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) $\in \mathbb{N}$

Applications

- Shipping
- Exams (?)

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

s.t.

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

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?

$\boxed{\text{A) } \{1, 3\}}$ B) $\{1, 1\}$ C) $\{1, 2, 3\}$ D $\{2, 3\}$

Can't repeat would need to add another copy

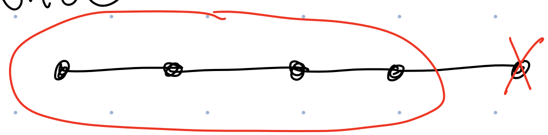
Dynamic Programming

1. Think about "final options" of a solution (solution as sequence of choices)

   MWIS: final vertex in set or not

2. (a) For each option, think about relevant subproblem + (b) create recurrence

   S_i



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

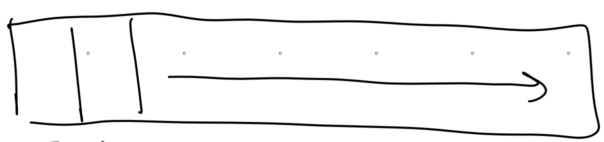
B.C.

3. Convert into recurrence for objective function

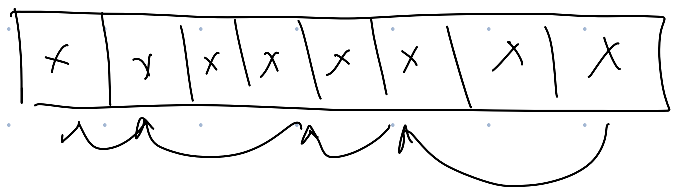
$$A_n = \max\{A_{n-1},\ A_{n-2} + w(v_n)\}, \quad \text{B.C.}$$

4. Pseudocode:
   - create A array with FOR loop from base case
   - work backwards through A to get solution

A



B.C.



Knapsack Problem

Input: n items

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

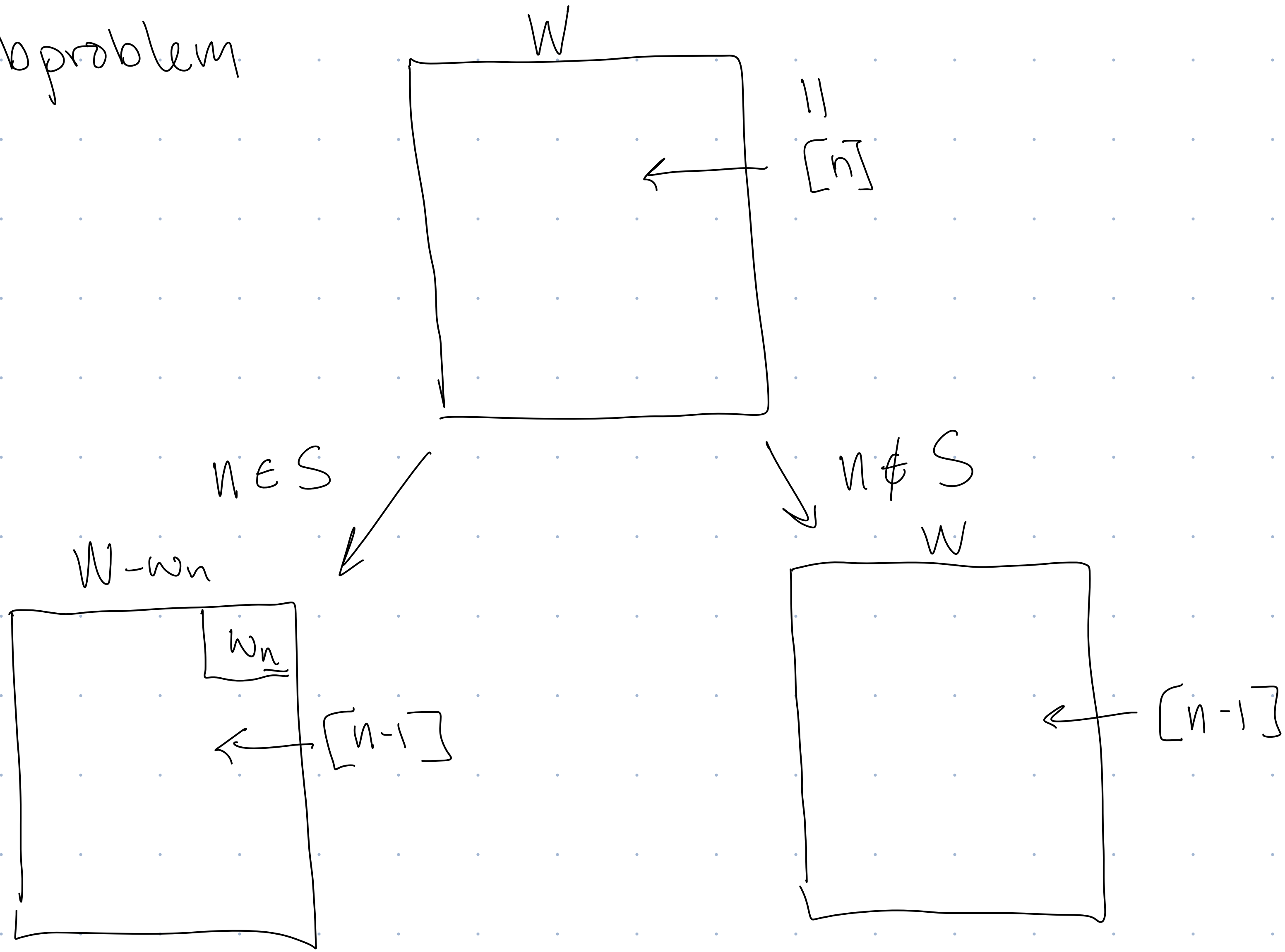
Capacity W (max weight)

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

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?
   - Item n added or not

2a Subproblem



In subproblems, Weight changes, items available change.
Need subproblems that depend on both

Define:

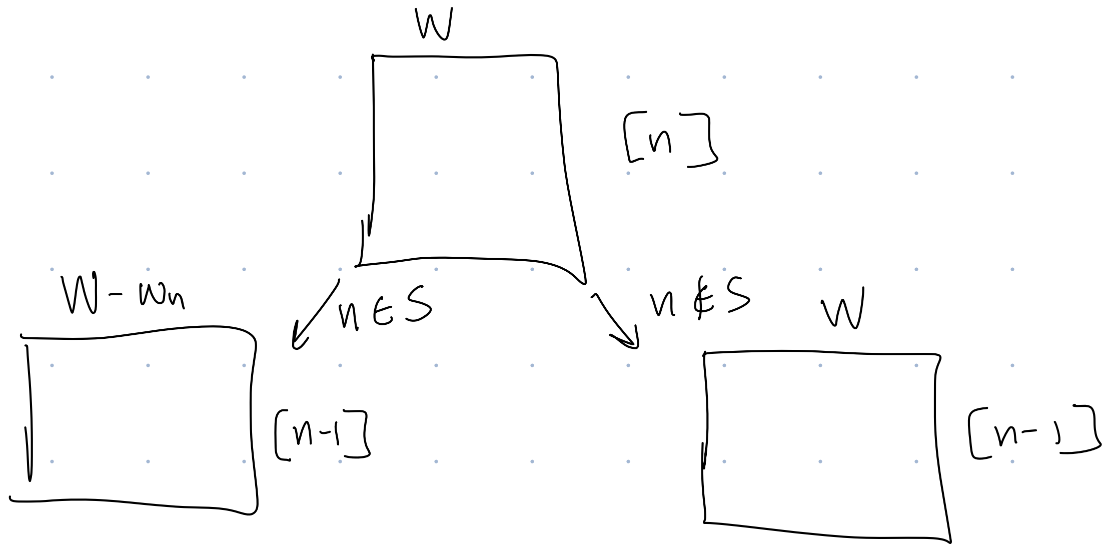
$$S_{i,w} = \left\{ S \subseteq [i] \text{ s.t. } \begin{array}{l} \bullet\ W(S) \leq w \\ \bullet\ V(S) \text{ is maximized} \end{array} \right.$$

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

What is $S_{2,4}$?

A) $\{1, 1\}$ $\quad$ B) $\boxed{\{1, 2\}}$ $\quad$ C) $\{2, 3\}$ $\quad$ D) $\{4\}$

2b



$$S_{n,w} = \begin{cases} S_{n-1,\, w-w_n} \cup \{n\} & n \in S_{n,w} \\ S_{n-1,\, w} & n \notin S_{n,w} \end{cases}$$

Base case later

Value of optimal set

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

Want to store in an Array

$$A[i, c] = \begin{cases} \max\{ \underbrace{A[i-1, c-w_i] + v_i}_{i \in S} \, , \, \underbrace{A[i-1, c]}_{i \notin S} \} \\ \text{Base case(s)} \end{cases}$$

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

	1	2	3
value	6	5	8
weight	2	1	3

$W = 5$

capacity (weight)

[n]	0	1	2	3	4	5
$\phi \leftarrow 0$	0	0	0	0	0	0
$\{1\} \leftarrow 1$	0	0	6	6	6	6
$\{1,2\} \leftarrow 2$	0	5	6	11	11	11
$\{1,2,3\} \leftarrow 3$	0	5	6	11	13	14

this item only possible if $c \geq w_i$, room for i

$$A[n, W] = \begin{cases} \max\{ \overbrace{A[n-1, W-w_n] + v_n}^{n \in S}, \overbrace{A[n-1, W]}^{n \notin S} \} \\ \boxed{0 \quad \text{if} \quad n = 0} \end{cases}$$

?

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 ← 0 to W:
|   A[0, c] = 0 // Base case
For i ← 1 to n:
|   For c ← 0 to W:
|   |   A[i, c] ← A[i-1, c]
|   |   If c ≥ w[i] and A[i-1, c-w[i]] + v[i] > A[i-1, c]:
|   |   |   A[i, c] ← A[i-1, c-w[i]] + v[i]
S ← ∅
i ← n
c ← W
While i > 0:
|   if A[i, c] = A[i-1, c]: i--
|   else: S ← S ∪ {i}
|   |   c ← c - w[i]
|   |   i --
```

c

i \ c	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	6	6	6	6
2	0	5	6	11	11	11
3	0	5	6	11	13	14

Tips for working backwards through A:

- Use if-else sequences, not if-if
  - → In edge case where two terms are equal:

$$\max\left\{ \boxed{A[n-1, W-w_n] + v_n}, \boxed{A[n-1, W]} \right\}$$

equal

your code will fail :(

- Pick which option to check first wisely to avoid index-out-of-bounds errors
  - → If check if $A[i, c] = A[i-1, c]$ first, no I-O-O-B error.
  - → If check if $A[i, c] = A[i-1, c-w[n]] + v[n]$ first can get I-O-O-B error if $c-w[n]$ goes out-of-bounds. Would need extra if statement to avoid.

Runtime

$O(nW)$

```
For c ← 0 to W:
| A[0, c] = 0 // Base case
For i ← 1 to n:
|   For c ← 0 to W:
|   |   A[i, c] ← A[i-1, c]
|   |   If c ≥ w[i] and A[i-1, c - w[i]] + v[i] > A[i-1, c]:
|   |   |   A[i, c] ← A[i-1, c - w[i]] + v[i]
S ← [∅]
i ← [n]
c ← [W]
While (i > 0 and c > 0):
|   if A[i, c] = A[i-1, c] :  i--
|   else:
|   |   S ← S ∪ {i}
|   |   c ← c - w[i]
|   |   i --
```