Goals

- Create an algorithm for Knapsack
- Prove correctness of dynamic programming alg.
- Describe why prefix-free codes are good.

Knapsack Problem

Input: n items, each has
- value \( v_j \in \mathbb{Z}^+ \)
- size \( w_j \)

- Capacity \( W \)

Output: A subset \( S \subseteq \{1, 2, \ldots, n\} \) that maximizes \( \sum_{i \in S} v_i = V(S) \)
and satisfies \( W(S) = \sum_{i \in S} w_i \leq W \) under constraints

\( K_{i,r} \) = optimal subset if only 1st \( i \) items are allowed, capacity \( r \).

1. Form of optimal solution \( S \) of \( K_{n,W} \)
   (i) \( n \notin S \)
   (ii) \( n \in S \)

2. Optimal soln in terms of smaller soln
   (i) If \( n \notin S \), \( S \) is optimal for \( K_{n-1,W} \)
   (ii) If \( n \in S \), \( S - \{n\} \) is optimal for \( K_{n-1,W-w} \)
   (Proof by contradiction)
3. Recurrence Relation for $V_{i,r} = \text{value of objective function of } K_{i,r}$

4. Write code to store $V_{i,r}$ in array

5. Write code to work backwards through array to recover $K_{n,w}$
3 Recurrence Relation for objective function

Base Cases: \( V_{0,r} = 0 \quad \forall r \in \{0, \ldots, W\} \)

\[
V_{i,r} = \max \left\{ V_{i-1,r}, V_{i-1,r-w_i} + v_i \right\}
\]

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

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

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

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

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

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

This option not allowed if \( r-w_i < 0 \)
Write pseudocode to get $S$ using array $V$

$S = \emptyset$

$i = n$

$r = W$

while $i > 0$:

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

$S = S + i$

$r = r - w_i$

$i --$

Ex: $W = 5$

$v_1 = 6 \quad w_1 = 2$
$v_2 = 5 \quad w_2 = 1$
$v_3 = 8 \quad w_3 = 3$
$v_4 = 7 \quad w_4 = 4$

Q: What is the runtime of DP knapsack algorithm?

A) $O(n)$  B) $O(n)$  C) $O(n + W)$  D) $O(nW)$