A basic algorithm paradigm like Greedy, Divide-and-Conquer. Those have names making great sense. DP's name chosen in '40s literally to be obfuscatory, to make CS research sound cool to army bosses. Better names: Memo(r)ization, Recursion w/o repetition. Hallmark: Like Divide and Conquer but with few, nested subprobs.

Speaking of recursion, most traditional intro to it uses...? Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, ...

Any guesses? $F_{100} \approx 3.54$ quintillion. $F(n) \approx \varphi^n / \sqrt{5}$ digits

Recursive alg:
```python
def Fib(n):
    if n <= 1 return n
    return Fib(n-1) + Fib(n-2).
```

Correct "by def": Time? Let $T(n) = \# \text{rec. calls made by } \text{Fib}(n)$. $T(0) = 0$, $T(1) = 1$, $T(2) = 2$. If $T(n) = 2 \cdot T(n-1) + T(n-2)$, about $10^{n/2}$, $\varphi^{n-1.6}$, $\exp(n)$.
"Memoization": 

Fib(n):
   if Memo(n) ≠ null, return Memo[n]
   if n ≤ 1: answer := n
   else answer := Fib(n-1) + Fib(n-2)
   Memo[n] := answer
   return answer

[How many recursive calls total?]
- every time we do recursive call, we do 2, but
- we fill one entry of Memo[]
  \[ \text{rec calls} ≤ 2(n+1) \]
  \[ = \text{# entries memo} \] [exponential improvement!]

"Top-down D.P.":

Define recursive algo, memoize.
Can sort of always do it, but when does it
really help? When \"few\" (O(n), O(n^2))
subprobs.

Conversely, think abt. recursive MergeSort...
subprobs are like unsorted lists \rightarrow\ exp. many

Say, how does recursion actually work?
Actually fills Memo in L-to-R order

[Can do this intentionally...]

Not magic. Your chip just serially
does instructions,
It effectively builds the above
tree...

[Image: Diagram of recursive tree with nodes labeled: 0, 1, 2, 3]
**Bottom-Up D.P.:** fill out memoization results in order of recursion would...

\[
\text{Fib}(n): \quad \begin{aligned}
\text{base: } & F[0] := 0, F[1] := 1 \\
\{ & \text{for } i = 2 \ldots n \} \\
F[i] & := F[i-1] + F[i-2] . \quad \text{/dah!}
\end{aligned}
\]

\[O(n) \quad \text{/ adding two } O(n) \text{-digit nums!} \]

**Time:** \(O(n)\)

**Space?** \(n\) array entries of \(\leq n\) digits \(\approx O(n^2)\)

**Often space can be reduced in D.P.:** selective forgetting of stale memoization results you're sure you won't need?

\[
\begin{aligned}
& \text{if } n \leq 1 \text{ return } n : \\
& \text{prev} := 0 \\
& \text{curr} := 1 \\
& \text{for } i = 2 \ldots n : \\
& \text{next} := \text{curr} + \text{prev} \\
& \text{prev} := \text{curr} \\
& \text{curr} := \text{next} \\
& \text{return } \text{curr}
\end{aligned}
\]

\[\text{// think about it!} \]
// Another important alg. for comp bio/genomics...
L.C.S. = Longest Common Subsequence.

Input: two strings $x, y$; e.g. $x = GACATTA CGA$; $y = GCTA CAGT$

Out: L.C.S. length of $x \setminus y$ (not substring; discont'd ok)

[[ Like you can match chrs up but, in a non-crossing matching way...]]

[[ Brute force bad, $2^n$ subsets of $x$]]

[[ Basic first idea of DP:]] Find recursive definition/alg.

[[ Not totally obv, but some greed is good.]]

• if $x[i] = y[j]$, may as well "take it"  
  (can never hurt; scores a pt; blocks nothing)
  \( \Rightarrow \) ans. is \( 1 + \text{LCS}(x[2\ldots n], y[2\ldots m]) \)

• else...
  \( \text{LCS}(x[1\ldots n], y[2\ldots m]) \) or \( \text{LCS}(x[2\ldots n], y[1\ldots m]) \), whichever better.

Q: How to "index" the subprobs?  
A: \( x[0\ldots n], y[0\ldots m] \).

def LCS(i, j): // find LCS len of
  if \( i > n \) or \( j > n \), return 0  // got down to at least one empty string
  else if \( x[i] = y[j] \): answer: = \( 1 + \text{LCS}(i+1, j+1) \)
  else answer: = \( \max(\text{LCS}(i+1, j), \text{LCS}(i, j+1)) \).
  return answer.

// exercise: show # of rec. calls = \( 2^n \cdot 2^m \)
Memoize!

\[
\begin{align*}
\text{if } \text{Memo}\{i,j\} = \text{null} : \\
\text{Memo}\{i,j\} := \text{answer}.
\end{align*}
\]

Time? Each \( \log_2 n \) rec. calls gives \( n \) Memo entry \( \approx n \log_2 n \).

"Bottom-up"?

in what order are entries actually filled...?

better to think: what's a sensible order to fill so that I'll always have the prior answers I'll need?

\[
\begin{array}{cccc}
\text{R} & \text{A} & \text{R} & \text{A} \\
3 & 2 & 1 & 1 \\
3 & 2 & 2 & 1 \\
4 & 1 & 1 & 0 \\
\end{array}
\]

By rows or cols ok, just from \( m \) \( \downarrow \) \( n \).

eq: \( x = \text{RARE} \) y = \text{AREA}

for \( i = 1 \ldots n+1 \)

\( \text{LCS}\{i, n| \} := 0 \)

for \( j = 1 \ldots m+1 \)

\( \text{LCS}\{n, j| \} := 0 \)

\[
\begin{align*}
\text{if } x(i) = x(j) : \\
\text{LCS}\{i,j\} &= 1 + \text{LCS}\{i-1, j-1\} \\
\text{else} &
\end{align*}
\]

Q: Given Memo\{\}, how to find actual subseqs?

A: Start at top-left, "follow decisions".

\( 3 \)

\( \text{del} R \)

\( \text{del} A \)

\( \text{del} x \)

\( \text{del} y \)

\( \text{del} E \)
Improve Space usage?  only "needs"

... can get space to $O(mn)$

Not so simple to see w/ only using $O(mn)$ space, but can be done.

Clear!

<< quadratic time?  No! So sad, but ~5 years ago proved "impossible" (assuming $P \neq NP$ variant).

One more example ... just try to set up recursion...

Max-Weight-Indep-Set:
Input: $G = (V,E)$, "weights" $w(v) > 0 \forall v \in V$.
Output: An "indep set": $S \subseteq V$ s.t. no two verts in $S$ have an edge.

Goal: Maximize $w(S) = \sum_{v \in S} w(v)$.

Think: places to put broadcasting dev.

Get pts for placing at vert.
Can't have two adj. Interference!

NP-hard! 😞  ❄️  [no subexp. time alg. prob.]
Efficiently solvable if $G$ is a tree.

More generally if $G$ has small "freewidth".

Make $G$ a rooted tree:

Trees + recursion:

M.W.A.!: A rooted tree is a node, with children who are rooted trees.

**MWIS**($r$) := \[ \max \sum \text{weights of } r, \text{ independent set } \text{ you can get in } \text{ tree rooted at } r \]

// Idea: can either "take" $r$, get $\sum$, but then you can't take kids... but the grandkids probs are sep

\[ \text{or don't take, and solve for kids}... \]

\[ \text{MWIS}(r) := \max \{ w(r) + \sum \text{MWIS}(v), \sum \text{MWIS}(u) \} \]

MWIS(leaf $l$) := weight $l$.

Memoize as before! O(n) space, one slot for each vert.

Bottom-up? (A bit annoying, need to traverse from leaves up)

Great case for using recursion and letting compiler handle it!!