Knapsack problem:

Backpack with total capacity $B$
& $n$ objects with:
- integer weights $w_1, \ldots, w_n$
- integer values $v_1, \ldots, v_n$

Goal: Find subset $S$ of objects that:
  a) fits in the backpack
  & b) maximizes the total value.

In other words, find $S \subseteq \{1, \ldots, n\}$ where:
  a) $\sum_{i \in S} w_i \leq B$
  & b) maximizes $\sum_{i \in S} v_i$

Application: scheduling jobs.

Two versions:
1) Without repetition: one copy of each object
2) With repeats: unlimited supply of each object.
Version 1: one copy of each object.

What about greedy approach?

Example: 4 objects: 1, 2, 3, 4

Values: 15, 10, 8, 1

Weights: 15, 12, 10, 5

Total capacity $B = 22$

Greedy: take most valuable/per unit of weight

Sort by $r_i = \frac{v_i}{w_i}$. Note, $r_1 > r_2 > r_3 > r_4$.

Greedy solution: objects 1 & 4

Total value = 16.

Optimal solution: objects 2 & 3

Total value = 18.
Dynamic Programming approach:

Step 1: Define the subproblem in words.
initial attempt: Prefix of input.

Let \( K(j) = \text{max value achievable using a subset of objects } 1, \ldots, j \).

Step 2: Express \( K(j) \) in terms of \( K(1), \ldots, K(j-1) \).

Consider \( K(j) \): two options — use object \( j \) or not.
If don't use object \( j \), then \( K(j) = K(j-1) \).

If use object \( j \), then can we add \( j \) to the optimal for \( K(j-1) \)?

Need to know how much weight is available?
if include \( j \) then want optimal solution with weight \( \leq B - w_j \).

So need to keep track of weight available.
Attempt 2 -

Step 1: Subproblem definition

For $b \& j$ where $0 \leq b \leq B \& 0 \leq j \leq n$,

let $K(b, j) =$ max value achievable using a subset of objects 1, ..., $j$ & total weight $\leq b$.

Goal: compute $K(B, n)$.

Step 2: Recurrence relation -

For $K(b, j)$:

either:

- Use object $j$ then want optimal solution for subset of objects 1, ..., $j-1$ with total weight $\leq b - w_j$
  
  then $K(b, j) = K(b - w_j, j - 1) + v_j$

- Don't use object $j$
  
  then $K(b, j) = K(b, j - 1)$
Therefore,

if \(w_j \leq b\),
\[
K(b, j) = \max_j y_j + K(b - w_j - 1, j)
\]

if \(w_j > b\),
\[
K(b, j) = K(b, j - 1).
\]

Base cases:
\[
K(b, 0) = 0 \\
K(0, j) = 0
\]

Recurrence for \(K(b, j)\) uses \(K(?, j - 1)\)
So fill table from \(j = 0 \rightarrow n\)
Knapsack \( N \), Repeat \((B, w_1, \ldots, w_n, \nu_1, \ldots, \nu_n)\):

For \( j = 0 \rightarrow n \): \( K(0, j) = 0 \)

For \( b = 0 \rightarrow B \): \( K(b, 0) = 0 \)

For \( j = 1 \rightarrow n \)

For \( b = 1 \rightarrow B \)

if \( w_j > b \),

then \( K(b, j) = K(b, j-1) \).

else \( K(b, j) = \max \{ y_j + K(b-w_j, j-1), \ \ K(b, j-1) \} \).

Return \( K(B, n) \)

\( \underline{Running \ time:} \)

outer for loop of size \( O(n) \)

\( \times \) inner loop of size \( O(B) \)

\( \Rightarrow O(nB) \) time.
Version 2: unlimited supply of each object

Try same subproblem as before:

\[ K(b,j) = \text{max value achievable using subset of objects } 1, \ldots, j \text{ (Possibly with repeats) and total weight } \leq b. \]

if \( w_j \leq b \):

either:

- don't include \( j \) so \( K(b,j) = K(b, j-1) \).
- or include \( j \), but how many times?

adding one more copy then:

\[ K(b, j) = v_j + K(b-w_j, j) \]

\( j \) instead of \( j-1 \)

So can use \( j \) again.
Therefore,
if \( w_j \leq b \)
\[
K(b, j) = \max \{ K(b, j-1), y_j + K(b - w_j, j) \}
\]
if \( w_j > b \)
\[
K(b, j) = K(b, j-1).
\]

**Knapsack Repeat** \((B, w_0, \ldots, w_n, y_1, \ldots, y_n)\):

For \( j = 0 \rightarrow n \), \( K(0, j) = 0 \)
For \( b = 0 \rightarrow B \), \( K(b, 0) = 0 \)
For \( j = 1 \rightarrow n \),
For \( b = 1 \rightarrow B \),
if \( w_j > b \)
\[
\text{then } K(b, j) = K(b, j-1)
\]
else \( K(b, j) = \max \{ K(b, j-1), y_j + K(b - w_j, j-1) \} \)

Return \( K(B, n) \).
Filling the table:

\[ K = \begin{array}{c}
\text{uses} \\
K(b,j-1) \\
\text{or} \\
K(b-w_j,j) \\
\text{earlier column} \\
\text{same column last earlier}
\end{array} \]

Running time: \( O(nB) \) as before.

Alternative DP for version with repeats:
- Don't need to keep track of which objects considered.

1. Let \( K(b) = \text{max value achievable using total weight } \leq b \) & all objects 1, ..., n allowed.

2. Recurrence: try all possibilities for "last object added."
   - So try all \( l \) where \( 1 \leq l \leq n \) & \( w_l \leq b \).

Hence, \( K(b) = \max \{ K(b-w_l)+v_l : 1 \leq l \leq n, w_l \leq b \} \)

\( K \) is one-dimensional, but each entry takes \( O(1) \) time to fill in so \( O(nB) \) time.
Our algorithm had running time $O(nB)$.

Let $V = \sum_{i=1}^{A} V_i = \text{total value of all objects}.$

Consider the version without repeats then the max value is $\leq V$.

Exercise (hard): Design DP algorithm with running time $O(nV)$.

Also, discussion about the fact that knapsack (we'll see later in the course) is NP-hard.