Dynamic Programming:

Toy example: computing Fibonacci numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

$F_0 = 0$, $F_1 = 1$, for $n > 1$: $F_n = F_{n-1} + F_{n-2}$

Recursive algorithm:

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

$T(n)$ = # of steps for Fib1(n)

for $n \leq 1$, $T(n) = O(1)$

for $n \geq 2$, $T(n) = T(n-1) + T(n-2) + O(1)$

Note: $T(n) \geq F_n$

& $F_n \simeq \frac{\phi^n}{\sqrt{5}}$ where $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803\ldots$ is the golden ratio.

Dynamic Programming algorithm:

```
Fib2(n)
    F[0]=0, F[1]=1
    for i=2→n
        F[i] = F[i-1] + F[i-2]
    Return(F[n])
```

Running time is $O(n)$.

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
Second step: define recurrence – if can't then modify

Attempt 1: Try same problem on prefix of input.

Let $L(i)$ = length of LIS in $a_1, \ldots, a_i$
What's the recurrence?

Earlier example:

A =	5	7	4	-3	9	1	4	8
L =	1	2	2	2	3	3	4	1

$L(7) = L(6) = 3$ from 5, 7, 9

$L(8) = 4$ from -3, 1, 4, 8
but did we keep track of
-3, 1, 4 at $L(6)$?
Need to remember all possible endings
but do we keep suboptimal -3, 1
for $L(5)$?

Attempt 2:

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

Now we know that the solution for $S(i)$ ends at a_i

So we know if we can add a_j on to it.

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

Algorithm:

LIS($a_1, \ldots, a_n$):

```
for j=1→n
    T(j)=1
    for i=1→j-1
        if a_i<a_j & T(i)+1>T(j)
            then T(j)=1+T(i)

max=1
for j=2→n
    if T(j)>T(max) then max=j
Return(T(max))
```

- Running time: $O(n^2)$

LCS = longest common subsequence

input: 2 strings $X = x_1 x_2 \cdots x_n$ & $Y = y_1 y_2 \cdots y_m$

goal: find length of longest string which is a subsequence of both X & Y.

Example: X = BCDBCDA

Y = ABECBA

length of LCS is 4 for BCBA

Application: used in unix diff for comparing files.

Define subproblem: try prefixes of X & Y.

For i & j where $0 \le i \le n$ & $0 \le j \le m$ let:

$L(i,j)$ = length of LCS in $x_1, \ldots, x_i$ & $y_1, \ldots, y_j$

Base cases: $L(i,0)=0$ & $L(0,j)=0$

For recurrence, two cases: $X_i=Y_j$ or $X_i \neq Y_j$.

If $X_i \neq Y_j$ then a common subsequence ends in X_i, Y_j, or neither.

if it doesn't end in X_i then $L(i,j)=L(i-1,j)$

" in Y_j then $L(i,j)=L(i,j-1)$

thus $L(i,j)=\max\{L(i-1,j), L(i,j-1)\}$

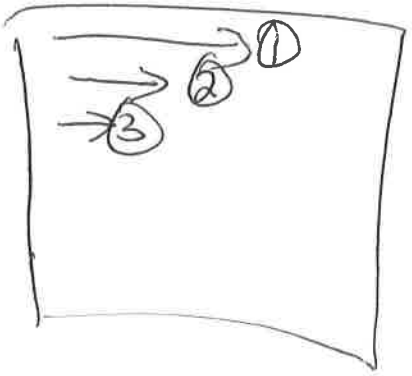
If $X_i=Y_j$ then a common subsequence ends in $X_i=Y_j$ or neither

if it ends in $X_i=Y_j$ then $L(i,j)=1+L(i-1,j-1)$

if it ends in neither then we can add on $X_i=Y_j$ at the end to make it longer.

Thus, $L(i,j)=1+L(i-1,j-1)$.

Fill the table row by row

$L=$ 

LCS(X, Y):

for $i=0 \to n$, $L(i,0)=0$

for $j=0 \to m$, $L(0,j)=0$

for $i=1 \to n$

  for $j=1 \to m$

    if $X_i = Y_j$ then

      $L(i,j) = 1 + L(i-1, j-1)$

    else

      $L(i,j) = \max\{L(i,j-1), L(i-1,j)\}$

Return($L(n,m)$)

Running time: $O(nm)$

- How would you do:

  longest common substring?

How do you find a LCS?

Backtrack

```
Print-LCS(i,j)
    if L(i,j) = 0 then return()
    if Xi = Yj then [Print-LCS(i-1,j-1)
                     output("Xi = Yj")
                     return()]
    if Xi ≠ Yj & L(i,j) = L(i-1,j)
                then [Print-LCS(i-1,j)
                      return()]
    if Xi ≠ Yj & L(i,j) = L(i,j-1)
                then [Print-LCS(i,j-1)
                      return()]
```