To Create D. P. (dynamic programming) algorithm:

1. Think of form of optimal solution.
   - WMIS on line: \( V_n \in S \) or \( n \notin S_n \)

2. Optimal soln in terms of smaller soln? (Multiple cases)
   (i) \[ \text{MWIS} \]
   (ii) \[ \text{MWIS} + \]

3. Create recurrence relating value of optimal solution to smaller solutions.
   \[ A(k) = \max \{ A[k-1], A[k-2] + W_k \} \]

4. Store values in an array using for-loop

5. Work backwards through array to reconstruct optimal solution
Dynamic Programming - More Practice

Knapsack Problem \( K(v_1, \ldots, v_n, w_1, \ldots, w_n, W) \)

Input: \( n \) items, each has
- value \( v_i \)
- size \( w_i \)

- Capacity \( W \)

Output: A subset \( S \subseteq \{1, 2, \ldots, n\} \) that maximizes \( \sum_{i \in S} v_i = V(S) \) \( \uparrow \) objective function

and satisfies \( \sum_{i \in S} w_i \leq W \) constraints

Applications
- Cargo trucks
- Investments

\[
\begin{align*}
V_i &= \text{expected yield} \% \\
W_i &= \text{min investment amount} \\
W &= \text{amount to invest}
\end{align*}
\]
Some notation

• Let \( K_{i,r} \) be subproblem on 1st \( i \) items, with capacity \( R \).
• Solution to \( K_{i,r} \) is a set \( S \subseteq \{1,2,\ldots,i\} : W(S) \leq R \).
• Optimal solution is solution with largest \( V(S) \).

1. Let \( S \) be the optimal solution \( K_{n,w} \),

Only two possibilities \( \{ (i) \; n \in S \rightarrow \quad \text{[2]} \quad \text{How to relate to optimal solution of smaller problem?} \}

(ii) \( n \notin S \rightarrow \)

(i) If \( S \) is optimal soln to \( K_{n,w} \) and \( n \notin S \), then \( S \) is optimal soln to \( K_{n-1,w} \).

\[ \text{Pf: For contradiction, suppose } S \text{ is not opt soln for } K_{n-1,w} \text{ but } S \text{ is optimal soln to } K_{n,w} \text{ and } n \notin S. \text{ Note } S \text{ is a solution to } K_{n-1,w} \text{ but since it is not optimal, } \exists \text{ a soln } S' \text{ for } K_{n-1,w} \text{ such that } V(S') > V(S). \text{ But } S' \text{ is also a solution to } K_{n,w}, \text{ and since } V(S') > V(S), \text{ } S \text{ is not optimal for } K_{n,w}, \text{ a contradiction.} \]

(Recall, if trying to prove \( P \Rightarrow Q \) by contradiction, assume \( P \land \neg Q \) then try to show \( \neg P \).)
(ii) If $S$ is opt. soln for $K_{n,w}$ and $v \notin S$, then $S - \{v\}$ is opt. soln for $K_{n-1,w-w_v}$.

Proof: Suppose for contradiction $S - \{v\}$ is not opt., but $S$ is opt. for $K_{n,w}$ and $v \notin S$. Then $\exists S'$, a soln for $K_{n-1,w-w_v}$ s.t. $V(S') > V(S - \{v\})$. But then $S + \{v\}$ is a soln to $K_{n,w}$, and $V(S + \{v\}) > V(S)$, so $S$ is not opt. for $K_{n,w}$, a contradiction.

Consequence: Let $V[i,r]$ be value of optimal soln to $K_{i,r}$.

If $S$ is opt. for $K_{i,r}$ and $i \notin S$, then $V[i-1,r] = V[i,r]$.

If $S$ is opt. for $K_{i,r}$ and $i \in S$, then $V[i-1,r-w_i] = V[i,r] - V_i$.
3. Recurrence Relation

Base Case:\n\[
V[0, r] = 0
\]
\[
V[i, 0] = 0
\]

\[
V[i, r] = \max \{ V[i-1, r], V[i-1, r-w_i] + v_i \}
\]

4. Create a for-loop to fill out. (Include base case).

for \( r = 0 \) to \( W \):

\[
V[0, r] = 0
\]

for \( i = 1 \) to \( n \):

\[
V[i, 0] = 0
\]

for \( i = 1 \) to \( n \):

for \( r = 1 \) to \( W \):

\[
V[i, r] = \max \{ V[i-1, r], V[i-1, r-w_i] + v_i \}
\]
Pf of correctness: What are we trying to prove?

Let $V_{i,r}$ be optimal value for $K_{i,r}$.

Loop Invariant: $V[i',r'] = V_{i',r'}$ for all $0 < i, 0 < r' \leq r$.

Initialization: $i=1 \quad r=1 \quad \checkmark$

Maintenance: Since $i-1 < i$, $V[i-1,r] = V_{i-1,r}$

Now one of (i) or (ii) must be true, so using previous proof, either $V[i,r] = V[i-1,r]$ or $V[i,r] = V[i-1,r-w_i] + v_i$

but want optimal, so choose larger.

Termination: All $V[i,r] = V_{i,r}$ for $1 \leq i \leq n, 1 \leq r \leq W$
5. Write pseudocode to get $S$ using array $V$

$S' = \emptyset$

$i = n$

$r = W$

while $i > 0$:

if $V[i, r] = V[i-1, r-w_i] + v_i$

$S' = S' + i$

$r = r - w_i$

$i = i - 1$
Pf that loop is correct. Let $S$ be opt. solution. Can assume $V[i, w]$ is value of opt. sol. on $K_i, w$.

Loop Invariant: $S'$ contains all elements of $S$ larger than $i$.

$r = W - W(S')$

Initially: $i = n, S = \emptyset, r = W$

Maintenance: Suppose loop invariant is true going into loop. Then $S'$ contains elements of $S$ larger than $i$, so we just need to figure out those less than or equal to $i$. Since we've already used up capacity $W(S')$, we need to solve $K_i, W - W(S') = K_i, r$, to figure out remaining elements.

We now determine if $i \in \text{opt. soln for } K_i, r$.

Only two options (i), or (ii) and at least one is true, so either $V[i, r] = V[i-1, r]$ or $V[i, r] = V[i-1, r - W_i] + V_i$, and which one it is tells us whether to add $i$ or not. In either case invariant is maintained.