Dynamic Programming approach:

1) Define subproblem in words, e.g.,
   \[ F(i) = \text{ith Fibonacci number} \]

2) State a recurrence in terms of smaller subproblems, e.g.,
   \[ F(i) = F(i-1) + F(i-2) \]

3) Solve subproblems from smallest to largest

Longest increasing subsequence:

Input: a numbers \(a_1, a_2, \ldots, a_n\)

E.g., 5, 2, 8, 6, 3, 6, 9, 7

A subsequence is a subset in order

E.g., using indices 2, 4, 5, 7 gives the subsequence 2, 6, 3, 9

So a subsequence is a subset \(a_{i_1}, a_{i_2}, \ldots, a_{i_k}\)

where \(1 \leq i_1 < i_2 < \ldots < i_k \leq n\)

(So increasing indices)
Subsequence is increasing if

$$a_i < a_{i+1} < \ldots < a_{i+k}$$

e.g., $$a_2, a_5, a_6, a_7 = 2, 3, 6, 9$$

is an increasing subsequence since $$2 < 3 < 6 < 9$$.

Goal: Given $$a_1, \ldots, a_n$$, find an increasing subsequence of max length.

First, let's focus on finding just the length of the longest increasing subsequence (LIS).

First step: define the subproblem in words.

Natural idea:

$$S(i) = \text{length of longest increasing subsequence in } a_1, \ldots, a_i$$

Goal: compute $$S(n)$$. 

How to write a recurrence for $S(j)$ in terms of $S(1), S(2), \ldots, S(j-1)$?

For example in $5, 2, 8, 6, 3, 6, 9, 7$

$S(5) = 2$

to figure out $S(6)$ we need to know where the LIS for $S(5)$ ends at.
E.g., if $S(5) = 2$ corresponds to $5, 8$ then we can not add 6 on to it.
But if $S(5) = 2$ corresponds to $2, 3$ then we can add 6 on to it.

So we need to keep track of all possible endings.

Cleaner approach: add extra condition to the definition of the subproblem to remember what number it ends at.
Let \( L(j) \) = length of longest increasing subsequence in \( a_1, \ldots, a_j \) which ends at \( a_j \) (\& includes \( a_j \)).

Goal: compute \( \max_j L(j) \)

View as a graph:

Then, \( L(j) = \) length of longest path ending at \( a_j \).

For recurrence for \( L(j) \):
- Look at all incoming edges \( (i, j) \).
- Take \( L(i) + 1 \).

\[
L(j) = 1 + \max_i L(i) : i < j, a_i < a_j \]
LIS(A)
input: A = [a₁, ..., aₙ]

for j = 1 → n
    L(j) = 1
    for i = 1 → j - 1
        if L(i) + 1 > L(j) & aᵢ < aⱼ
            then L(j) = L(i) + 1

Let max = 1
for i = 1 → n
    if L(i) ≥ L(max)
        then max = i

Return (L(max))

Running time: O(1) per i
O(n) sized loop over i
O(n) sized loop over j
O(n) × O(n) × O(1) = O(n²) total time
How to find the actual subsequence?

keep track of the next to last index i

that gives the max

then backtrack

Earlier example:

$A = 5 \ 2 \ 8 \ 6 \ 3 \ 6 \ 9 \ 7$

$L = 1 \ 1 \ 1 \ 2 \ 2 \ 3 \ 4 \ 4$

$\text{Prev} = \begin{bmatrix} \text{NULL} & \text{NULL} & 1 & 1 & 2 & 5 & 6 & 6 \end{bmatrix}$

To reconstruct $S(7)$, follow the path:

$\text{NULL} \leftarrow a_8 \leftarrow a_5 \leftarrow a_6 \leftarrow a_7$

$2 \leftarrow 3 \leftarrow 6 \leftarrow 9$

So it's: 2, 3, 6, 9
LIS(A)

for j = 1 → n
    \(L(j) = 1, \text{prev}(j) = \text{NULL}\)

for i = 1 → j - 1
    if \(L(i) + 1 > L(j) \& \& a_i < a_j\)
        then \(L(j) = L(i) + 1\)
        \(\text{prev}(j) = i\)

let max = 1

for i = 1 → n
    if \(L(i) > L(\text{max})\) then max = i

to get the subsequence:

\[ i = \text{max} \]
\[ \text{output}(i) \]
\[ \text{while prev}(i) \neq \text{NULL}: \]
\[ i = \text{prev}(i) \]
\[ \text{output}(i) \]
Key ideas:

- Use prefix of input for subproblems
- Strengthen subproblem by adding extra info into it.