_k = x'_i w_i + \sum_{k=1}^{n} x_k w_k \)
  - Rearrange to get \( \sum_{k=1}^{n} x_k w_k = (\sum_{k=1}^{n} x'_k w_k - x'_i w_i) \)
  - So \( \text{weight}' = (\sum_{k=1}^{n} x'_k w_k - x'_i w_i) + x'_i w_i = \sum_{k=1}^{n} x'_k w_k \)
\textbf{FORMAL FEASIBILITY ARG}

- **WTP:** \( \forall i : x_i' \in [0, 1] \)
  and \( \text{weight}' = \sum_{i=1}^{n} w_i x_i' \leq M \)
- **Case 2:** \( \text{weight} + w_i > M \)
  - We have \( w_i > M - \text{weight} \) \text{(by case)}
    and \( M - \text{weight} \geq 0 \) \text{(by invariant)}
  - So \( 0 \leq \frac{M - \text{weight}}{w_i} < 1 \) which means \( x_i' \in [0, 1) \)
  - \( \text{weight}' = M = \text{weight} + (M - \text{weight}) \) \text{(by line 8)}
  - \( \text{weight}' = \sum_{k=1}^{n} x_k w_k + (M - \text{weight}) \) \text{(by invariant)}
  - But \( x_k' = x_k \) for all \( k \neq i \) and \( x_i = 0 \) so \( \sum_{k=1}^{n} x_k w_k = x_i' w_i + \sum_{k=1}^{n} x_k w_k \)
  - Rearrange to get \( \sum_{k=1}^{n} x_k w_k = (\sum_{k=1}^{n} x_k w_k - x_i' w_i) \)
  - So \( \text{weight}' = (\sum_{k=1}^{n} x_k w_k - x_i' w_i) + (M - \text{weight}) \)
  - And \( M - \text{weight} = x_i' w_i \) so \( \text{weight}' = \sum_{k=1}^{n} x_k' w_k \)

\begin{verbatim}
for i = 1..n
  if weight + w[i] > M then
    X[i] = (M - weight) / w[i]
    weight = M
    break
  else
    X[i] = 1
    weight = weight + w[i]
\end{verbatim}
EXCHANGE ARGUMENT for proving optimality
OPTIMALITY – AN EXCHANGE ARGUMENT

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

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

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 \( x_j \neq y_j \). \( X \) and \( Y \) are identical up to \( x_j \) and \( y_j \), respectively.
What’s the relationship between $x_j$ and $y_j$?
Can we have $y_j > x_j$?

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

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

Optimal solution $Y$

$j =$ first index where the solutions differ

Must have $y_j < x_j$

$(x_j - y_j)$
Greedy solution \( X \)

Optimal solution \( Y \)

\[ x_1 \quad x_2 \quad \cdots \quad x_{j-1} \quad x_j \]

\[ y_1 \quad y_2 \quad \cdots \quad y_{j-1} \quad y_j \]

\( j = \) first index where the solutions differ

Can \( Y \) be all zeros after \( y_j \)?

No! It would be worth less than \( X \)
Greedy solution $X$

Optimal solution $Y$

Fraction of item in knapsack

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 $j$?

How much of item $k$ should we remove?
Since item j is worth more per unit weight, replacing even a tiny amount of item k with item j will improve the solution.

So, we remove an infinitesimal $\delta > 0$ of weight of item k, and add $\delta$ weight of item j.
Greedy solution \( X \)

Modified optimal solution \( Y' \)

j = first index where the solutions differ

To move \( \delta \) weight from item \( k \) to item \( j \)...

What fraction of item \( j \) are we adding?

What fraction of item \( k \) are we removing?

\[
y_j' = y_j + \frac{\delta}{w_j}
\]

\[
y_k' = y_k - \frac{\delta}{w_k}
\]
The idea is to show that

\( Y' \) is feasible, and

\[
\text{profit}(Y') > \text{profit}(Y).
\]

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

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

- To show $Y'$ is feasible, we show $y'_k \geq 0$, $y'_j \leq 1$ and $\text{weight}(Y') \leq M$
- Let's show $y'_k \geq 0$
  - By definition, $y'_k = y_k - \frac{\delta}{w_k}$
  - So, $y'_k \geq 0$ iff $y_k - \frac{\delta}{w_k} \geq 0$ iff $\delta \leq y_k w_k$
  - And we know $y_k$ and $w_k$ are both positive
  - So, this constrains $\delta$ to be smaller than this positive number
  - Therefore, it is possible to choose positive $\delta$ s.t. $y'_k \geq 0$

Existence proof, but a non-constructive one
FEASIBILITY OF $Y'$

- To show $Y'$ is feasible, we show $y_k' \geq 0, y_j' \leq 1$ and $\text{weight}(Y') \leq M$
- Now let’s show $y_j' \leq 1$
  - By definition, $y_j' = y_j + \frac{\delta}{w_j}$
  - So, $y_j' \leq 1$ iff $y_j + \frac{\delta}{w_j} \leq 1$ iff $\delta \leq (1 - y_j)w_j$
  - Recall $y_j < x_j$, so $y_j < 1$, which means $(1 - y_j) > 0$
  - So, this constrains $\delta$ to be smaller than some positive number
Finally, we show $\text{weight}(Y') \leq M$.

Recall changes to get $Y'$ from $Y$:

- We move $\delta$ weight from item $k$ to item $j$.
- This does not change the total weight!

So $\text{weight}(Y') = \text{weight}(Y) \leq M$.

Therefore, $Y'$ is feasible!
SUPERIORITY OF $Y'$

- 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, if $\delta > 0$ then $\delta \left( \frac{p_j}{w_j} - \frac{p_k}{w_k} \right) > 0$

- Since we can choose $\delta > 0$, we have $\text{profit}(Y') > \text{profit}(Y)$.

(Fraction of item $j$ added) $\times$ (profit for item $j$)

(Fraction of item $k$ removed) $\times$ (profit for item $k$)

Contradicts optimality of $Y$! So assumption $X \neq Y$ is bad. Therefore, $X$ is optimal.
WHAT IF ELEMENTS DON’T HAVE DISTINCT PROFIT/WEIGHT RATIOS?
There may be many optimal solutions

**Key idea:** Let $Y$ be an optimal solution that matches $X$ on a maximal number of indices.

**Observe:** if $X$ is really optimal, then $Y = X$

Suppose not for contra

- We will modify $Y$, preserving its optimality, but making it match $X$ on **one more index** (a contradiction!)
Greedy solution $X$

Optimal solution $Y$

Fraction of item in knapsack

$x_1$, $x_2$, \ldots, $x_{j-1}$, $x_j$

$y_1$, $y_2$, \ldots, $y_{j-1}$

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

$y_j \neq x_j$
fraction of item in knapsack

Greedy solution $X$

- $x_1$, $x_2$, ..., $x_j$

Optimal solution $Y$

- $y_1$, $y_2$, ..., $y_{j-1}$

Must have $y_j < x_j$
Must exist $k > j$ such that $y_k > x_k$ because weight of $X$ and $Y$ must be the same.

Remove some weight $\delta$ of item $k$ and add the same weight of item $j$ with the goal of making the solutions equal on index $k$ or index $j$.

Let $\delta = \min\{w_j(x_j - y_j), w_k(y_k - x_k)\}$ observe $\delta > 0$. 

Fraction we should add to $j$ to make solutions equal on index $j$: $x_j - y_j$ 

Fraction we should remove from $k$ to make solutions equal on index $k$: $y_k - x_k$ 

Weight to add: $w_j(x_j - y_j)$ 

Weight to remove: $w_k(y_k - x_k)$
Greedy solution $X$

<table>
<thead>
<tr>
<th>Item 1</th>
<th>Item 2</th>
<th>...</th>
<th>Item j</th>
<th>Item k</th>
<th>Item n</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
<td>...</td>
<td>$x_{j-1}$</td>
<td>$x_j$</td>
<td>$x_k$</td>
</tr>
</tbody>
</table>

Optimal solution $Y$

<table>
<thead>
<tr>
<th>Item 1</th>
<th>Item 2</th>
<th>...</th>
<th>Item j</th>
<th>Item k</th>
<th>Item n</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>$y_2$</td>
<td>...</td>
<td>$y_{j-1}$</td>
<td>$y_j$</td>
<td>$y_k$</td>
</tr>
</tbody>
</table>

Suppose $\delta = w_k(y_k - x_k)$

Modified optimal solution $Y'$

<table>
<thead>
<tr>
<th>Item 1</th>
<th>Item 2</th>
<th>...</th>
<th>Item j</th>
<th>Item k</th>
<th>Item n</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1'$</td>
<td>$y_2'$</td>
<td>...</td>
<td>$y_{j-1}'$</td>
<td>$y_j'$</td>
<td>$y_k'$</td>
</tr>
</tbody>
</table>

In this case, since $\delta = w_k(y_k - x_k)$, we end up with $y_k' = x_k$

If $\delta$ were $w_j(x_j - y_j)$, we would have $y_j' = x_j$
To show $Y'$ is feasible, we show \(\text{weight}(Y') \leq M\) and \(y_k' \geq 0, y_j' \leq 1\)

**Weight**
We move $\delta$ weight from item $k$ to item $j$
This does not change the total weight!
So \(\text{weight}(Y') = \text{weight}(Y) = M\)
FEASIBILITY OF $Y'$

- Showing $y'_k \geq 0$
  - By definition, $y'_k = y_k - \frac{\delta}{w_k} \geq 0$ iff $\delta \leq y_k w_k$
  - But $\delta$ is the minimum of $w_j(x_j - y_j)$ and $w_k(y_k - x_k) \leq w_k y_k$
  - And $w_k(y_k - x_k) \leq w_k y_k$ so $\delta \leq y_k w_k$
- Showing $y'_j \leq 1$
  - $y'_j = y_j + \frac{\delta}{w_j} \leq 1$ iff $\frac{\delta}{w_j} \leq 1 - y_j$ iff $\delta \leq w_j(1 - y_j)$ (rearranging)
  - $\delta \leq w_j(x_j - y_j)$ (definition of $\delta$)
  - and $w_j(x_j - y_j) \leq w_j(1 - y_j)$ (by feasibility of $X$, i.e., $x_j \leq 1$)
**PROFIT OF** $Y'$

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

- Since $j$ is before $k$, and we consider items with more profit per unit weight first, we have $\frac{p_j}{w_j} \geq \frac{p_k}{w_k}$.

- Since $\delta > 0$ and $\frac{p_j}{w_j} \geq \frac{p_k}{w_k}$, we have $\delta \left( \frac{p_j}{w_j} - \frac{p_k}{w_k} \right) \geq 0$.

- Since $Y$ is optimal, this **cannot be positive**.

- So $Y'$ is a new optimal solution that **matches $X$ on one more index than $Y$**.

- Contradiction: $Y$ matched $X$ on a **maximal** number of indices!
SUMMARIZING EXCHANGE ARGUMENTS

- If inputs are distinct
  - So there is a unique optimal solution
  - Let $O \neq G$ be an optimal solution that beats greedy
  - Show how to change $O$ to obtain a better solution

- If not
  - There may be many optimal solutions
  - Let $O \neq G$ be an optimal solution that matches greedy on as many choices as possible
  - Show how to change $O$ to obtain an optimal solution $O'$ that matches greedy for even more choices
I don’t think we will have time to get past here.
But if we do, great, we can catch up.
INTERVAL COLOURING
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.
### More Examples

<table>
<thead>
<tr>
<th>Example</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not feasible!</td>
</tr>
<tr>
<td></td>
<td>Same color, but disjoint. <strong>OK!</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7 intervals, 6 colours. <strong>Feasible, but not optimal</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7 intervals, 2 colours. <strong>Optimal</strong></td>
</tr>
</tbody>
</table>
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: \(A_1 \ldots A_{10}\)
**EXAMPLE:**

**ORDER MATTERS!**

<table>
<thead>
<tr>
<th>A_1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_2</td>
<td></td>
</tr>
<tr>
<td>A_3</td>
<td></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>

![Diagram with ordered elements](image-url)
EXAMPLE: ORDER MATTERS!
**EXAMPLE:**

**ORDER MATTERS!**

<table>
<thead>
<tr>
<th>A_1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_2</td>
<td>1</td>
</tr>
<tr>
<td>A_3</td>
<td></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>

![Diagram showing order matters example](image-url)
EXAMPLE: ORDER MATTERS!
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>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></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

63
EXAMPLE: ORDER MATTERS!
EXAMPLE: ORDER MATTERS!
EXAMPLE:
ORDER
MATTERS!
EXAMPLE:
ORDER MATTERS!

| A1 | 1 |
| A2 | 1 |
| A3 | 2 |
| A4 | 2 |
| A5 | 3 |
| A6 | 2 |
| A7 | 4 |
| A8 | 4 |
| A9 | 2 |
| A10| 1 |

x-axis

0 2 4 6 8 10 12 14 16 18 20
**EXAMPLE:**
**ORDER MATTERS!**

<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</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td>4</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Used 4 colours**

**Can we do better?**

---

*x-axis*
EXAMPLE:
ORDER MATTERS!

Pre-sort intervals by increasing start time!
EXAMPLE:
ORDER MATTERS!

Pre-sort intervals by increasing start time!
EXAMPLE: ORDER MATTERS!
EXAMPLE:
ORDER MATTERS!
EXAMPLE: ORDER MATTERS!
EXAMPLE: ORDER MATTERS!
**Example:**

Order matters!
EXAMPLE: ORDER MATTERS!
EXAMPLE:
ORDER
MATTERS!
EXAMPLE: ORDER MATTERS!
**EXAMPLE:**

**ORDER MATTERS!**

<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>4</td>
</tr>
<tr>
<td>A_5</td>
<td>5</td>
</tr>
<tr>
<td>A_6</td>
<td>6</td>
</tr>
<tr>
<td>A_7</td>
<td>7</td>
</tr>
<tr>
<td>A_8</td>
<td>8</td>
</tr>
<tr>
<td>A_9</td>
<td>9</td>
</tr>
<tr>
<td>A_10</td>
<td>10</td>
</tr>
</tbody>
</table>

![Chart showing the concept of order matters with different colored segments representing different elements over a timeline.](chart.png)
EXAMPLE: ORDER MATTERS!

Used 3 colours

Can we do better?
$d = \# \text{ of colours used so far}$

$\text{finish}[c] = \text{finish time of last interval to receive colour } c$

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.
EXAMPLE: RUNNING GREEDY

Initial state
EXAMPLE: RUNNING GREEDY

Code before the loop: just assign colour 1

Code:  
\begin{align*}
\text{d} & = 1 \\
\text{finish}[1] & = 83
\end{align*}
## EXAMPLE: RUNNING GREEDY

**Diagram Description:**
- The diagram illustrates the process of assigning colors to points on a timeline based on a greedy algorithm.
- Each point (A1 to A10) represents a segment on the x-axis.
- The color assignment is indicated by different shades of blue.
- The algorithm checks if it can reuse a color for a given point and updates the finish condition accordingly.

**Algorithm Steps:**
1. **Initialization:**
   - Assume `finish[1] = 0` to start.

2. **Iteration:**
   - For each point `A_i`:
     2. If true, reuse color 1.
     3. If false, cannot reuse color 1.
     4. If no reuse, create a new color.

**Example:**
- Consider point `A_1`.
- Check if it can reuse color 1.
- If not, assign a new color.

**Check Points:**
- Point `A_2`:
  - Check if it can reuse color 1.
  - If not, assign a new color.

**Conclusion:**
- The algorithm iterates over each point, ensuring that colors are assigned efficiently without conflicts.

**Formal Note:**
- The image includes placeholders for `i=2`, `d=2`, and `finish[1]=`. These variables are used to demonstrate the algorithm's flow.
- The algorithm iterates over `c` to check for color reuse.
- The condition `Is finish[1] <= s_2?` is evaluated at each step to determine color reuse.

**Key Points:**
- **Is finish[1] ≤ s_2?**
  - No. We cannot reuse colour 1.
  - Cannot reuse any colour. Create a new one!
EXAMPLE: RUNNING GREEDY

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

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

$\text{Cannot reuse any colour. Create a new one!}$
EXAMPLE: RUNNING GREEDY

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

<table>
<thead>
<tr>
<th>A_1</th>
<th>1</th>
<th>finish[1]</th>
<th>finish[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_3</td>
<td></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>

Is $finish[1] \leq s_3$?
No. We cannot reuse colour 1.

Is $finish[2] \leq s_3$?
No. We cannot reuse colour 2.

Cannot reuse any colour. Create new one.

x-axis

i=3
d=2
EXAMPLE: RUNNING GREEDY

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

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

Is finish[1] ≤ s₃?
No. We cannot reuse colour 1.

Is finish[2] ≤ s₃?
No. We cannot reuse colour 2.

Cannot reuse any colour. Create new one.
**EXAMPLE:**

**RUNNING GREEDY**

<table>
<thead>
<tr>
<th></th>
<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></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>
</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>
</tr>
</tbody>
</table>

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.
EXAMPLE: RUNNING GREEDY

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

<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>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Is $finish[1] \leq s_4$?
Yes. We can reuse colour 1.

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

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

<table>
<thead>
<tr>
<th>$i$ = 5</th>
<th>$d$ = 3</th>
<th>$\text{finish}[1]$ =</th>
<th>$\text{finish}[2]$ =</th>
<th>$\text{finish}[3]$ =</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_2$</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_7$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_8$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_9$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
EXAMPLE: RUNNING GREEDY

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

\[
\begin{array}{c|c|c|c|c}
\text{A}_1 & \text{A}_2 & \text{A}_3 & \text{A}_4 & \text{A}_5 \\
1 & 2 & 3 & & \\
\text{A}_6 & \text{A}_7 & \text{A}_8 & \text{A}_9 & \text{A}_{10} \\
& & & & \\
\end{array}
\]

\( i = 5 \)
\( d = 3 \)

\( \text{finish}[1] = \) 
\( \text{finish}[2] = \) 
\( \text{finish}[3] = \)

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.
**Example:** Running \textit{Greedy}

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>

<table>
<thead>
<tr>
<th>$i=6$</th>
<th>$d=3$</th>
<th>$\text{finish}[1]=$</th>
<th>$\text{finish}[2]=$</th>
<th>$\text{finish}[3]=$</th>
</tr>
</thead>
</table>

Is $\text{finish}[1] \leq s_6$?

No. We \textbf{cannot} reuse colour 1.

Is $\text{finish}[2] \leq s_6$?

Yes. We \textbf{can} reuse colour 2.
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.

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.

Let $F_D$ be the first interval that has the last colour $D$

Let $L_c$ be the last interval that has colour $c$ and starts before $F_D$ ends

We prove $F_D$ overlaps every interval $L_c$ for all $c < D$

If $D = 1$ the proof is immediate, so suppose $D \geq 2$
Let $F_D$ be the first interval that has colour $D$

Let $L_c$ be the last interval that has colour $c$ and starts before $F_D$ ends

We prove $F_D$ overlaps every interval $L_c$ for all $c < D$
TIME COMPLEXITY?

Preprocess(A[1..n])
  sort A by increasing start time
  let s[1..n] be the start times in A
  let f[1..n] be the finish times in A
  return GreedyIntervalColouring(s, f)

GreedyIntervalColouring(s[1..n], f[1..n])
  d = 1
  colour[1] = 1
  finish[1] = f[1]
  for i = 2..n
    reused = false
    for c = 1..d
      if finish[c] <= s[i] then
        colour[i] = c
        finish[c] = f[i]
        reused = true
        break
    if not reused then
      d++
      colour[i] = d
      finish[d] = f[i]
  return d

O(n log n)

Total $O(n \log n + nd)$

Could be $O(n \log n)$ if only a constant number of colours are needed
(or even $\log n$ colours!)

Could be $O(n^2)$ if $n$ colours are needed

Most accurate complexity statement is $\Theta(n \log n + nD)$ where $D$ is # colours used

What inefficiencies exist in this algorithm? Could we make it faster with clever data structure usage?
IMPROVING THIS ALGORITHM

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

Heap

Min element: NULL
**EXAMPLE:**
**HEAP-BASED ALGORITHM**

**Min element:** finish at time 3

**Heap** finish at time 3

---

**Iteration i=1**

Check heap minimum

Empty, so a new colour is needed

---

X-axis
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!

Heap

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Check heap minimum

Check if finish time 3 is before $s_2$

No. New colour!

$x$-axis
EXAMPLE: HEAP-BASED ALGORITHM

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

Finish at time 7

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

<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>

x-axis

0  2  4  6  8  10  12  14  16  18  20

Check heap minimum
Check if finish time 3 is before s_2
No. New colour!
**EXAMPLE: HEAP-BASED ALGORITHM**

**Min element:** finish at time 3

**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>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Check heap minimum

Check if finish time 3 is before $s_3$

No. New colour!

---

**Iteration i=3**

Check heap minimum

Check if finish time 3 is before $s_3$

No. New colour!

---

Min element: finish at time 3

Heap finish at time 3

finish at time 7

---

**x-axis**

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
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!

A_1 1
A_2 2
A_3 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

A_1 finish at time 3

A_2 finish at time 3

A_3 finish at time 3

A_4 finish at time 7

A_5 finish at time 5

A_6

A_7

A_8

A_9

A_{10}

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

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Check heap minimum</th>
<th>Check if finish time 3 is before $s_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i=4$</td>
<td></td>
<td>Yes. <strong>Reuse</strong> colour, <strong>deleteMin</strong> and <strong>insert</strong> new finish time into heap!</td>
</tr>
</tbody>
</table>

---

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

---

**x-axis**
**EXAMPLE: HEAP-BASED ALGORITHM**

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

**Heap**
- finish at time 7
- finish at time 5

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

<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>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

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

**HEAP-BASED ALGORITHM**

<table>
<thead>
<tr>
<th>Min element:</th>
<th>finish at time 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>finish at time 9</td>
</tr>
<tr>
<td></td>
<td>finish at time 7</td>
</tr>
<tr>
<td></td>
<td>finish at time 5</td>
</tr>
</tbody>
</table>

**Iteration i=4**

- Check heap minimum
- Check if finish time 3 is before $s_4$

<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>

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

**HEAP-BASED ALGORITHM**

Min element: finish at time 5

Heap:
- Finish at time 9
- Finish at time 7
- Finish at time 5

<table>
<thead>
<tr>
<th>Iteration i=5</th>
<th>Check heap minimum</th>
<th>Check if finish time 5 is before $s_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

---

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

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

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

<table>
<thead>
<tr>
<th>Iteration i=5</th>
<th>Check heap minimum</th>
<th>Check if finish time 5 is before $s_5$</th>
<th>Yes. Reuse colour, deleteMin and insert new finish time into heap!</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₁</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A₂</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A₃</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A₄</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A₅</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A₆</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A₇</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A₈</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A₉</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A₁₀</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**x-axis**

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
<th>20</th>
</tr>
</thead>
</table>

Check heap minimum  
Check if finish time 5 is before $s_5$
EXAMPLE: HEAP-BASED ALGORITHM

Min element: finish at time 7

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>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

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 7

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>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

x-axis

0  2  4  6  8  10  12  14  16  18  20

finish at time 9

finish at time 7

finish at time 13
EXAMPLE: HEAP-BASED ALGORITHM

Min element: finish at time 7

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>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Check heap minimum

Check if finish time 7 is before $s_6$

Iteration i=6

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

**HEAP-BASED ALGORITHM**

### Min element:
- Finish at time 9

### Heap:
- Finish at time 9
- Finish at time 13

### Table:

<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=6**

Check heap minimum

Check if finish time 7 is before $s_6$

---

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

Min element: finish at time 9

Heap
- A1 finish at time 9
- A2 finish at time 11
- A3 finish at time 13

Iteration i=6
Check heap minimum
Check if finish time 7 is before $s_6$

And so on, and so forth...
Preprocess(A[1..n])
sort A by increasing start time
let s[1..n] be the start times in A
let f[1..n] be the finish times in A
return GreedyIntervalColouring(s, f)

GreedyIntervalColouring(s[1..n], f[1..n])
d = 1
colour[1] = 1
h = new minPQ
h.insert([[f[1], colour[1]]])

for i = 2..n
  (fc, c) = h.min()
  if fc <= s[i] then
    h.deleteMin()
    colour[i] = c
  else
    d++
    colour[i] = d
    h.insert([[f[i], colour[i]]])

return d

$O(\log S)$ where $S = \text{size(priority queue)}$

$O(1)$

$O(\log D)$

Total $\Theta(n \log n) + \Theta(n \log D)$

Since $n \geq D$, $\Theta(n \log n)$