Recursion
( A telescope for Computer Scientists )

→ Intro.

Many times it's hard to define an object/relation explicitly.
But it may be easier to define an object/relation in terms of itself (but at a smaller scale).

Ex. $1, 2, 4, 8, 16 \ldots \rightarrow 2^n \rightarrow a_n = 2^n, n=0, 1, \ldots$
or $\begin{cases} a_n = 2a_{n-1} \\ a_0 = 1 \end{cases}$, this provides additional info of the structure of the series.

1. Recursively defined functions.

To define a function with the set of non-negative integers as its domain (eg. $f(n)$),

1) Specify $f(0)$ (boundary condition).
2) Specify a rule for finding its value at $n$ from its value(s) at smaller integer(s) $k < n$.

→ This is a recursive or inductive definition.

Ex. Give an inductive definition of the factorial function $F(n) = n!$

Q. What do we need here? (two things)

1) $F(0) = 1$

2) $F(n) = n F(n-1)$

$= n \cdot (n-1) F(n-2)$

$= n \cdot (n-1) (n-2) F(n-3)$

$= \ldots$
Ex. Give a recursive def of \( \sum_{k=0}^{n} A_k \)

\[ F(n) = \sum_{k=0}^{n} A_k. \]
1) \( F(0) = \sum_{k=0}^{0} A_k = A_0. \)
2) \( F(n) = \sum_{k=0}^{n-1} A_k + A_n = F(n-1) + A_n. \)

Ex. The Fibonacci \#s, \( f_0, f_1, f_2, \ldots \)

\[ \begin{align*}
\begin{cases}
f_0 = 0, & f_1 = 1, \\
f_n = f_{n-1} + f_{n-2}.
\end{cases}
\end{align*} \]

\[ \begin{align*}
f_2 &= f_0 + f_1 = 1 \\
f_3 &= f_1 + f_2 = 1 + 1 = 2 \\
f_4 &= f_3 + f_2 = 1 + 2 = 3 \\
f_5 &= 5 \\
f_6 &= 8.
\end{align*} \]

In recursion

The Fibonacci \#s is very useful for proving many properties of these \#s. Here is an example.

Ex. Prove \( f_n > \alpha^{n-2} \) where \( \alpha = \frac{(1+\sqrt{5})}{2} \), \( n \geq 3 \).

Q: Which proof technique?
- We use induction by definition -> Induction

Let \( P(n) \) be the statement that \( f_n > \alpha^{n-2} \)

To prove: \( P(n) \) is true whenever \( n \geq 3 \).

To start:
1) \( n = 3 \).
\[ \alpha^{n-2} = \alpha = \frac{1 + \sqrt{5}}{2} < 2 \quad (\therefore \sqrt{5} < 3) \]

\[ n = 4 \]

\[ \alpha^{n-2} = \alpha^2 = \frac{(1 + \sqrt{5})^2}{2} = \frac{(3 + \sqrt{5})}{2} < 3 = F_4 \]

2) Inductive hypothesis.

For all \( 0 \leq k < n \), \( f_k > \alpha^{k-2} \)

\( \text{i.e.} \quad f_k > \alpha^{k-2} \)

3) we now need to prove \( f_n > \alpha^{n-2} \)

Based on 1)

\[ \alpha^2 = (3 + \sqrt{5})^2 = 1 + \alpha \]

\[ f_{n-2} \alpha^{n-2} = \alpha^2 \cdot \alpha^{n-4} = (\alpha + 1) \cdot \alpha^{n-4} = \alpha^{n-3} + \alpha^{n-4} \]

\[ \therefore \]

If \( n \geq 3 \), \( f_{n-1} > \alpha^{n-3}, \ f_{n-2} > \alpha^{n-4} \)

\[ \therefore f_n = f_{n-1} + f_{n-2} > \alpha^{n-3} + \alpha^{n-4} = \alpha^{n-2} \]

\( P(n) \) is true.

\[ \begin{align*}
\text{Ex.} & \\
2 & \text{Recursive algo. (program)}
\end{align*} \]

Def. 'Any algo calls itself (but with a smaller input) is a recursive algo.'

\[ \begin{align*}
\text{Ex.} & \quad \text{Compute } A^n \\
F(n) &= a^n = a \cdot F(n-1) & \text{if } n=0, \ F(0)=1
\end{align*} \]
procedure \texttt{POWER1}(a, n)
    if \( n = 0 \) then return 1
    else return \((ax \text{ power } a, n-1)\)

Q: \# of multiplications? \(O(n)\)

Ex. Given a list of integers, find the location of a particular \# in \( S \). — linear search in recursion

Sol. Given \( a_1, a_2 \ldots, a_n \) and \( x \) to be found in \( S \).
At \( i \)-th step, we compare \( a_i \) and \( x \).

Procedure \texttt{lsearch}(i, j, x) \rightarrow we start \texttt{lsearch}(1, n, x)
    if \( a_i = x \) then
        return \( i \)
    else if \( i = j \) then
        return -1 (or 0)
    else return \texttt{lsearch}(i+1, j, x)

Q: what is the \# of operations needed (in the worst case)?
A: \( n \) comparisons \(\rightarrow O(n)\)
\[
T(n) = T(n-1) + 1.
\]

Q: can we do better than \( O(n) \) ?
We can if \( S \) is sorted.

Ex. Given a list of sorted integers \( S \), find the location of \( x \).

Sol: Procedure \texttt{sssearch}(i, j, x)\hspace{1cm} a_1 \leq a_2 \leq a_3 \leq \cdots \leq a_n
    \[
    k = \frac{(i+j)}{2};
    \]
    if \( a_k = x \) then
        return \( k \).
Procedure $Ssearch (n, j, x) \rightarrow Ssearch (L, n, x)$

if $i > j$ then
    return $-1$ (or 0)
else $k = (i+j)/2$
    if $a_k = x$ then
        return $k$
    else if $a_k > x$ then
        return $Ssearch (i, k-1, x)$
    else return $Ssearch (k+1, j, x)$

Q: What is the complexity (# of comparisons)?

\[
\begin{align*}
T(n) & = 1 + T\left(\lceil n/2 \rceil \right) \\
T(1) & = 1
\end{align*}
\] $\Rightarrow O(\log_2 n)$

Ex: Computing $a^n$ again. $\Rightarrow$ can we do better than $O(n)$

s01: What is the deal: binary.

if $n = 2^k$, $a^n = a^k \cdot a^k$
if $n = 2^k+1$, $a^n = a \cdot a^k \cdot a^k$

Procedure $Power2 (a, n)$
if $n = 0$, return 1
else if $(n \mod 2) = 0$ then
    return $Power2 (a, n/2) \cdot Power2 (a, n/2)$
el return $a \cdot Power2 (a, n/2) \cdot Power2 (a, n/2)$

Q: Complexity?
No saying at all!
Trick: you have to save the work!

(Improved) Procedure \textit{Power 3}(a, n)

\begin{align*}
\text{if } n &= 0 \text{ then } \\
& \quad \text{return } 1 \\
\text{else if } n &= 1 \text{ then } \\
& \quad \text{return } a \\
\text{else if } (n \mod 2) &= 0 \text{ then } \\
& \quad [b = \text{Power 3}(a, n/2); \\
& \quad \quad \text{return } b \cdot b ] \\
\text{else } & [b = \text{Power 3}(a, (n-1)/2); \\
& \quad \quad \text{return } a \cdot b \cdot b ] \\
\end{align*}

Q complexity?
\begin{align*}
\sum T(n) &= 2 + T(\lceil n/2 \rceil) \Rightarrow \mathcal{O}(\log_2 n) \\
T(2) &= 1, \quad T(1) = 0
\end{align*}

2\' Recursion vs. Iteration.
In general, those linear operations can be put into an iterative procedure, but truly recursive calls may not be.
3. Recurrence Relation.

We have been talking about RR all along —- RR captures some intrinsic problem structures.

Def. A recurrence relation (RR) for a seq. \( \{a_i\} \) is an equation that expresses \( an \) in terms of one or more of the previous terms of the seq. A seq. is called a solution of a RR if its terms satisfy the RR.

\[ a_0 \text{ is the initial condition, } a_0 + RR \Rightarrow \text{ determines the seq. } \{a_i\}, \ i = 0, 1, \ldots \]

3.1 Modeling with RR.

A useful tool.

Ex. A savings account pays 11% interest per year, with interest compounded annually. How much will be in the account after 30 years if deposit $1,000?

Sol. Where to start? \( \Rightarrow \) construct \( \{a_n\}, \ n = 0, 1, \ldots, 30 \)

\[ a_0 = 1,000 \]
\[ a_n = a_{n-1} + 0.11 a_{n-1} = 1.11 a_{n-1} \]
\[ a_1 = 1.11 a_0 \]
\[ a_2 = 1.11 a_1 = (1.11)^2 a_0 \]
\[ \text{and } \]

\[ a_n = (1.11)^n a_0 \Leftrightarrow \text{RR we want!} \]

\[ n = 30, \quad A_{30} = (1.11)^{30} a_0 = (1.11)^{30} \times 1,000 = \$ 22,892.30 \]

A nearly 23 fold increase!
Ex. The Tower of Hanoi.

\[ n \text{ disks} \]

peg 1  peg 2  peg 3

Goal: move one disk at a time
Restriction: big disk cannot be on top of a smaller one.

Q: how? What's the sequence? What's the # of moves?

# of moves:

\[ a_1 = 1 \] look for \( \{ a_n \} \)

The idea is --- move the bottom one

\[ a_n = a_{n-1} + 1 + a_{n-1} \]

\[ \text{move to peg 2} \quad \text{move (n-1) from peg 2 to peg 3} \]

\[ a_n = 2a_{n-1} + 1 \]

\[ = 2(2a_{n-2} + 1) + 1 = 2^2 a_{n-2} + 2 + 1 \]

\[ = 2^2(2a_{n-3} + 1) + 2 + 1 = 2^3 a_{n-3} + 2^2 + 2 + 1 \]

\[ \vdots \]

\[ = 2^{n-1} a_1 + 2^{n-2} + 2^{n-3} + \cdots + 2 + 1 \]

\[ = 2^{n-1} - 2^0 = 2^n - 1 \]

This was from an ancient myth:

A group of monks have to move 64 gold disks from one peg to another, based on the rule. They assume the move one disk in 1 second, how long to finish?

\[ 2^{64} - 1 = 18,446,744,073,709,551,615 \]