Complexity for recursive algorithms

def: recursive algorithm: calls itself, either directly or indirectly

$$n! = \prod_{i=1}^{n} i$$

or

$$\begin{cases} n! = n\,(n-1)! \\ 0! = 1 \end{cases}$$

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

param: n
unit op: *

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

step 0: choose params / unit ops
step 1: find recurrence equation(s)
step 2: solve the system

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

$$\begin{aligned} &= (C(n-2)+1)+1 \\ &= C(n-2)+2 \\ &= (C(n-3)+1)+2 \\ &= C(n-3)+3 \\ &\vdots \\ &= C(n-i)+i \end{aligned}$$

$i = n-1$

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

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

Indian exponentiation

$$x^n = \begin{cases} 1 & \text{for } n=0 \\ x & \\ (x^p)^2 & p \\ (x^p)^2 \cdot x & \end{cases}$$

```
sqr(n) {
    return n*n
}
```

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

param: n
unit : *

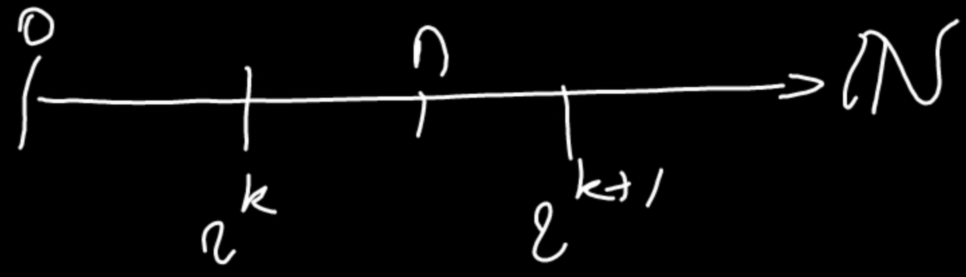
$$C(0) = 0$$
$$C(2p) = C(p) + 1$$
$$C(2p+1) = C(p) + 2$$

$n = 2^k$ $\quad C(n) = C(2^k) = C(2^{k-1}) + 1$

$i = k$ $\quad = C(2^0) + k \quad = C(2^{k-2}) + 2$

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

$\quad = 2 + k = 2 + \log n = \Theta(\log n)$



$$k < \log n < k+1$$

$$k+1 < \log n + 1 < k+2$$

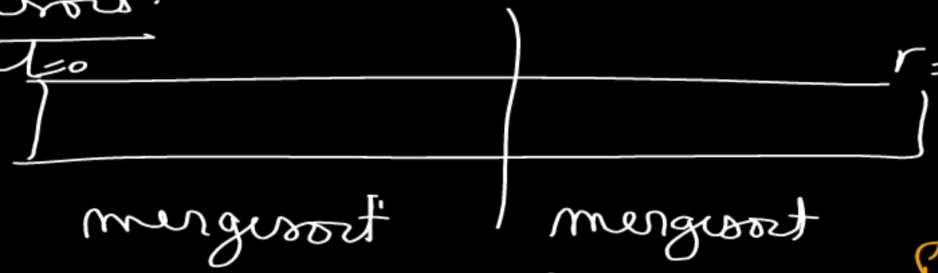
$$C(n) \leq C(2^{k+1})$$

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

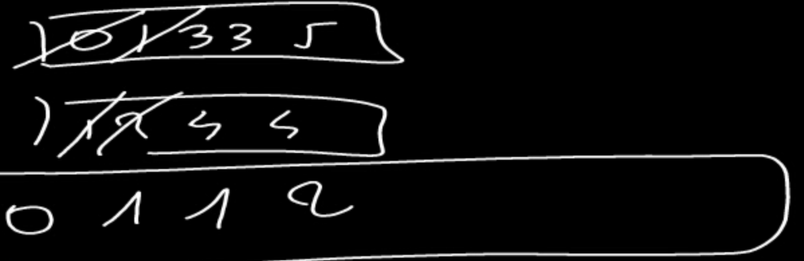
$$= k+1$$

$$\leq \log n + 1 = \mathcal{O}(\log n)$$

mergesort:



param: r-l+1
unit op: <



C(merge): ④(n

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

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

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

$n = 2^k$

$C(n) = C(2^k) = 2\left(C\left(2^{k-1}\right) + 2^k\right)$

$= 2\left[2\left(C\left(2^{k-2}\right) + 2^{k-1}\right)\right] + 2^k$

$= 2^2 C\left(2^{k-2}\right) + 2 \cdot 2^k$

$= 2^2\left[2C\left(2^{k-3}\right) + 2^{k-2}\right] + 2 \cdot 2^k$

$= 2^3 C\left(2^{k-3}\right) + 3 \cdot 2^k$

$= 2^i C\left(2^{k-i}\right) + i\,2^k$

$i = k$

$= 2^k C(1) + k\,2^k$

$= 2^k + k\,2^k = ④(n \log n)$

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

$$C(1) = 0$$

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

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

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

$$= 4^2\left[4\,C\left(\frac{n}{2^3}\right) + \frac{n}{4}\right] + 3n$$

$$= 4^3\,C\left(\frac{n}{2^3}\right) + 7n$$

$$\vdots$$

$$= 4^i\,C\left(\frac{n}{2^i}\right) + \left(2^i - 1\right)n$$

$n = 2^i$

$$= n^2\,C(1) + n^2 - n$$

$$= \Theta(n^2)$$

Master theorem for "divide & conquer" algorithms

$$C(n) = a\,C\left(\frac{n}{b}\right) + n^c$$

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

if $a = b^c$, $C(n) = \Theta(n^c \log n)$

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

mergesort: $a = b = 2$, $c = 1$

$$C(n) = \Theta(n \log n)$$

$a = 4$, $b = 2$, $c = 1$

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

Fibonacci: param: n
unit op: $+$

$$f_0 = f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

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

n	f_n
0	1
1	1
2	2
3	3
4	5
5	8
6	13
7	21
⋮	

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

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

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

$D(0) = D(1) = 1$

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

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

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

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

$= 2\left(2C(n-2) + 1\right) + 1$

$= 2^2 C(n-2) + 3$

$= 2^2\left(2C(n-3) + 1\right) + 3$

$= 2^3 C(n-4) + 7$

$\vdots$

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

$i = n - 2$

$2^{n-2} C(1) + 2^{n-2} - 1$

$= \Theta(2^n)$

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

$D(n) = \Theta(\ell^n)$?

hyp : $D_n = \Theta(\ell^n)$

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

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

$$\boxed{\ell^2 - \ell - 1 = 0}$$

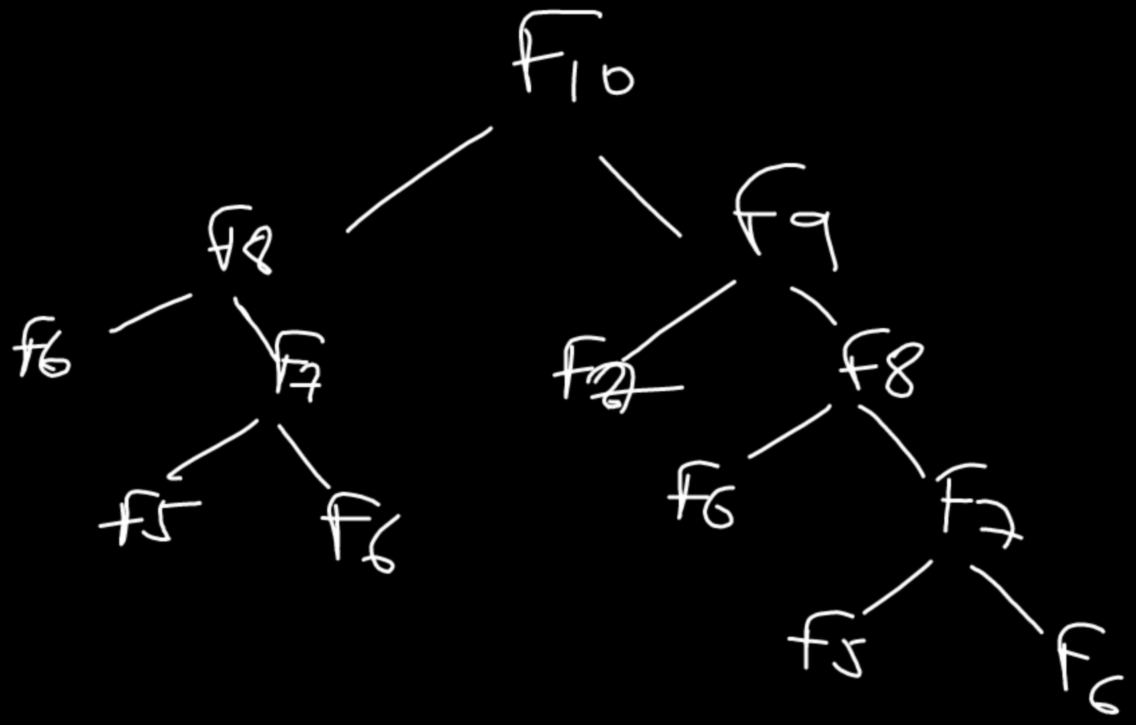
$$\Delta = 1 + 4 = 5$$

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

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

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

$t(+) = 100$ ns $= 10^{-7}$ s


F10
F8
F9
F6
F7
F7
F8
F5
F6
F6
F7
F5
F6

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

$fibo(n) \Rightarrow fibo3(n, 1, 1)$

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

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

$= C(n-2) + 2$

$\vdots$

$= C(n-i) + i$

$= \cancel{C(1)} + n - 1$

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

$fibo(7) = fibo3(7, 1, 1)$

$= fibo3(6, 2, 1)$

$= fibo3(5, 3, 2)$

$= fibo3(4, 5, 3)$

$= fibo3(3, 8, 5)$

$= fibo3(2, 13, 8)$

$= fibo3(1, 21, 13) = 21$

Abstract Data Types (ADTs)

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

T: type being defined

U: set of used (already defined) types

O: set of legal operations on the type

P: set of preconditions on some operations

A: set of axioms describing the semantics of the operations

ex: Booleans:

ADT Boolean

Operations

* T : → Boolean
* f : → Boolean

and : Boolean × Boolean → Boolean

not : Boolean → Boolean

Axioms

A_1: and (T, x) ≡ x

A_2: and (F, x) ≡ F

A_3: not (T) ≡ F

A_4: not (F) ≡ T

and (not (and (T, not (f)), not (T)))

x = not (f)

A1: and (not (not (f)), not (T))

A3: and (not (not (f)), not (T))

A4: and (not (T), F), not (T))

A4 and (F, F)

A2 F

Set of constructors: minimum subset of operations needed to generate any value of the type

Prop: a complete set of axioms must describe the result of applying all non-constructor operations to all constructors

ADT Natural

Uses Boolean

Operations

$\_==\_ : Natural \times Natural \longrightarrow Boolean$

$* 0 : \longrightarrow Natural$

$\_+\_ : Natural \times Natural \longrightarrow Natural$

$* s\_ : Natural \longrightarrow Natural$

Axioms

$A_1 \quad 0 + x \equiv x$

$A_2 \quad x + 0 \equiv x$

$A_3 \quad sx + y \equiv x + sy$

$A_4 \quad 0 = 0 \equiv T$

$A_5 \quad 0 = sx \equiv F$

$A_6 \quad sx = 0 \equiv F$

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

2 + 3 ?

$ss0 + sss0$

$A_3 \quad s0 + ssss0$

$A_3 \quad 0 + sssss0$

$A_1 \quad sssss0$

2 = 2 ?

$ss0 = ss0$

$A_7 \quad s0 = s0$

$A_7 \quad 0 = 0$

$A_4 \quad T$

1 = 2 ?

$s0 = ss0$

$0 = s0$

$A_5 : F$

A Counter has a natural value (0 at start). new

It can only be incremented / decremented by 1 at a time.

Decrementing a null Counter has no effect

ADT Counter

Use Natural

Operations

* new: ⟶ Counter
  value: Counter ⟶ Natural
* incr: Counter ⟶ Counter
  decr: Counter ⟶ Counter

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

tiny.one/algo2

value(decr(incr(incr(new)))) c

A_4 value(incr(new))

A_2 value(new)+1

A_1 0+1 ≡ 1