Review: $O(\cdot)$ notation.

For functions $f(n)$ and $g(n)$,

\[ f(n) = O(g(n)) \text{ if there is a constant } c > 0 \text{ where } f(n) \leq cg(n) \]

Examples:

a) $f(n) = 3n^2 + 10n - 5n^{2.5} + 1.7n^3$

\[ f(n) = O(n^3) \text{ and } f(n) = O(n^5) \]

b) $g(n) = 5\log^2 n + 7\sqrt{n}$

\[ g(n) = O(f(n)) \]

c) $f(n) = n^2$, $g(n) = 2^{4/\log n}$

Don't specify base then base 2

So: $2^{\log n} = n$

$\ln n = \log_{\text{base } 2} n$
\[ \log_{10} n = O(\log n) \]
\[ \ln n = O(\log n) \]

**Manipulating \( \log \)'s:**

\[ g(n) = 2^{4\log n} = (2^{\log n})^4 = n^4 \]

hence, \( f(n) = O(g(n)) \) but \( g(n) \neq O(f(n)) \).

**How about \( f(n) = 3^{\log_5 n} \) express as a polynomial in \( n \) for constant \( c > 0 \).**

\[ 3^{\log_5 n} \]

want to match so note \( 3 = 5^{\log_5 3} \)

then:

\[ f(n) = 3^{\log_5 n} = (5^{\log_5 3})^{\log_5 n} = (5^{\log_5 n})^{\log_5 3} \]

\[ \log_5 3 < 1 \] so \( f(n) = O(n) \).
Dynamic Programming (DP):

Toy example: Computing Fibonacci numbers:

\[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots\]

defined by:

\[F_0 = 0, F_1 = 1\]

and for \(n > 1\), \(F_n = F_{n-1} + F_{n-2}\)

Natural recursive algorithm:

\[
\text{Fib1}(n):
\]

if \(n = 0\), return(0)
if \(n = 1\), return(1)
return(Fib1(n-1) + Fib1(n-2))

Running time ???

Look at time as a function of input size \(n\)
in this case \(n = n^{th}\) Fibonacci

Other examples:
\(n = \#\) of numbers to sort
\(n = \#\) of vertices in input graph
\(n = \#\) of bits in input numbers to multiply
Look at running time in $O(1)$
ignore constant factors since depend on specific machine.

**Model of computation:**

- $O(1)$ time to add/subtract/multiply/divide any basic arithmetic operation
- Unlimited memory
- $O(1)$ time to read/write unit of memory

**Analyzing Fib1$(n)$:**

Let $T(n) =$ # of steps for computing $i^{th}$ Fibonacci

$T(0) = O(1)$ & $T(1) = O(1)$

for $n > 1$: $T(n) = T(n-1) + T(n-2) + O(1)$

hence, $T(n) \geq F(n)$

$T(n-1) + T(n-2) + c \geq F(n-1) + F(n-2)$

$F(n) = \frac{\phi^n}{\sqrt{5}}$ where $\phi = \frac{1 + \sqrt{5}}{2} \approx 1.618$ is the golden ratio.

So exponential-time algorithm.
Why is Fib1() so slow?

\[
T(n) \\
T(n-1) \quad T(n-2) \\
T(n-2) \quad T(n-3) \quad T(n-3) \quad T(n-4) \\
T(n-3) \quad T(n-4) \\
\ldots
\]

Recomputing small subproblems many times.

Better approach:

Work bottom-up.
Start with smallest F(0) & F(1).
& go to larger
then only solve each subproblem once.
**Fib2(n):**

- if \( n = 0 \), return (0)
- if \( n = 1 \), return (1)
- create an array \( F[0...n] \)
- \( F[0] = 0, F[1] = 1 \)
- for \( i = 2 \rightarrow n \):
  - \( F[i] = F[i-1] + F[i-2] \)
- return \( F[n] \)

**Running time:**

- \( O(1) \) per \( i \) & \( O(n) \) sized loop over \( i \)
- \( \Rightarrow O(n) \) total time.
Non-trivial example: Longest increasing subsequence (LIS)

Input: n numbers $a_1, a_2, \ldots, a_n$

Length of longest subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_L}$

which is increasing $a_{i_1} < a_{i_2} < \ldots < a_{i_L}$

Note: a subsequence is a subset of indices in order: $1 \leq i_1 < i_2 < i_3 < \ldots < i_L \leq n$.

Example: $A = 5, 2, 8, -1, 6, 3, 6, 9, 2, 4, -3, 7$

$LIS = -1, 3, 4, 7$ (or $2, 3, 6, 7$, etc.)

Length = 4

First step:

Define the subproblem in words

(think inductive proof; define inductive hypothesis)

Natural idea: let $S(j) =$ length of LIS in $a_1, \ldots, a_j$

Same problem on prefix

(always use prefixes as first attempt)
Goal: compute \( S(n) \).

Second step: Write recurrence for \( S(j) \) in terms of \( S(1), \ldots, S(j-1) \).

Idea: \( S(j) \) is max of \( S(j-1) \) and \( 1 + \) best of \( S(j-1) \) ending at \( \leq a_j \) best LIS in \( a_1, \ldots, a_{j-1} \) + best LIS that can add \( a_j \) to.

Example: \[ A = 5, 2, 8, -1, 6, 3, 6, 9, 2, 4, -3, 7 \]

\( S = 1, 1, 2, 2, 2, 2, ? \)

For \( S[7] \), how do we know what \( S[6] \) ends at?

Many possible endings

Want smallest ending character which is \( 3 \) from \( -1, 3 \)

But for \( S[4] = 2 \) corresponding to \( 5, 8 \)

So how do we have want LIS ending at \(-1\).
Possible ending numbers are the numbers appearing in A.

For each ending number what's LIS ending at it.

Let $L(j)$ = length of LIS in $a_1, \ldots, a_j$ ends at $a_j$ & includes $a_j$.

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

Recurrence: for $a_j$

$$L(j) = 1 + \max_i L(i) : i < j, a_i < a_j$$

length of LIS ending at $a_i$ for $a_i < a_j$ & $i < j$. 
**LIS(A):**

\[
\text{for } j = 1 \rightarrow n \\
L(j) = 1 \\
\text{for } i = 1 \rightarrow j-1 \\
\text{if } L(i) + 1 > L(j) \& a_i < a_j \\
\quad \text{then } L(j) = L(i) + 1 \\
\quad \text{Prev}(j) = i
\]

Let max = 1

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

Return \( L(\text{max}) \)

**Running time:**

\( O(1) \text{ per } i \times O(n) \text{ sized loop } \times O(n) \text{ loop over } i \)

\( O(1) \times O(n) \times O(n) = O(n^2) \)
Earlier example:

\[ A = 5, 2, 8, -1, 6, 3, 6, 9, 2, 4, -3, 7 \]
\[ L = 1, 1, 2, 1, 2, 2, 3, 4, 2, 3, 1, 4 \]

How to find LIS?

Keep track of \( i \) that achieves the max using \( \text{Prev}(j) \)

Then backtrack

So, \(-1, 2, 4, 7\).