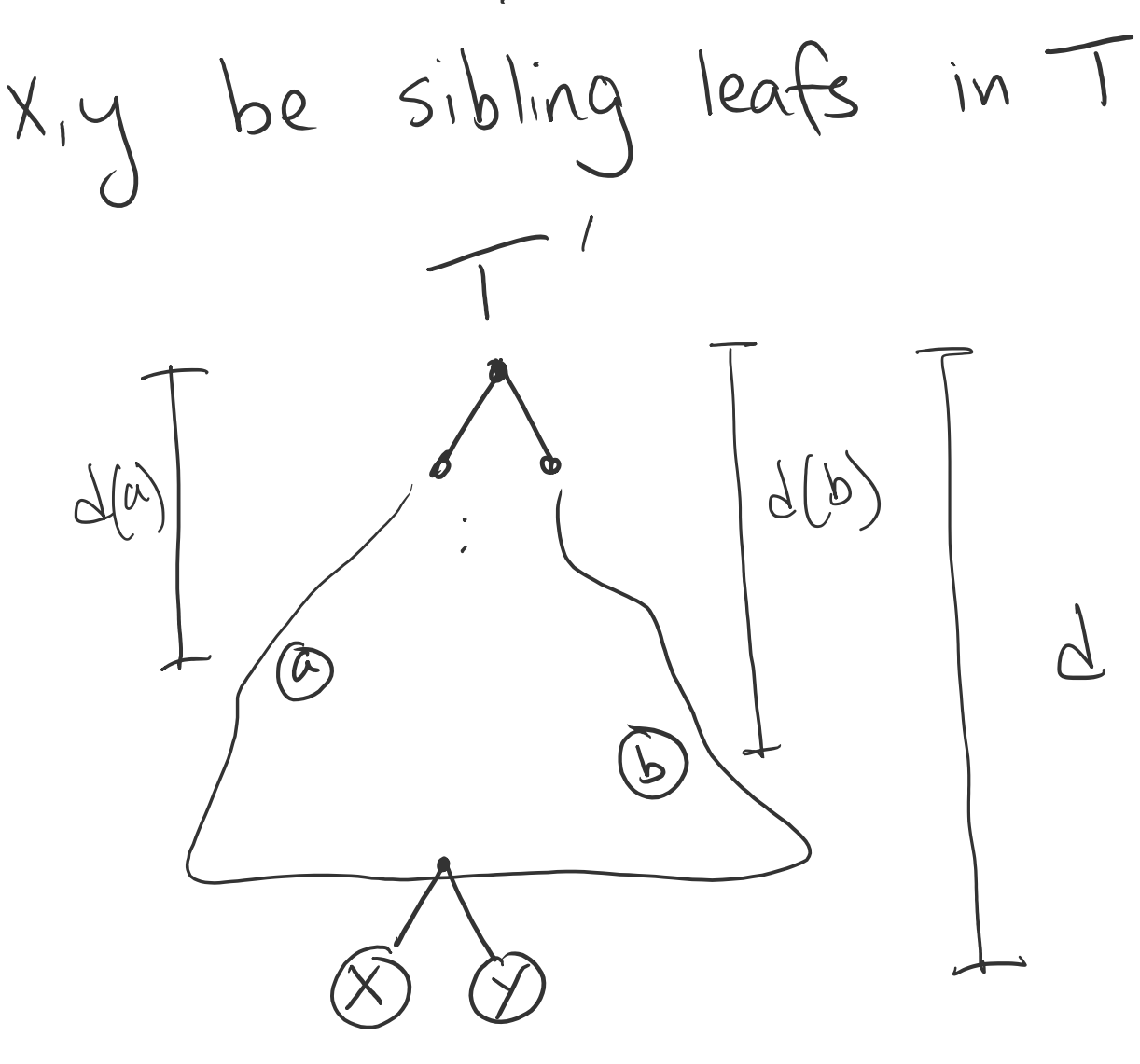
Goals:
- "Finish" proving Huffman's algorithm is optimal
- Understand Knapsack Problem
- Review Dynamic Programming Approach

Built-world accessibility barriers

Lemma 1: There is an optimal tree for Σ with a,b siblings.

Let T' be any optimal tree without a, b siblings. Let x,y be sibling leafs in T' at maximum depth.



If we exchange positions $x \leftrightarrow a$ and $y \leftrightarrow b$, then new tree T'_{ex} will have the same or smaller average length as T' because... Thus, ...

8: Knapsack

Friday, April 2, 2021
11:51 AM

Knapsack Problem

Input:
- n items
  - value v_i
  - cost c_i (volume taken up) } $\in \mathbb{Z}^+$
- Capacity C

Output: A subset $S \subseteq \{1, 2, \dots, n\}$ that maximizes value

$V(S) = \sum_{i \in S} v_i$ ⇐ objective function

and satisfies

$C(S) = \sum_{i \in S} c_i \leq C$ ⇐ constraints

Ex: $C = 5$

Item	Value	Cost
1	6	2
2	5	1
3	8	3
4	7	4
5	7	4

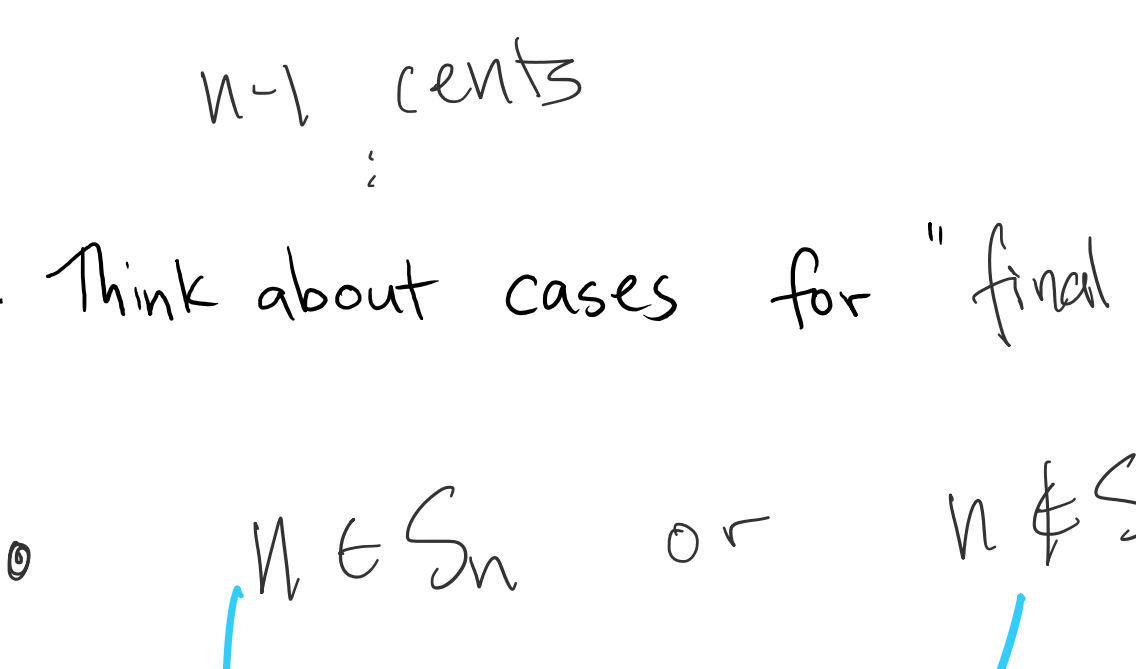
(items 1–4: one of each item; row 5: ← adding a second copy of item 4)

What is optimal S:

A): {1, 3} B) {1, 4} C) {1, 2, 3}, D) {2, 3}

Designing a Dynamic Programming Alg.

0. Create series of increasingly smaller subproblems
   Recurrence object: Optimal output of each subproblem

- 

  S_i = MWIS of G_i

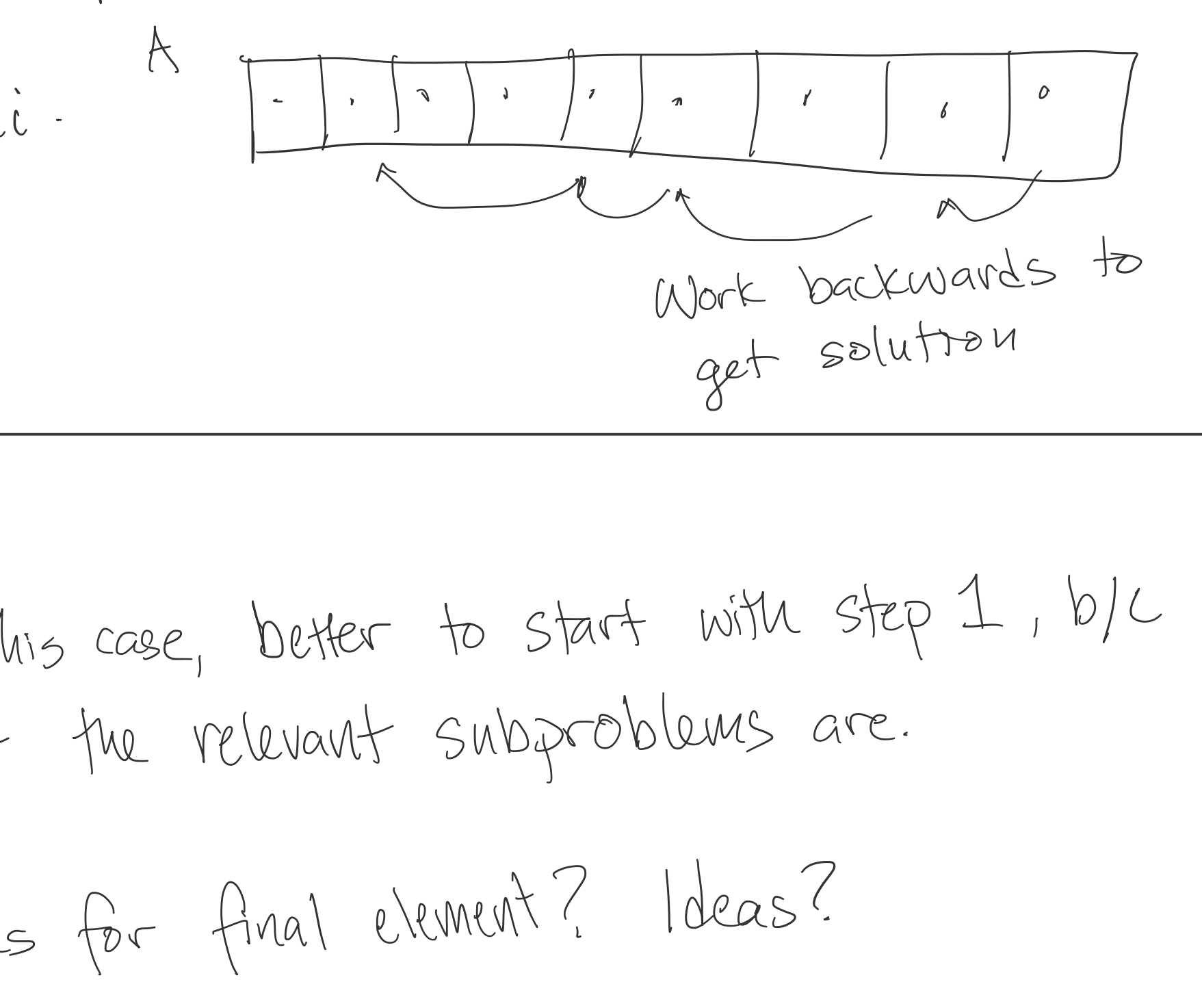
- n cents
  n-1 cents
  ⋮
  M_i = optimal coins to make i cents

1. Think about cases for "final elements" of recurrence object
   - $n \in S_n$ or $n \notin S_n$
   - last coin was 1, 4, or 6.

2. For each case, create recurrence in terms of smaller recurrence objects
   - $S_i = \begin{cases} S_{i-1} \\ S_{i-2} \cup \{v_n\} \end{cases}$ & B.C.
   - $M_i = (M_{i,1}, M_{i,4}, M_{i,6}) = \{$

3. Convert into recurrence for objective function
   - $W(S_n) = \max\{W(S_{i-1}), W(S_{i-2}) + w_n\}$ & B.C.
   - $C(n) = \min\{ \quad \}$ & B.C

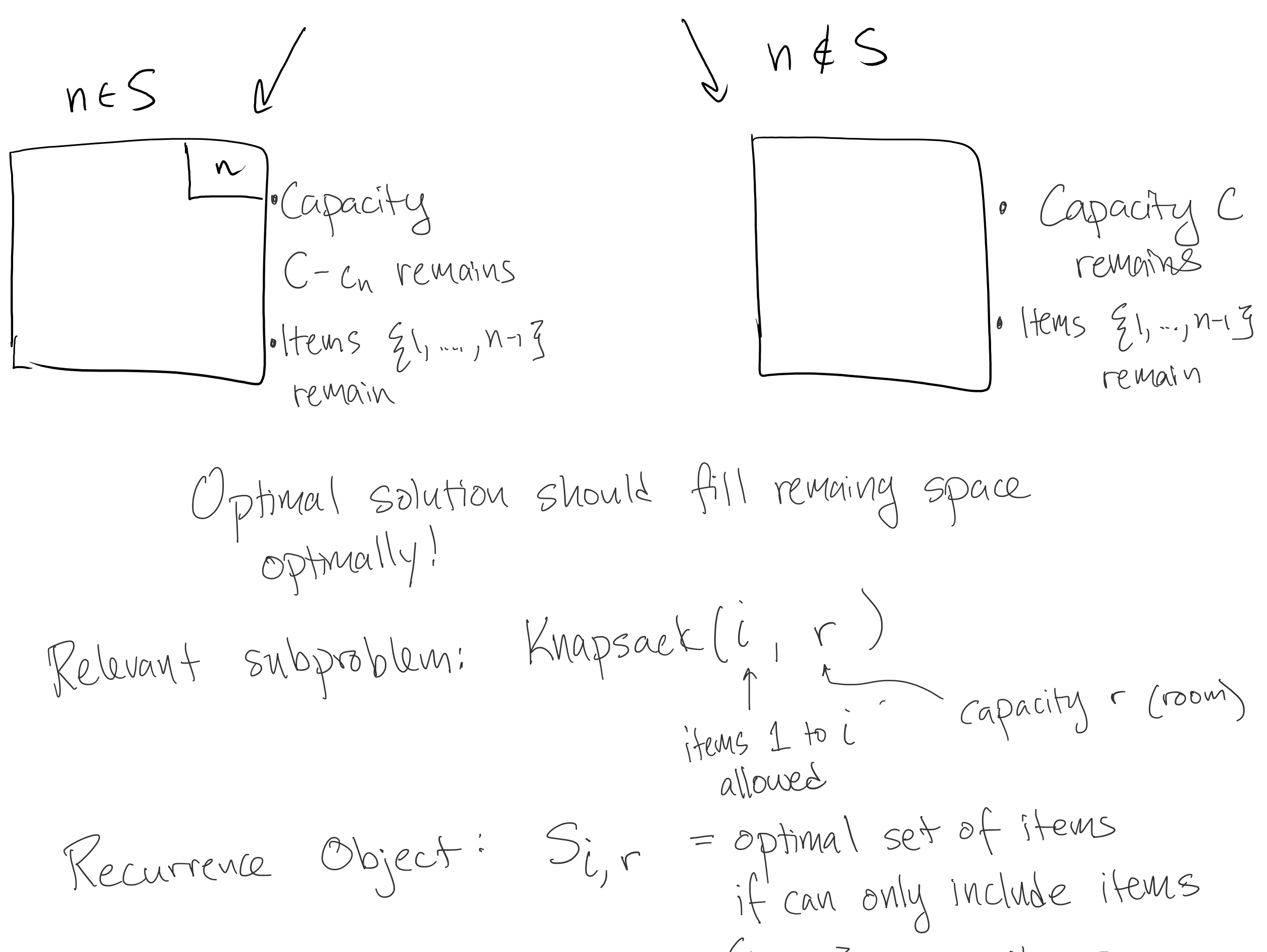
4. Pseudocode: Store $W(S_n)$, $C(n)$



In this case, better to start with step 1, b/c not clear what the relevant subproblems are.

Cases for final element? Ideas?

$n \in S_n$ vs $n \notin S_n$



Optimal solution should fill remaing space optimally!

Relevant subproblem: Knapsack(i, r)
- i: items 1 to i allowed
- r: capacity r (room)

Recurrence Object: $S_{i,r}$ = optimal set of items if can only include items $\{1, \dots, i\}$, capacity r

What is $S_{2,4}$

A) {1, 1}
→ B) {1, 2}
C) {2, 3}
D) {4}

Item	Value	Cost
1	6	2
2	5	1
3	8	3
4	7	4