Goal: choose as many disjoint intervals as possible, (i.e., without any overlap)

Algorithm:
1. 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 $f_i$).

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 given set of intervals, greedy picks as many as optimal

VISUAL EXAMPLE
Input: 
```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
```

$G$: $G_1$, $G_2$, $G_3$, $G_4$, $G_5$

$O$: $O_1$, $O_2$, $O_3$, $O_4$, $O_5$

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 $G$. 
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 $O'$.

**COMPARING $O'$ WITH $G$**

Looks like $f(G_1) \leq f(O'_1)$ and $f(G_2) \leq f(O'_2)$... 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:**

Base case: $f(G_1) \leq f(O'_1)$ since $G$ chooses the interval with the earliest finish time first.

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 $f(G_i) \leq f(O'_i)$

**USING THIS LEMMA**

- 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!

A DIFFERENT PROOF

"Slick" ad-hoc approaches are sometimes possible...
Let $F = \{f_1, \ldots, f_n\}$ be the finishing times of the intervals in $X$.

No interval finishes strictly to the left

No interval starts strictly to the right

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

- Contains $f_i$
- Contains $f_j$
- Contains $f_i$ and $f_j$

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

**Possibility Greedy Strategies for Knapsack Problems**

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

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 $p_j$)

How about a second example input:

- Profits $P = [20, 50, 100]$
- Weights $W = [10, 20, 10]$
- 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 \( W = [10, 20, 100] \)
  - Weight limit \( M = 10 \)
  - Profit divided by weight
    - \( P/W = [2, 2.5, 1] \)
  - Algorithm selects last item for 100 profit (optimal)

It turns out strategy #3 is optimal…

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 \( \text{weight} + w_i \leq M \)
  - Rearranging the latter we get \((M - \text{weight})/w_i < 1\)
  - And weight \( \leq M \), so \((M - \text{weight})/w_i \geq 0\)
  - So, we have \( x_i \in [0, 1] \)

Running time complexity?

Can do preprocessing in \( \Theta(n\log n) \) time.

Total \( \Theta(n) \) (or \( \Theta(n\log n) \) if input is already sorted)

Preprocess \( A[i, n] \)

GreedyKnap(sack \( p[i, n], w[i, n], W \))

\( X = [0, 0, \ldots, 0] \)

\( \text{Create array} \)

\( \text{Sort items by decreasing profit divided by weight} \)

\( \text{For all items} \)

\( \text{If we cannot fit the entire item} \)

\( \text{Put in as much of the item} \)

\( \text{Otherwise take the entire item} \)

\( X[i] = 1 \)

\( \text{Let } p[i, n] \text{ be the profits in } A \)

\( \text{Let } w[i, n] \text{ be the weights in } A \)

\( \text{Return GreedyKnap(sack}(p[i, n], w[i, n], W) \text{)} \)

Feasibility: all intervals 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 \( \text{weight} + w_i \leq M \)

Rearranging the latter we get \((M - \text{weight})/w_i < 1\)

And weight \( \leq M \), so \((M - \text{weight})/w_i \geq 0\)

So, we have \( x_i \in [0, 1] \)
**MINOR MODIFICATION TO FACILITATE FORMAL PROOF**

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

**FORMAL FEASIBILITY ARG**

- Loop Invariant: \( \forall j : x_j \in [0, 1] \) and \( \text{weight} = \sum_{i=1}^{n} w_i x_i \leq M \)
- 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 weight' = \( \sum_{k=1}^{n} x_k w_k - x_i w_i \) + \( x_i w_i \) = \( \sum_{k=1}^{n} x_k w_k \)

- Case 2: weight = \( w_i > M \)
  - We have \( w_i > M = \text{weight} \) and \( M = \text{weight} \geq 0 \)
  - So \( 0 \leq \text{weight} < 1 \) which means \( x_i \neq 0 \) (by invariant)
  - weight' = \( \text{weight} + (M - \text{weight}) \) (by line 8)
  - weight' = \( \sum_{k=1}^{n} x_k w_k + (M - \text{weight}) \) (by invariant)
  - But \( x_i = x_i \) 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 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 weight' = \( \sum_{k=1}^{n} x_k w_k \)

**OPTIMALITY – AN EXCHANGE ARUGMENT**

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

\[
p_1 > p_2 > \ldots > p_n
\]

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 \( y_j \), and \( y_j > x_j \), 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.

Must have $y_j < x_j$

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

So, we remove an infinitesimal $\delta > 0$ amount of item $k$, and add $\delta$ weight of item $j$. 
Recall changes to get $Y'$:

By definition, $\delta = \frac{w_j}{y_j}$. We move $y_j$ weight from item $k$ to item $j$.

$Y'$ is feasible, so $y'_j > 0$ for all $j$. $Y'$ is a feasible solution.

FEASIBILITY OF $Y'$

- To show $Y'$ is feasible, we show $y'_j \geq 0, y'_i \leq 1$ and weight($Y'$) $\leq M$
- Let's show $y'_j \geq 0$
  - By definition, $y'_j = y_j - \delta$
  - So, $y'_j \geq 0$ if $y_j \geq \delta w_j \geq \delta y_k w_k$
  - 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. $y'_j \geq 0$

Existence proof (non-constructive one)

FEASIBILITY OF $Y''$

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

FEASIBILITY OF $Y''$

- Finally, we show 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 weight($Y''$) = weight($Y$) $\leq M$
  - Therefore, $Y''$ is feasible!
WHAT IF ELEMENTS DON'T HAVE DISTINCT PROFIT/WEIGHT RATIOS?

OPTIMALITY PROOF WITHOUT DISTINCTNESS

- 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!)
To show \( Y' \) is feasible, we show
\[
\text{weight}(Y') \leq M \quad \text{and} \quad y_k' \geq 0, y_j' \leq 1
\]

**Modified optimal solution \( Y' \)**

We move \( \delta \) weight from item \( k \) to item \( j \)

This does not change the total weight!

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

**Profit of \( Y' \)**

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

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

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

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

Since \( Y \)'s optimal, this \textbf{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 \textbf{maximal} number of indices!

**Feasibility of \( Y' \)**

- Showing \( y_k' \geq 0 \)
  - By definition, \( y_k' = y_k - \frac{\delta}{w_k} \geq 0 \) if \( \delta \leq y_k w_k \)
  - 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 \( y_j' \leq 1 \)
  - \( y_j' = y_j + \frac{\delta}{w_j} \leq 1 \) if \( \delta \leq w_j (1 - y_j) \) \[\text{(rearranging)}\]
  - \( \delta \leq w_j (1 - y_j) \) \[\text{(definition of \( \delta \))}\]
  - and \( w_j (x_j - y_j) \leq w_j (1 - y_j) \) \[\text{(by feasibility of \( X \), i.e., \( x_j \leq 1 \))}\]

**Summarizing exchange arguments**

- If inputs are distinct
  - So there is a unique optimal solution
  - Let \( O \not= G \) 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 \( O \not= G \) 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

**Interval colouring**

- I don't think we will have time to get past here.
  - But if we do, great, we can catch up.
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 \( col: A \rightarrow \{1, \ldots, c\} \) 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

- 7 intervals, 6 colours, Feasible, but not optimal

Example

- 7 intervals, 2 colours, Optimal

Same colour, but disjoint? OK!

EXAMPLE: ORDER MATTERS!

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 \( (i+1) \)th 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?

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!
EXAMPLE: ORDER MATTERS!
EXAMPLE: ORDER MATTERS!

EXAMPLE: ORDER MATTERS!

EXAMPLE: ORDER MATTERS!

EXAMPLE: ORDER MATTERS!

EXAMPLE: ORDER MATTERS!

EXAMPLE: ORDER MATTERS!

Pre-sort intervals by increasing start time!

Used 4 colours

Can we do better?

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

A1 A2 A3 A4 A5 A6 A7 A8 A9 A10

x-axis

0 2 4 6 8 10 12 14 16 18 20

Can we do better?

Used 3 colours

Example: RUNNING GREEDY

Finish \( [c] \) = 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.

For each interval \( A_i \), search for an appropriate colour \( c \).

Interval 1 gets colour 1

Check if we can reuse any colour \( c \) in 1..d.

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

If we didn't reuse a colour, use a new one.

\( i = 1 \)

Code before the loop: just assign colour 1

\( d = 1 \)

\( \text{finish}[1] = \)
EXAMPLE: RUNNING GREEDY

\[ \begin{align*}
A_1 & \quad \text{colour 1} \\
A_2 & \quad \text{colour 2} \\
A_3 & \quad \text{colour 3} \\
A_4 & \quad \text{colour 4} \\
A_5 & \quad \text{colour 5} \\
A_6 & \quad \text{colour 6} \\
A_7 & \quad \text{colour 7} \\
A_8 & \quad \text{colour 8} \\
A_9 & \quad \text{colour 9} \\
A_{10} & \quad \text{colour 10}
\end{align*} \]

\( x \)-axis

0 2 4 6 8 10 12 14 16 18 20

Is \( f_1 \leq s \) for each \( i \)?

No. We cannot reuse colour 1.

\( i = 2 \)

While loop over \( c \).

Check if we can reuse a colour in 1..d.

\( d = 2 \)

Finish[1] = 85

Is \( f_2 \leq s \) for each \( i \)?

No. We cannot reuse colour 1.

\( i = 3 \)

While loop over \( c \).

Check if we can reuse a colour in 1..d.

\( d = 2 \)


Is \( f_2 \leq s \) for each \( i \)?

No. We cannot reuse colour 2.

Cannot reuse any colour. Create a new one!


Is \( f_3 \leq s \) for each \( i \)?

Yes. We can reuse colour 3.

\( i = 4 \)

While loop over \( c \).

Check if we can reuse a colour in 1..d.

\( d = 3 \)


Is \( f_4 \leq s \) for each \( i \)?

Yes. We can reuse colour 1.

\( i = 5 \)

While loop over \( c \).

Check if we can reuse a colour in 1..d.

\( d = 3 \)


Is \( f_5 \leq s \) for each \( i \)?

No. We cannot reuse colour 1.

\( i = 6 \)

While loop over \( c \).

Check if we can reuse a colour in 1..d.

\( d = 3 \)


Is \( f_6 \leq s \) for each \( i \)?

No. We cannot reuse colour 2.

Is \( f_6 \leq s \) for each \( i \)?

Yes. We can reuse colour 3.


\( i = 7 \)

While loop over \( c \).

Check if we can reuse a colour in 1..d.

\( d = 3 \)


Is \( f_7 \leq s \) for each \( i \)?

No. We cannot reuse colour 1.

\( i = 8 \)

While loop over \( c \).

Check if we can reuse a colour in 1..d.

\( d = 3 \)


Is \( f_8 \leq s \) for each \( i \)?

No. We cannot reuse colour 2.

Is \( f_8 \leq s \) for each \( i \)?

Yes. We can reuse colour 3.
We prove a more general statement
Let $F_c$ be the first interval that has colour $c$
Let $L_c$ be the last interval that has colour $c$ and starts before $F_c$ ends
We prove $F_c$ overlaps every interval $L_a$ for all $a < c$
Let's argue $F_c$ overlaps $L_a$
Note $L_a$ must exist (otherwise greedy would just use colour 1 for $F_c$)
And $\text{finish}[L_a] > \text{start}[F_c]$ in every case

Moreover, every interval in $(\text{start}[L_a], \text{finish}[L_a])$ contains the starting time of $F_c$
So, we must use $c$ colour

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 $F_c$ be the first interval that has the last colour $c$
Let $L_c$ be the last interval that has colour $c$ and starts before $F_c$ ends
We prove $F_c$ overlaps every interval $L_a$ for all $a < c$
$D = 1$ the proof is immediate, so suppose $D \geq 2$

TIME COMPLEXITY?

Total $O(n \log n + md)$
Could be $O(n \log n)$ if only a constant number colours are needed
Could be $O(n^2)$ if $D$ colours are needed
Most accurate complexity statement is $O(n \log n + md)$ where $D$ is # colours used
IMPROVING THIS ALGORITHM

- Current greedy algorithm:
  - For each interval \( A_i \), compare its start time \( s_i \) with the \( finish[c] \) times of all colours introduced so far
  - Why? Looking for some \( finish[c] \) 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 \( finish[c] \) at all times in the algorithm
- Then we only need to look at minimum element

EXAMPLE: HEAP-BASED ALGORITHM

- Initial state

Iteration 1:
- Check heap minimum
- Empty, so a new colour is needed

Iteration 2:
- Check heap minimum
- Finish at time 3
- No. New colour!

Iteration 3:
- Finish at time 7
EXAMPLE: HEAP-BASED ALGORITHM

Iteration 1:
- Check heap minimum: Null
- Check if finish time is before $s_1$: No

Iteration 2:
- Check heap minimum: Null
- Check if finish time is before $s_2$: No

Iteration 3:
- Check heap minimum: Null
- Check if finish time is before $s_3$: Yes

Iteration 4:
- Check heap minimum: Null
- Check if finish time is before $s_4$: Yes

Iteration 5:
- Check heap minimum: Null
- Check if finish time is before $s_5$: Yes

Iteration 6:
- Check heap minimum: Null
- Check if finish time is before $s_6$: Yes

Iteration 7:
- Check heap minimum: Null
- Check if finish time is before $s_7$: Yes

Iteration 8:
- Check heap minimum: Null
- Check if finish time is before $s_8$: Yes

Iteration 9:
- Check heap minimum: Null
- Check if finish time is before $s_9$: Yes

Iteration 10:
- Check heap minimum: Null
- Check if finish time is before $s_{10}$: Yes

No new colour!
EXAMPLE: HEAP-BASED ALGORITHM

Iteration: 3
Check heap minimum: A1
Check if finish time 5 is before s5: Yes
Reuse colour, deleteMin and insert new finish time into heap!

Iteration: 4
Check heap minimum: A2
Check if finish time 7 is before s6: Yes
Reuse colour, deleteMin and insert new finish time into heap!

Iteration: 5
Check heap minimum: A3
Check if finish time 7 is before s6: Yes
Reuse colour, deleteMin and insert new finish time into heap!

Total Θ(n log n) + Θ(n log D) since n ≥ D, Θ(n log n)

And so on, and so forth...