Knapsack Problem:

Total capacity $B$
and $n$ objects with:

- integer weights $w_1, \ldots, w_n$
- integer values $v_1, \ldots, v_n$

Goal: Find subset $S$ of objects that:

1) fits in the backpack
2) maximizes the total value

In other words, find $S \subseteq \{1, \ldots, n\}$ where:

1) $\sum_{i \in S} w_i \leq B$
2) maximizes $\sum_{i \in S} v_i$

Application: scheduling jobs.
Versions:
1) without repetition: one copy of each object
2) with repetition: unlimited supply of each object

First: 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

\[ B = 22 \]

Greedy: sort by \( r_i = \frac{v_i}{w_i} \) so \( r_1 > r_2 > r_3 > r_4 \)

Greedy solution: objects 1 & 4 for total value = 16

Optimal solution: objects 2 & 3 for total value = 18.
Dynamic Programming approach:

First, define the subproblem
Initial attempt: Prefix

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

Second step: Express \( K(j) \) in terms of \( K(1), \ldots, K(j-1) \).

Consider \( K(j) \): given \( K(j-1) \), can we add object \( j \) to the optimal for \( K(j-1) \)?

Need to know how much weight is available.

Want best solution for subset of 1 \ldots j-1 with weight \( \leq B \)
& weight \( \leq B-w_j \)

Next round need \( B, B-w_j-1, B-w_j, B-w_j-w_j-1 \)
So need to consider all possible weights.
Subproblem definition:
For $b \leq j$ where $0 \leq b \leq B$ & $0 \leq j \leq n$, let $K(b, j) =$ max value achievable using a subset of objects $1 \ldots j$ & total weight $\leq b$.

Goal: compute $K(B, n)$

Recurrence relation:
For $K(b, j)$:
- either use object $j$:
  then want best solution for subset of $1 \ldots j-1$ with total weight $\leq b - w_j$ (so $j$ fits in)
- or don't use object $j$:
  then want best for $1 \ldots j-1$ with total weight $\leq b$. 


Therefore,

if $w_j \leq b$,

$$k(b_{j,j}) = \max \{ y_j + k(b-w_j, j-1), k(b_{j-1}, j) \}$$

if $w_j > b$,

$$k(b_{j,j}) = k(b_{j-1}, j)$$

Base cases:

$k(b, 0) = 0$

$k(0, j) = 0$

Recurrence for $k(b, j)$ uses $k(?, j-1)$

So fill table from $j = 0 \rightarrow j = n$.

```
   0 1 2 3 ...
0   k
1   k
2   k
3   k
```

fill column by column
Knapsack NoRepeat \((B, w_1, \ldots, w_n, v_1, \ldots, v_n)\):

For \(j = 0 \rightarrow n\), \(k(0, j) = 0\)

For \(b = 0 \rightarrow B\), \(k(b, 0) = 0\)

For \(j = 1 \rightarrow n\),
   For \(b = 1 \rightarrow B\),
      \[\text{if } w_j > b, \quad k(b, j) = k(b, j-1)\]
      \[\text{else } k(b, j) = \max(v_j + k(b-w_j, j-1), k(b, j))\]

Return \((k(B, n))\)

**Running time:**

For loop of size \(O(n)\) & inner loop of size \(O(B)\), thus total running time of \(O(nB)\).
Now: unlimited supply of each object

Try subproblem as before:

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

if \( w_j \leq b \)

either include object \( j \) but then

need to allow more copies of it so:

\[ V_j + k(b - w_j, j) \]

\[ k_j \text{ instead of } k_{j-1} \]

or don't include \( j \) so:

\[ k(b, j-1) \]
Therefore,
if \( w_j \leq b \),
\[
   k(b_j) = \max \{ k(b_j - 1), V_j + k(b - w_j, j) \}
\]
if \( w_j > b \),
\[
   k(b_j) = k(b_j - 1).
\]

Knapsack Repeat \((B, w_1, \ldots, w_n, v_1, \ldots, v_n)\):
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 \( k(b, j) = \max \{ V_j + k(b - w_j, j), k(b, j - 1) \} \)

Return \((k(B, n))\)
Filling the table:

\[ k(b, j) \text{ uses } k(b, j - 1) \text{ (previous column)} \]

or \( k(b, j) = k(b - w, j) \) (earlier in same column)

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

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

Recurrence:

Try all possibilities for last object \( l \) need that \( w_l \leq b \)

Hence:

\[ K(b) = \max \sum_{l} K(b-w_l) + w_l : 1 \leq l \leq n, w_l \leq b \]

\( K \) is one-dimensional but each entry takes \( O(n) \) time so \( O(nB) \) total time as before.