KNAPSACK

Learning Goals

- Describe Knapsack problem ✓
- Develop recurrence for Knapsack ✓
- Create dynamic programming alg
- Analyze runtime

Objects

Announcements

Programming, Ethics Assessment

Exit Tickets

n, W vs i, C

Knapsack Problem

Input: n items
- $v_i \in \mathbb{N}$ is value of i^{th} item
- $w_i \in \mathbb{N}$ is weight of i^{th} item

Capacity: W (max weight) $\in \mathbb{N}$

Output: $S \subseteq [n] = \{1, 2, 3, \ldots, n\}$

s.t.
- $V(S) = \sum_{i \in S} v_i$ is maximized
- $W(S) = \sum_{i \in S} w_i \leq W$

Applications
- Shipping
- Exam

Brute Force

Similar to MWIS

$O(n2^n)$

ex: $W = 5$

	1	2	3	4
value	8	5	6	7
weight	2	1	3	4

What is optimal set?

A) $\{1, 3\}$

B) ~~$\{1, 1\}$~~ No repeats

C) ~~$\{1, 2, 3\}$~~ $w > 5$

D $\{2, 3\}$ valid but low value

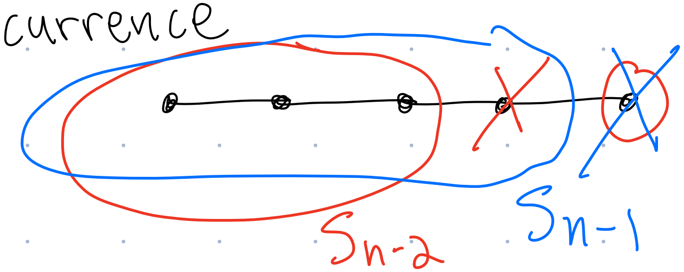
Dynamic Programming

1. Think about solution as sequence of choices. Final choice?

   MWIS: is each vertex in set or not → $v_n \in S$ or $v_n \notin S$

   ⓐ ⓑ

2. For each option, think about relevant subproblem + create recurrence



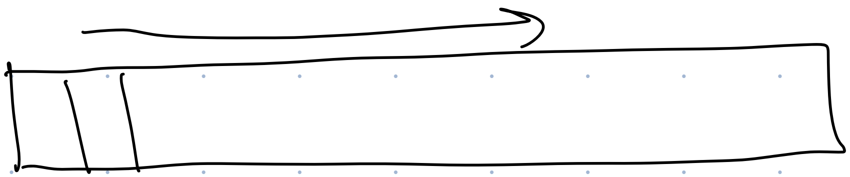
$$S_n = \begin{cases} S_{n-2} \cup \{v_n\} & \text{if } v_n \in S \\ S_{n-1} & \text{if } v_n \notin S \end{cases}$$

3. Convert into recurrence for objective function

$$W(S_n) \rightarrow A_n = \max\{A_{n-1},\ A_{n-2} + w(v_n)\},\ \text{B.C.}$$

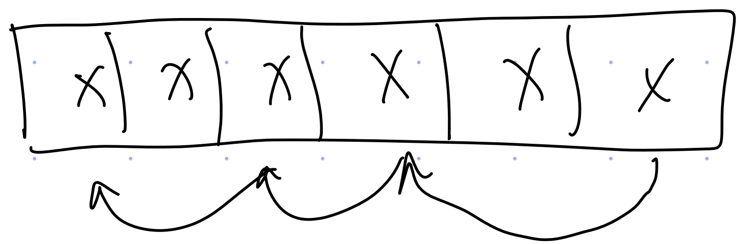
4. Pseudocode:

A



Base cases

Work backwards



Knapsack Problem

Input: n items

- v_i is value of i^{th} item
- w_i is weight of i^{th} item

Capacity W (max weight)

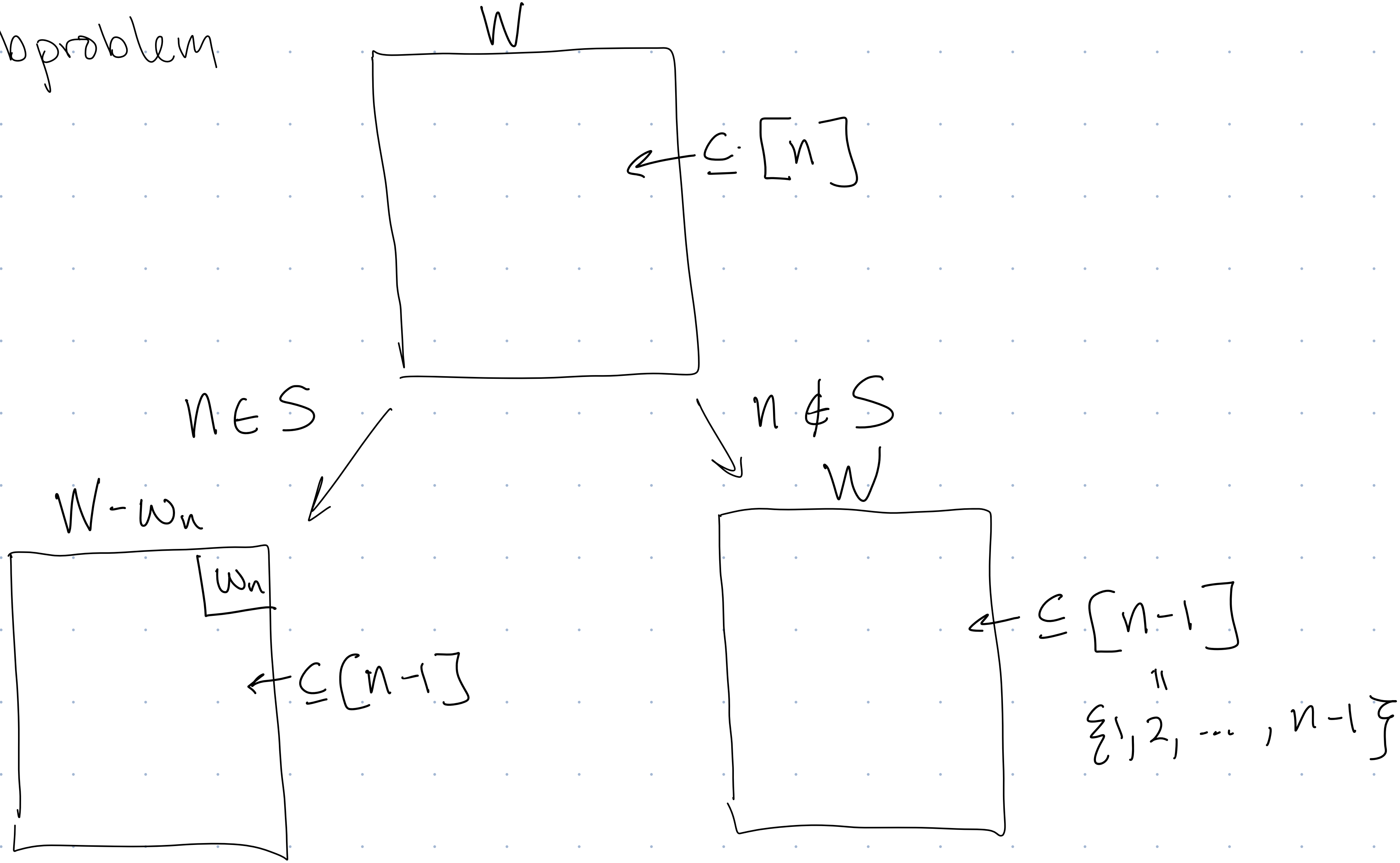
Output: $S \subseteq [n]$ ← notation $\{1, 2, 3, \ldots, n\}$

s.t.
- $V(S) = \sum_{i \in S} v_i$ is maximized
- $W(S) = \sum_{i \in S} w_i \leq W$

1. Ideas for final option?

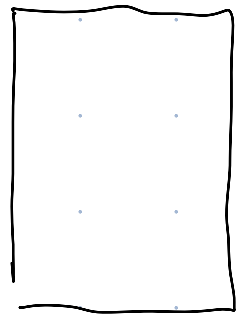
$n \in S$ or $n \notin S$

2a Subproblem



Define:

$$S_{i,c} = \left\{ \begin{array}{l} S \subseteq [i] \\ \cdot\ W(S) \leq C \\ \cdot\ V(S) \text{ is maximized} \end{array} \right.$$



	1	2	3	4
value	6	5	8	7
weight	2	1	3	4

What is $S_{2,4}$?

$S_{2,4} \subseteq [2] = \{1, 2\}$

A) $\cancel{\{1, 1\}}$

No repeats

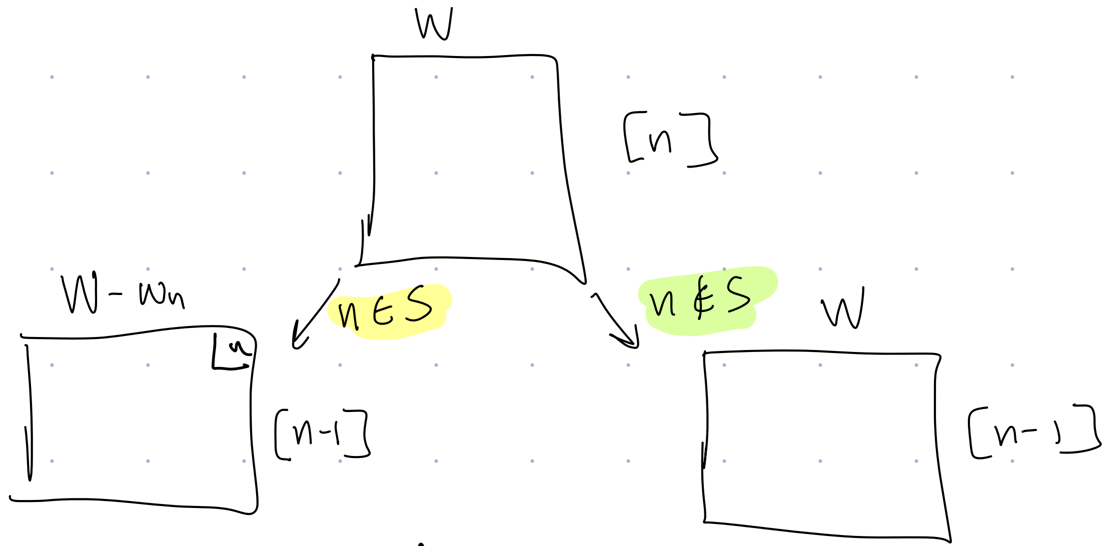
B) $\{1, 2\}$ ✓

C) $\cancel{\{2, 3\}}$

D) $\cancel{\{4\}}$

$\notin [2]$

2b



$$S_{n,W} = \begin{cases} S_{n-1,\, W-w_n} \cup \{n\} & \text{if } n \in S \\ S_{n-1,\, W} & \text{if } n \notin S \end{cases}$$

$$V(S_{i,c}) = A[i,c]$$

$$V(S_{n,W}) = \begin{cases} \max\{ V(S_{n-1,\, W-w_n}) + v_n ,\ V(S_{n-1,\, W}) \} \\ \text{Base cases} \end{cases}$$

$$\downarrow$$

$$\qquad\qquad\qquad\qquad i \in S \qquad\qquad\qquad\qquad\qquad i \notin S$$

$$A[i,c] = \begin{cases} \max\{ A[i-1,\, c-w_i] + v_i ,\ A[i-1,\, c] \} \\ \text{Base cases} \end{cases}$$

1. Fill out this array using recurrence for these items:

Input:

	1	2	3
value	6	5	8
weight	2	1	3

W = 5   n = 3

A

capacity (weight) c

i		0	1	2	3	4	5 = W
∅	← 0	0	0	0	0	0	0
{1}	← 1	0	0	6	6	6	6
{1,2}	← 2	0	5	6	11	11	11
{1,2,3}	← 3 = n	0	5	6	11	13	14

1 ∈ S

2 ∉ S

3 ∈ S

$S = \{1, 3\}$

$A[i, c] = V(S_{i,c})$

i=3

3 ∈ S   3 ∉ S

i ∈ S   i ∉ S

$$A[i, c] = \begin{cases} \max\{ A[i-1, c-w_i] + v_i ,\ A[i-1, c] \} \\ \boxed{} \end{cases}$$

$A[0,0] = 0$, $\forall c$, $A[0, c] = 0$

a) Determine Base Case(s)

b) Something else needed to properly use recurrence...

2. Write pseudocode to create A

Input: length-n arrays v and w, integer W

Output: Set $S \subseteq [n]$

For $c \leftarrow 0$ to W:

  $A[0, c] \leftarrow 0$ // Base Case

For $i \leftarrow 1$ to n:

  For $c \leftarrow 0$ to W: $i \notin S$

    $A[i, c] \leftarrow A[i-1, c]$

    If $c \geq w[i]$ and $A[i-1, c-w[i]] + v[i] > A[i-1, c]$

      $A[i, c] \leftarrow A[i-1, c-w[i]] + v[i]$

$i \in S$

Prog. language should not check 2nd clause if 1st is false

$S \leftarrow \boxed{\emptyset}$

$i \leftarrow \boxed{n}$

$c \leftarrow \boxed{W}$

While $i \geq 1$:

  if $A[i, c] = A[i-1, c]$: i--

  else:

    $S \leftarrow S \cup \{i\}$

    $c \leftarrow c - w[i]$

    i--