**Example**: \( W=5 \), \( n=4 \) items

<table>
<thead>
<tr>
<th>Item</th>
<th>Weight</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>40</td>
</tr>
</tbody>
</table>

**Solution**: items 2, 4 value = 90.

Start by finding max value (instead of items).

Express solution to a subproblem in terms of subsolutions.

*What is a subproblem? Imagine bars of gold are lined up. Entire problem - make decisions for all \( n \) bars.*

*Sub-problem - make decisions for first, say, \( i \) bars.*

**A[i]**: max value for first \( i \) items (with weight limit \( W \)).

**What does \( A[i] \) depend on?** \( A[i-1] \).

**A[i-1]** - max value for first \( i-1 \) items (with weight limit \( W \)).

**Problem?** If we decide to take item \( i \), then the weight limit for previous \( i-1 \) items is not \( W \).

(Weight limit is \( W - W_i \).)

**Example**: \( \{1, 1, ..., 1\} \)

\( W=100 \), if we take \( i \), items

\( i=5 \) lbs \( 1, ..., i-1 \) can weigh at most 95.
Therefore, d.p. formulation depends on 2 factors:

1) # items  
2) weight limit.

\[ A[i, j] = \text{max value from first } i \text{ items with weight limit } j. \]

**D.P. Formulation:**

For each item \( i \), we have a choice: take \( i \) or leave \( i \).

- leave \( i \): \[ A[i, j] = A[i-1, j] \]

  If we leave item \( i \), we have weight limit \( j \) for previous \( i-1 \) items.

Where in the matrix can we find the best value we can get for the first \( i-1 \) items and weight limit \( j \)?
take $i$. $A[i,j] = v_i + \min A[i, j-w_i]$ Note

If weight limit is $j$ and we take item $i$, previous $i-1$ items can weigh at most $j-w_i$.

Where in $A$ can we find best value for first $i-1$ items and weight limit $j-w_i$?

Example

Suppose deciding on $5^{th}$ item ($i=5$), weight limit is 15 ($j=15$).

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

(5 lbs)

Already made a decision for each of these, got some total value $\$100$.

If we leave $i$, value is still $\$100$.

If we take $i$, total weight of all items must be 15. So $i-1$ items can weigh at most 10.
One more case: what if weight of current item exceeds current weight limit? (ex: 5th item weighs 25 lbs?)

⇒ must leave item i!

\[
A[i,j] = \begin{cases} 
0 & \text{if } i=0 \text{ or } j=0 \\
A[i-1,j] \text{ (leave)} & \text{if } i,j>0 \text{ and } w_i > j \quad \text{(A)} \\
\max \left\{ \begin{array}{l}
A[i-1,j] \text{ (leave)} \\
\max \left\{ \begin{array}{l}
A[i-1,j-w_i] \text{ (take)} \quad \text{if } i,j>0, \\
v_i + A[i-1,j-w_i] \text{ (take)} \quad \text{if } w_i \leq j
\end{array} \right. 
\end{array} \right. 
\end{cases}
\]

(1) Base Case?
(2) How to fill A? \( A[0][0..W] = 0 \) \( A[0..n][0] = 0 \)

Then nested for loop: for \( i=1..n \), for \( j=1..W \)

(3) Where is value (max value) in \( A[n][W] \)

(4) Run Time? \( O(n \cdot W) \)

(5) Actual solution (optimal items)? <Later>
- Matrix $A$ of size $n \times W$
- nested for loop to fill matrix
- Optimal value at $A_{[n][W]}$
- Run Time: $O(nW) \leq$ Pseudopolynomial $W$ may be $2^n$

**Example:**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-10</td>
<td>-10</td>
<td>-10</td>
<td>-10</td>
<td>-10</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-10</td>
<td>-10</td>
<td>-10</td>
<td>-10</td>
<td>-10</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>-10</td>
<td>-10</td>
<td>-10</td>
<td>-10</td>
<td>-10</td>
</tr>
</tbody>
</table>

**Weights:**
- (too heavy) $A$
- (leave value, take value) $B$ $\Box =$ maximum

Optimal value: 90

Notice: optimal substructure ex. $A_{[3][47]}$ is optimal value for first 3 items w/ weight limit $j$.

How to find these items?
1. Create empty list I of size n
2. Set \( i = n, j = W \) //Start at optimal value
3. If \( A[i, j] \) was "leave"
4. //Go to \( A[i-1, j] \)
5. \( i = i - 1 \)
6. Else // \( A[i, j] \) was "take"
7. Add i to I.
8. //Go to \( A[i-1, j - w_i] \)
9. (Set) \( i = i - 1, j = j - w_i \)
10. Repeat from (3) while \( i \) and \( j \) > 0

\[
\begin{array}{c|c|c|c}
\hline
i & j & 4 & 5 \\
\hline
4 & take & 4 & I: [4, 2] \\
2 & leave & 3 & \\
2 & take & 2 & \\
1 & 0 & \\
\hline
\end{array}
\]

Notice: Incorrect to just find all \( i \)'s s.t. \( A[i, j] > A[i-1, j - w_i] \)

\( 80 > 50 \)

But not optimal to take item 3!

Note about Runtime: O(nW) but W may be \( O(2^n) \)
so not really polynomial, instead pseudo-polynomial,
Fractional Knapsack

Consider variation of 01-Knapsack.

No bars of gold, now bags of gold dust

\[ \begin{align*}
&\text{Bag 1: } & \text{value } & \text{weight } \\
&\text{Bag 2: } & 100 & 20 \\
&\text{Bag 3: } & 120 & 30 \\
&\text{Bag 4: } & 60 & 10
\end{align*} \]

How much of each bag to take to maximize total value while not exceeding weight limit?

\[ \begin{align*}
\text{total weight } & \leq W \\
\text{total value } & \geq \text{value}
\end{align*} \]

n bags 1...i...n of gold dust. Bag i has gold type i.
Each bag has total weight \( w_i \) and total value \( v_i \).

Knapsack has weight capacity \( W \).

Now, can take a fraction \( x_i \) of bag i (with gold type i): where \( 0 \leq x_i \leq 1 \)

\[ \Rightarrow \text{Gold type } i \text{ contributes weight } w_i \cdot x_i \]
\[ \Rightarrow \text{value } v_i \cdot x_i \]

Goal: Find \( x_i \) for \( 1 \leq i \leq n \) that maximizes

\[ \sum_{i=1}^{n} v_i x_i \quad \text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq W \]

Fractional Knapsack applications <on slides>
(Solution?)

Since dealing with fractions, Brute Force would yield an infinite number of possible solutions.

(Greedy?)
1. Sort bags in decreasing order of \( \frac{v_i}{w_i} \) "worth"
2. Continuously take as much as possible of type with highest worth until \( W \) reached.

ex: \( W = 50 \)

\[
\begin{array}{ccc}
10 \text{ lbs} & 20 \text{ lbs} & 30 \text{ lbs} \\
\$60 & \$100 & \$120 \\
\end{array}
\]

\[
\begin{array}{c}
\frac{v_i}{w_i}
\end{array}
\]

\[
\begin{array}{c}
6 \\
5 \\
4 \\
\end{array}
\]

\[
\begin{array}{c}
\frac{20}{30} \Rightarrow 80 \\
\frac{20}{20} \Rightarrow 100 \\
\frac{10}{10} \Rightarrow 60 \\
\end{array}
\]

\[
\begin{array}{c}
\$240 \\
\end{array}
\]

Note: didn't work for 01-KS ble by choosing the item with highest \( \frac{v_i}{w_i} \) we may end up with empty space in the Ks (no other items fit in that space). Possible that another set of items with lower \( \frac{v_i}{w_i} \) uses up the space more efficiently so we fill the Ks up more + get higher total value. For frac-KS we don't have this problem ble we can take as much of an item as will fit.
Fractional Knapsack (T, W) // T is set of n gold types/bags // W is knapsack weight limit
1. for i = 1 to n
   \[ G_i = \frac{v_i}{w_i} \] // "worth" of type i
2. Sort G in descending order
3. \( m = 0 \) // weight to take of current type
4. \( c = 0 \) // current weight of k.s.
5. while (\( c < W \))
6. \{ get next type, \( i \), from G \} // see example!
7. \* if \( w_i + c \leq W \) // taking all of \( i \) does not exceed \( W \)
   \[ m = w_i \] // take all
8. else
9. \[ m = W - c \] // take as much that fits
10. \[ x_i = \frac{m}{w_i} \]
11. \[ c = c + m \]

More concisely:
\[
\text{if: } w_i \leq W - c \Rightarrow m = w_i \quad \text{or} \quad m = \min(w_i, W - c)
\text{else: } w_i > W - c \Rightarrow m = W - c
\]

Run-Time: \( n \) types
Sort G: \( O(n\log n) \)
while loop: \( O(n) \) (Step 6 - get next type, \( n \) types)
\( = O(n) \) only if \( c \) (and \( m \)) increases by 1 each time, so
all \( w_i = 1 \) \( \implies \) Time: \( O(\min(n, W)) \)
Big Picture:

<table>
<thead>
<tr>
<th>Divide and Conquer</th>
<th>Dyn. Prog.</th>
<th>Greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MergeSort</td>
<td>Change-Making (non-U.S)</td>
<td>Change-Making (U.S)</td>
</tr>
<tr>
<td>Selection</td>
<td>Assem. Line</td>
<td>Frac-KS</td>
</tr>
<tr>
<td>Closest Points</td>
<td>LCS</td>
<td>01-KS</td>
</tr>
</tbody>
</table>

How to know if a greedy algorithm is optimal?

Prove it! (Many ways, we will discuss one)

Use something like an *Inductive Proof*.

Show 2 properties:

1. Greedy Choice Property - some optimal solution contains the first choice made by greedy
   => "first choice is optimal" \(\sim\) Base Case

2. Optimal Substructure property - optimal solution to entire problem contains optimal solutions to subproblems
   => "If some sub-solution is optimal, the next sub-solution is optimal" \(\sim\) Inductive Step