Dynamic Programming:

Example 1: LIS = longest increasing subsequence.

Given n numbers $a_1, \ldots, a_n$,
find the length of the LIS.

Example: 5, 7, 4, -3, 9, 1, 4, 8, 6, 7, 5

LIS = 5 from -3, 1, 4, 6, 7

First step, define subproblem in words, then define recurrence. If can't find a recurrence then probably get an idea how to revise

Attempt 1:

Let $T(i)$ = length of LIS in $a_1, \ldots, a_i$

What's the recurrence?

For the above example, $T(6)=3$ from 5, 7, 9,
but we want -3, 1 so that $T(7)=3$ from -3, 1, 4.

Then for $T(8)$ can we add 8 on?

Yes if it's -3, 1, 4, no if it's 5, 7, 9,
how do we know which?
Keep track of all possible endings, so try the following.

Attempt 2:

Let $T(i)$ = length of LIS in $a_1, \ldots, a_i$ which includes a_i

Now we know the end so we know if we can add to it.

Hence, $T(i) = 1 + \max_{1 \le j < i} \{T(j) : a_j < a_i\}$

Algorithm:

LIS($a_1, \ldots, a_n$)

for $i = 1 \to n$
  $T(i) = 1$
  for $j = 1 \to i-1$
    if $a_j < a_i$ then $T(i) = \max\{T(i), 1 + T(j)\}$

max = 1
for $i = 2 \to n$
  if $T(i) > T(\max)$ then max = i

Return($T(\max)$)

Running time: $O(n^2)$

Knapsack:

n objects with integer weights $w_1, \ldots, w_n$
& integer values $v_1, \ldots, v_n$
total capacity B

What's subset S of objects where

$$\sum_{i \in S} w_i \leq B$$

& which maximizes $\sum_{i \in} v_i$

Version 1: one copy of each object.

Attempt 1: Let $T(i)$ = max value attainable using subset of objects $1, \ldots, i$

But then for $T(i)$ can we add object i to optimal solution for $T(i-1)$? May want suboptimal solution for $T(i-1)$ which has enough capacity so that can add object i.
So want to see optimal solution for given capacity.

Attempt 2: Let $T(i,b)$ = max value attainable using subset of $1, \ldots, i$ & total capacity $\leq b$.

<u>Recurrence</u>: if $w_i \le b$

$$\text{then } T(i,b) = \max\{\overset{\text{Don't use } i}{T(i-1,b)},\ \overset{\text{best of } 1,\dots,i-1 \text{ with remaining weight}}{T(i-1,b-w_i)} + \overset{\text{Use } i}{v_i}\}$$

$$\text{else } T(i,b) = T(i-1,b)$$

<u>Final answer</u>: $T(n,B)$

<u>Running time</u>: $O(nB)$

<u>Version 2</u>: Unlimited supply of each object.

Now we don't need to keep track of what objects are used so far.

Let $T(b)$ = max value attainable using total capacity $\le b$.

For the recurrence, try all possibilities for the last object to add.

$$T(b) = \max_{1 \le i \le n} \{T(b-w_i) + v_i : w_i \le b\}$$

<u>Final result</u>: $T(B)$

<u>Running time</u>: $O(nB)$

Knapsack has running time $O(nB)$.

Is this a polynomial-time algorithm?

No, the number B is part of the input and it's input size is $O(\log B)$.

We'll see later that knapsack is NP-complete.

We'll reduce from a 3-SAT instance with n variables & m constraints, and the knapsack instance will have B = exponentially large in n & m.