\[ F_0 = 0 \quad F_1 = 1 \quad F_n = F_{n-1} + F_{n-2} \]

**RecFibo(n):**

if \( n = 0 \)
    return 0
else if \( n = 1 \)
    return 1
else
    return RecFibo(n - 1) + RecFibo(n - 2)

\[
T(n) = T(n-1) + T(n-2) + 1
= 2F_{n+1} - 1
= \Theta(\phi^n) \quad \phi = \frac{\sqrt{5} + 1}{2} \approx 1.62
\]
Memoization: remember results of subproblems in case we use them again

F[0...J]: global memoization array

\[
\text{MemFibo}(n): \\
\text{if } n = 0 \\
\quad \text{return } 0 \\
\text{else if } n = 1 \\
\quad \text{return } 1 \\
\text{else} \\
\quad \text{if } F[n] \text{ is undefined} \\
\quad \quad F[n] \leftarrow \text{MemFibo}(n - 1) + \text{MemFibo}(n - 2) \\
\quad \text{return } F[n]
\]

O(n) space

Time?

Observation: when we compute \( F[i] \), we already computed \( F[i-1] \)
we're computing them in increasing order!

\[
\text{ITERFIBO}(n):
F[0] \leftarrow 0
F[1] \leftarrow 1
\text{for } i \leftarrow 2 \text{ to } n
\quad F[i] \leftarrow F[i-1] + F[i-2]
\text{return } F[n]
\]

Doug's time!

dynamic programming

\[
\text{ITERFIBO2}(n):
prev \leftarrow 1
curr \leftarrow 0
\text{for } i \leftarrow 1 \text{ to } n
\quad \text{next } \leftarrow \text{curr} + \text{prev}
\quad prev \leftarrow \text{curr}
\quad curr \leftarrow \text{next}
\text{return } curr
\]
Optimization problem:
many different valid/feasible solutions but...
each solution has a value
return one of maximum
minimum value

Rod Cutting:
Given a non-negative integer \( n \) and an array of integers \( p[1..m] \).

\[ \text{n units of steel rod} \]
Want to cut rod into integer length pieces for resale. Sale a pieces of length $\ell$ to get $P(\ell)$ USD.

Goal: Want a list of positive integers $i_1, i_2, \ldots, i_k$

\[ \sum_{j=1}^{k} i_j = n \]

maximizing \[ \sum_{j=1}^{k} P(i_j) \].
Today: just focus on max value (revenue)

We're making a sequence of decisions.

Let's focus on the first one & let recursion tell us the consequences, so we know what first choice is best.
What is length of first piece?

Suppose we choose length $j$...

left with rod of length $n-j$...

\( \text{CutRod}(i) = \max \text{ revenue from cutting a rod of length } i \in n. \)

\[
\text{CutRod}(i) = PC_j + \text{CutRod}(i-j) \quad \text{if } j \text{ is first choice}
\]
But which $j$?

$$\text{Cut Rod}(i) = \begin{cases} 
\max \{ P[j] + \text{Cut Rod}(i-j) \} & \text{if } i > 0 \\
0 & \text{otherwise}
\end{cases}$$

$\text{Cut Rod}(n)$ is the optimal value we want!
Backtracking: try each option to make exactly one decision, using recursion to learn about qualities/sequences of the decision

Optimal substructure: optimal solution to instance incorporates optimal solutions to subproblems
Use an array $\text{CutRod}[0..n]$.

So fill array in increasing order.

RodCutting($n, P[1..n]$):

1. $\text{CutRod}[0] \leftarrow 0$
2. for $i \leftarrow 1$ to $n$
   1. $R[i] \leftarrow 0$
   2. for $j \leftarrow 1$ to $i$
      1. if $P[j] + R[i-j] > R[i]$
         1. $R[i] \leftarrow P[j] + R[i-j]$
3. return $R[n]$
Dynamic programming is not about filling in tables. It’s about smart recursion!

See Erickson 3.4.

1) Formulate problem recursively.
   a) Say in English what the recursive subproblems solve.
   b) Say what parameters give solution to original problem.
6) Give recursive solution
(with base cases)

2) Build solutions from bottom up,
   a) identify subproblems
      what are possible parameter values? (0 ≤ i ≤ n for CutRod)
   b) choose a data structure
   c) identify dependencies.
   d) find an evaluation order.
e) Analyze space and running time.

- Space: size of structure
- Time: (usually) # subproblems x time per subproblem

f) Write down the algorithm (for loops?)