# Complexity of recursive algorithms

def: an algorithm that calls itself, either directly or indirectly

$f() \rightarrow g()$

$n! = n \times n-1 \times n-2 \times \cdots \times 2$

$= \prod_{i=2}^{n} i$

$= n \cdot (n-1)!$

$1! = 1$

```
fact(n) {
    if (n<2) return 1
    return n * fact(n-1)
}
```

```
fact(n) {
    res = 1
    for i in 2..n
        res = res * i
    return res
}
```

param: $n$

unit op: $*$

$n = i = n$

$$\begin{cases} C(0) = 0 \\ C(1) = 0 \\ C(n) = C(n-1) + 1 \end{cases}$$

step 1: find the equation(s) of recurrence

step 2: solve the system

$= (C(n-2)+1)+1$

$= C(n-3)+3 = C(n-i)+i$

$= C(1) + n - 1 = \Theta(n)$

general process:
0) choose param and unit op(s)
1) find the stop condition(s)
2) find the recurrence equations

$$C(n) = f(\underset{i<n}{C(i)})$$

3) solve these equations

$x^n$ : pow(x, n) {
if (n == 0) return 1
return x * pow(x, n-1)
}

indian exponenciation

```
pow(x, n) {
    if (n == 0) return 1
    if (n == 1) return x
    if (n % 2 == 0) return sqr(pow(x, n/2))
    return x * sqr(pow(x, (n-1)/2))
}
```

$$\begin{cases} C(0) = C(1) = 0 \\ C(2p) = C(p) + 1 \\ C(2p+1) = C(p) + 2 \end{cases}$$

$x^0 = 1$, $x^1 = x$

$x^{2p} = (x^p)^2$

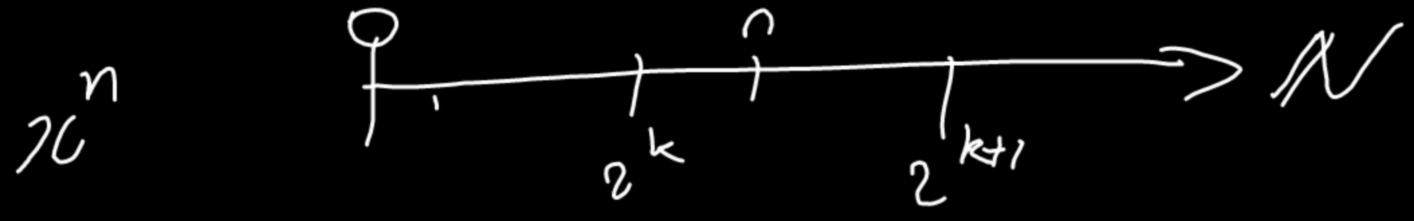
$x^{2p+1} = (x^p)^2 * x$

if $n = 2^k$:

$C(2^k) = C(2^{k-1}) + 1$
$= C(2^{k-2}) + 2$
$= C(2^{k-3}) + 3$
$= C(2^{k-i}) + i$

$i = k-1 \Rightarrow$
$C(2^k) = C(1) + k - 1$
$= \log n - 1 = \Theta(\log n)$

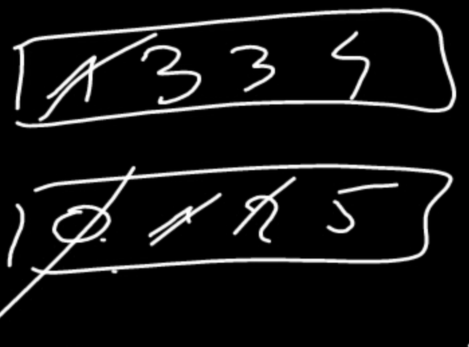
$n = 2^k \Rightarrow k = \log n$




$\exists k / 2^k \leq n \leq 2^{k+1}$

$$C(n) \leq C(2^{k+1}) = k \leq \log n + 1$$

# mergesort:

0 1 1 2 3 3 4 5

merge(A, B) is $\Theta(n)$, $n = \max(\text{size}(A), \text{size}(B))$

```
mergesort(a, l, r) {
    if (r > l)
        merge(mergesort(a, l, (l+r)/2), mergesort(a, (l+r)/2 + 1, r))
}
```

mergesort(a, 0, size(a)-1)

param: size of array: r - l + 1

unitop: comparison

$$\begin{cases} C(0) = C(1) = 0 \\ C(n) = 2C\left(\frac{n}{2}\right) + n \end{cases}$$

$C(0) = C(1) = 0$

$$C(n) = 2C\left(\frac{n}{2}\right) + n$$

$n = 2^k$

$= kn$

$= n \log(n)$

$= 2\left(2C\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$

$= 4C\left(\frac{n}{4}\right) + 2n$

$= 4\left(2\left(\left(\frac{n}{8}\right) + \frac{n}{4}\right)\right) + 2n$

$= 8\left(\left(\frac{n}{8}\right) + 3n\right)$

$\vdots$

$= 2^i C\left(\frac{n}{2^i}\right) + in$

$= 2^k C(1) + kn$

quadratic:

$$C(n) = 4\,C\left(\frac{n}{2}\right) + n$$

$a: 4$
$b: 2$
$c: 1$

$$C(n) = n^{\log_2 4} = n^2$$

---

**Master theorem:** $C(n) = a\,C\left(\frac{n}{b}\right) + n^c$ (admitted)

if $a < b^c$ : $C(n) = \Theta(n^c)$

if $a = b^c$ : $C(n) = \Theta(n^c \log_b n)$

if $a > b^c$ : $C(n) = n^{\log_b a}$

mergesort: $a = 2$, $b = 2$, $c = 1$ $\rightarrow$ $C(n) = \Theta(n \log n)$

$\Theta(n^3)$: $C(n) = 8\,C\left(\frac{n}{2}\right) + n^2$

or

$C(n) = C\left(\frac{n}{2}\right) + n^3$

Fibonacci:

$$\begin{cases} f_0 = f_1 = 1 \\ f_n = f_{n-1} + f_{n-2} \end{cases}$$

```
fibo(n) {
  if (n<2) return 1
  return fibo(n-1)+fibo(n-2)
}
```

param n
unit op +

$$\begin{cases} C(0) = C(1) = 0 \\ C(n) = C(n-1) + C(n-2) + 1 \end{cases}$$

(+1 . +1)

$$D(n) = C(n) + 1$$

$$\begin{cases} D(0) = D(1) = 1 \\ D(n) = D(n-1) + D(n-2) \end{cases}$$

guess: $D(n) = \ell^n$

$$\ell^n = \ell^{n-1} + \ell^{n-2}$$

$$\ell^2 = \ell + 1$$

$$\ell^2 - \ell - 1 = 0$$

$$\Delta = 1 + 4 + 5$$

$$\ell = \frac{1 \pm \sqrt{5}}{2}$$

$$\frac{1+\sqrt{5}}{2}$$

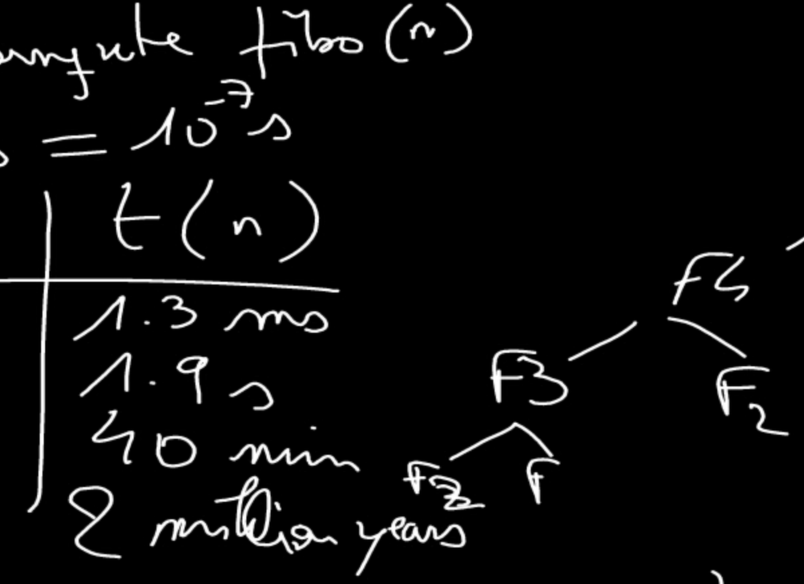
$$C(n) = \left(\frac{1+\sqrt{5}}{2}\right)^n$$

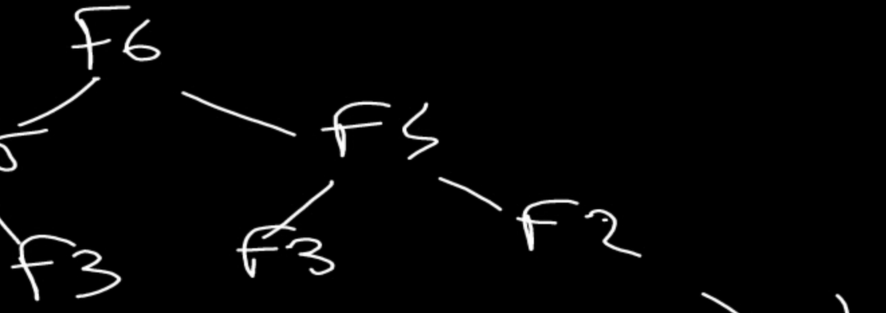
$$\begin{aligned} D(n) &= 2\,D(n-1) \\ &= 4\,D(n-2) \\ &= 8\,D(n-3) \\ &= 2^n\,D(n-n) \\ &= 2^n \end{aligned}$$

time to compute fibo(n)

+ : 100 ns $= 10^{-7}$ s

| n | t(n) |
|---|---|
| 20 | 1.3 ms |
| 30 | 1.9 s |
| 50 | 40 min |
| 100 | 2 million years |




```
fibo3(n, f1, f0){
    if (n < 2) return f1
    return fibo3(n-1, f1+f0, f1)
}
```

```
fibo(n){ return fibo3(n, 1, 1)}
```

| n | fibo(n) |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 5 |
| 5 | 8 |
| 6 | 13 |
| 7 | 21 |

fibo(7) = fibo3(7, 1, 1)
= fibo3(6, 2, 1)
= fibo3(5, 3, 2)
= fibo3(4, 5, 3)
= (3, 8, 5)
= (2, 13, 8)
= (1, 21, 13) = 21

$C(1) = C(0) = 0$

$C(n) = C(n-1) + 1$

$= (C(n-2)+1)+1$

$= C(n-2)+2$

$= C(n-3)+3$

$\vdots$

$= C(n-i)+i$

$i = n-1$ $= C(1)+n-1$

$= \Theta(n)$

# Abstract Data Types (ADTs)

ADT def: (T, U, O, P, A)

T: type to be defined
U: already defined types that are part of the def. of T
O: operations allowed on T
P: preconditions for some operations
A: Axioms defining the semantics of operations

## Example: Booleans

ADT Boolean

Operations

T : ⟶ Boolean

F : ⟶ Boolean

and : Boolean × Boolean ⟶ Boolean

not : Boolean ⟶ Boolean

Axioms:

A1 : and(T, x) ≡ x

A2 : and(F, x) ≡ F

A3 : not(T) ≡ F

A4 : not(F) ≡ T

and(not(T), and(not(F), F))

A3: and(F, and(not(F), F))

A2: F

def: a internal of. is an operation whose return type is the defined type T

constructors = minimum set of constructors needed to generate any value of the type

Prop: the set of axioms must describe the result of applying any non-constructor to any constructor

ADT Natural

succ(x)
↳ Sx

Use Boolean

Operations

* $0 :$ → Natural
* succ : Natural → Natural
* $\_=\_$ : Natural × Natural → Boolean
* $\_+\_$ : Natural × Natural → Natural

$1 = 3$

$=(x, y)$

$x = y$

$S0 = SSS0$

$A_4$ $0 = SS0$

$A_2$ F

$A_1$ $0 = 0 \equiv T$

$A_2$ $0 = Sx \equiv F$

$A_3$ $Sx = 0 \equiv F$

$A_4$ $Sx = Sy \equiv x = y$

$2 = ?$ ?

$SS0 = SS0$

$A_4$ $S0 = S0$

$A_4$ $0 = 0$

$A_1$ T

$A_5 \quad 0 + x \equiv x$

$A_6 \quad x + 0 \equiv x$

$A_7 \quad sx + y \equiv x + sy$

$3 + 2$

$sss0 + ss0$

$A_7 \quad ss0 + sss0$

$A_7 \quad s0 + ssss0$

$A_7 \quad 0 + sssss0$

$A_2 \quad sssss0$

A Counter has a natural value (0 at start).
It can only be incremented or decremented by 1 at a time.
Decrementing a null Counter has no effect.

ADT Counter
Use Natural
Operations
* new : ⟶ Counter
* incr : Counter ⟶ Counter
decr : Counter ⟶ Counter
value : Counter ⟶ Natural

Axioms
$A_1$ value(new) ≡ 0
$A_2$ value(incr(c)) ≡ value(c) + 1
$A_3$ decr(new) ≡ new
$A_4$ decr(incr(c)) ≡ c

value(decr(incr(incr(new))))
($c$ = incr(new))
$A_3$ value(incr(new))
$A_2$ value(new) + 1
$A_1$ 0 + 1 ≡ 1