Dynamic programming

Algorithm Fibonacci (m)

if m ≤ 1 return 1
return Fibonacci (m-1) + Fibonacci (m-2)

m-1 levels
complexity will be ballpark
O(2^m)

(actually a bit better)

Common subproblems in different subtrees but
results are re-computed every time.
Memoize results of previous computations

- increase memory cost a bit (poly increase)
- reduce computation time a lot (exp/combinatorial → poly)
Algorithm: 

**FibDynProg (m)**

- **Array**: int $F[\text{a}]+1$
- $F[0]=1$  \| "base" of recursion \| go up
- $F[1]=1$

for (int $i=2$ to $m$)

\[ F[i] = F[i-1] + F[i-2] \]

return $F[m]$, \( \Theta(n) \) memory

$O(m) \text{ running time}$

**Candy Maze problem** (Midterm 2018)

$m \times m$ array filled in bottom half

\[
\begin{array}{ccc}
5 & 1 & 2 \\
1 & 2 & 9 \\
10 & 2 & 9 \\
\end{array}
\]

\text{Sum of elements on path $b$ as large as possible}

Algorithm: 

**Candy Maze ($a$, $i$, $j$, $m$)**

if ($i == m-1$) (last row)

return $a[i][j]$

else

return $a[i][j] + \max \left( a[i][j] + \max (\text{Candy Maze} (a, i, s, j, m)) \right) \text{Candy Maze} (a, i, s, j, m) \text{Candy Maze} (a, i, s, j, m)$
Algorithm Candy ReseDycksg (a, m) → iterative

```c
int result[m][m]
```

// bottom row same as 0's
```c
for (int i = 0; j < m; j++)
    result[m-1][j] = a[m-1][j];
```

```c
for (int i = m-2; i >= 0; i--)
    for (int j = 0; j < i; j++)
        result[i][j] = a[i][j] + \max(a[i+1][j], a[i+1][j+1]);
```

```c
return result[0][0],
```

```c
O(m) + O(m) = O(m^2)
```

Extra \( m^2 \) memory (results away) but big reduction in time!

At row i after the loops all elements are value of optimal path below. — loop invariant
Making change: $x$ out of money
$1, 25c, 10c, 5c, c$
Convert $x$ into a set of coins so few coins as possible
Max of $1 \rightarrow M_1$
$x = x - M_1 \rightarrow$ Max of $25c$

"Greedy" algorithm: only $x$ and at largest denomination matter!
Array $C[K]$ of denominations
$10c, 5c, c$ 
$x = 12c$
26c $\rightarrow$ optimal
Greedy solution: $10c + 26c \rightarrow 3$ coins not optimal!

If greedy works $\rightarrow$ awesome! But this is not always true
$Opt(0) = 0$
$Opt(x) = \begin{cases} 1 + \min_j \left[ Opt(x - C_j) \right] & \text{what is the best we could get with} \\
& \text{this set?} \\
\end{cases}$
add 1 coin
Greedy alg uses largest coin so far
Algorithm Make Change \((C, K, x)\)

\[
\text{int } \text{Opt}[\text{m}+1] \rightarrow \text{dep on } X
\]

\[
\text{Opt}[0] = 0
\]

for \(i = 1 \text{ to } m\) \(\text{// min } \text{Opt}[x-C[j]]\)

\[
m \leftarrow \text{large int for } j = 0 \text{ to } K-1 \text{ min up to } i
\]

\[
\text{if } (C[j] \leq i) \text{ then } \text{mini} = \min (\text{mini}, \text{Opt}[x-C[i]])
\]

\[
\text{and that we still have to make}
\]

\[
\text{Opt}[i] = 1 + \text{mini}
\]

\[
O(m \times K) \ (x \text{ is } \$+c \Rightarrow m = 100x)
\]

Greedy alg would only loop over \(C \Rightarrow O(K)\)!
Longest increasing subsequence

1 3 2 5 4 8 1 9

Subset of array increasing \( i < j \)
\[ a[i] < a[j] \]
as long as possible

For all lengths \( k \):
For all possible subsequences, check if order condition is satisfied

Loop over a large space, check each element
"Combinatorial" space
"Guess and check"