## Algoritmo (per numeri di fibonacci)

$$F_n = \begin{cases} 1 & n = 1, 2 \\ F_{n-1} + F_{n-2} & n \geq 3 \end{cases}$$

## formula di Binet

sezione aurea

$$x^2 = x + 1 \longrightarrow x^2 - x - 1 = 0 \quad \begin{cases} \Phi = \dfrac{1+\sqrt{5}}{2} \simeq 1.6\ldots \\ \hat{\Phi} = \dfrac{1-\sqrt{5}}{2} \simeq -0.6\ldots \end{cases}$$

complemento della sezione aurea

$$\forall n \geq 1 : F_n = 1/\sqrt{5} \left( \Phi^n - \hat{\Phi}^n \right)$$

## Dimostrazione tramite passo induttivo

$$n = 1 \Rightarrow F_n = 1/\sqrt{5} \left( \frac{1+\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2} \right) = 1/\sqrt{5} \left( \frac{2\sqrt{5}}{2} \right) = 1 \checkmark$$

$$n = 2 \Rightarrow F_n = 1/\sqrt{5} \left( \left( \frac{1+\sqrt{5}}{2} \right)^2 - \left( \frac{1-\sqrt{5}}{2} \right)^2 \right)$$

$$\hookrightarrow 1/\sqrt{5} \left( \frac{1+2\sqrt{5}+5}{4} - \frac{1-2\sqrt{5}+5}{4} \right)$$

$$\hookrightarrow 1/\sqrt{5} \left( \frac{4\sqrt{5}}{4} \right) = 1 \ V$$

per $n \geq 3$ l'ipotesi dice che la proprietà vale fino ad $n-1$ quindi se:

$$F_n = 1/\sqrt{5}\left(\Phi^n - \hat{\Phi}^n\right) \quad e \quad F_n = F_{n-1} + F_{n-2}$$

per ipotesi induttiva allora

$$F_n = 1/\sqrt{5}\left(\Phi^{n-1} - \hat{\Phi}^{n-1}\right) + 1/\sqrt{5}\left(\Phi^{n-2} - \hat{\Phi}^{n-2}\right)$$

$$= 1/\sqrt{5}\left[\left(\Phi^{n-1} + \Phi^{n-2}\right) - \left(\hat{\Phi}^{n-1} + \hat{\Phi}^{n-2}\right)\right.$$

....

$$1/\sqrt{5}\left(\Phi^n - \hat{\Phi}^n\right) \ ?$$

come ci arrivo? dimostro se è possibile che

$$\begin{cases} \Phi^n = \Phi^{n-1} + \Phi^{n-2} \\ \hat{\Phi}^n = \hat{\Phi}^{n-1} + \hat{\Phi}^{n-2} \end{cases}$$

se sono entrambe vere la dimostrazione è finita

divido per $\Phi^{n-2}$ e $\hat{\Phi}^{n-2}$

$$\begin{cases} \Phi^2 = \Phi + 1 \\ \hat{\Phi}^2 = \hat{\Phi} + 1 \end{cases}$$

$\rightarrow$ per definizione di $\Phi$, ovvero $x^2 = x+1$ la dimostrazione è verificata pseudocodice:

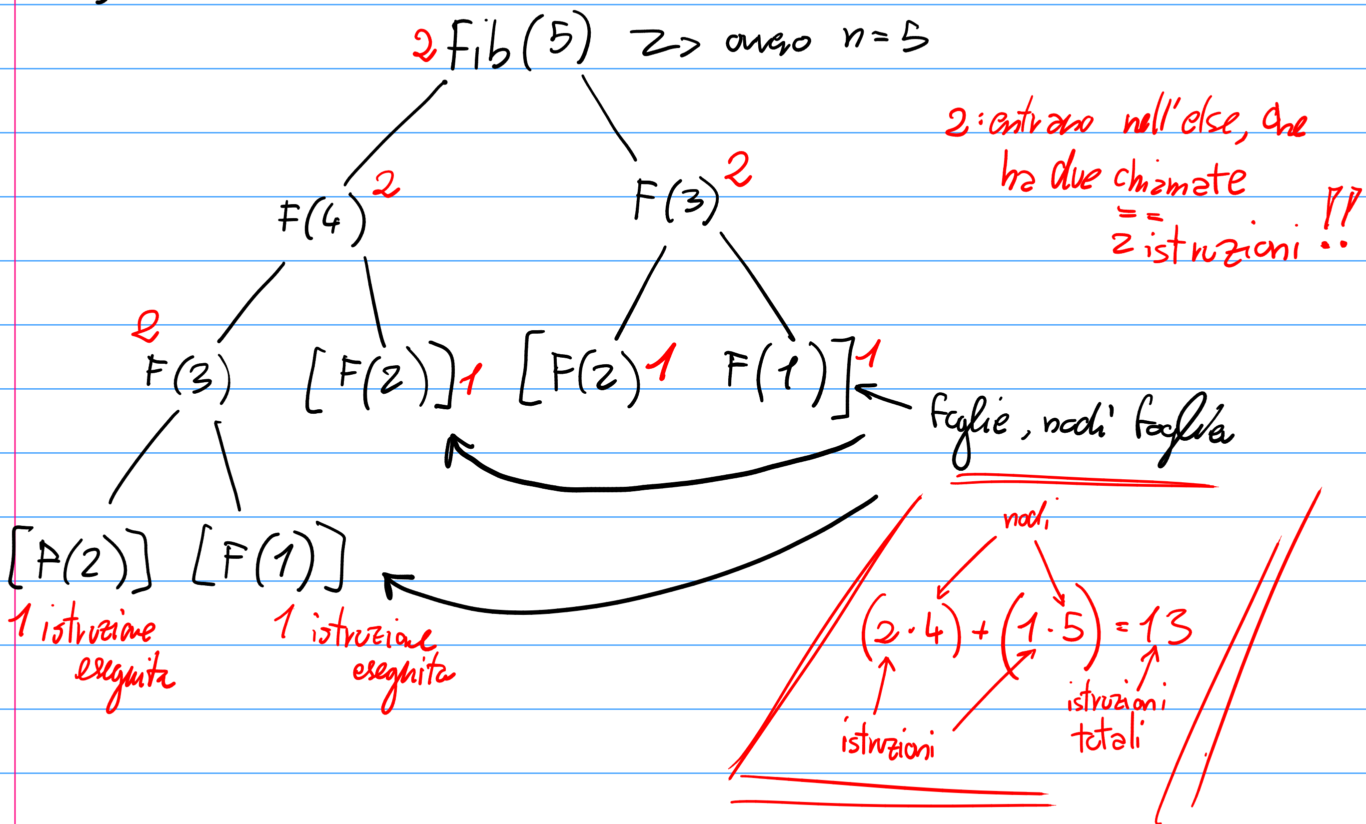
```
Fib(int n) -> int
    if n ≤ 2 then return 1;
    else return Fib(n-1) + Fib(n-2)
```

Complessità? Quante istruzioni sono eseguite?

| $n$ | $T(n)$ |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 4 |
| 4 | $2+4+1=7$ |

Albero delle ricorsioni (esempio pratico)

$n=5$



L'albero ci permette di calcolare qualunque complessità

$$T(5) = 13 \rightarrow 2 \cdot i(T_n) + f(T_n)$$

non foglie

$f$ = foglie

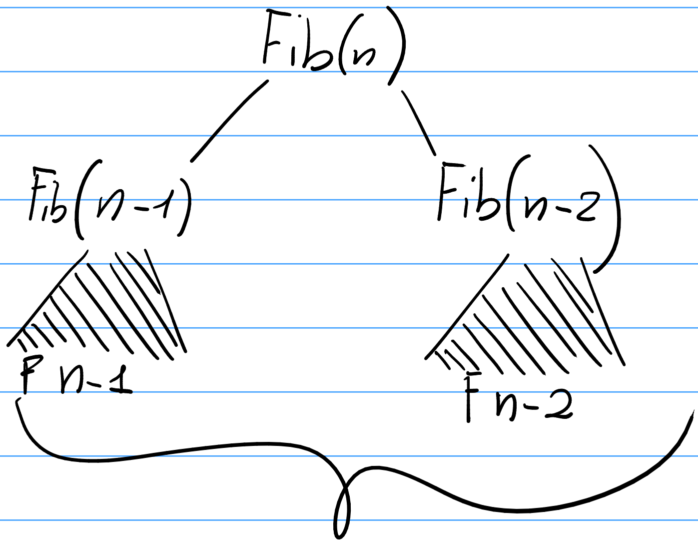
**Proprietà 1** Sia $T_n$ l'albero delle ricorsioni relativo alla chiamata Fib(n).

Allora il numero di foglie di $T_n$ è pari a $F_n$ (ennesimo numero di Fibonacci)

$$f(T_n) = F_n$$

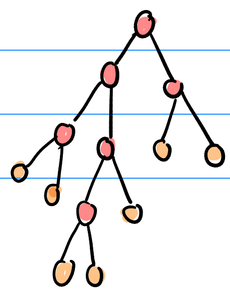
Dim: induttiva su n (meglio con il disegno

$$F_n = F_{n-1} + F_{n-2}$$



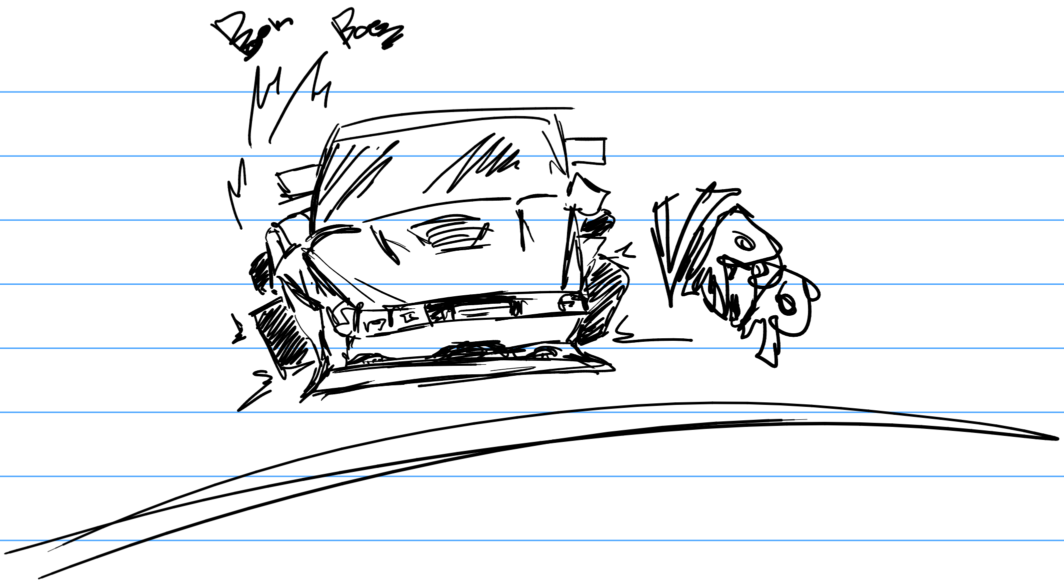

$F_n$ = iesimo numero di Fibonacci

**Proprietà 2** se ogni nodo ha esattamente due figli, allora $i(T) = f(T) - 1$



● $= i(T_n) = f(T_n) - 1$ (albero binario)

● $= F(n)$ (caso fibonacci)



Fib(int n) → int
if $n \leq 2$ then return 1;
else return Fib(n-1) + Fib(n-2)

ricorda ||| T(n) = numero di istruzioni mandate in esecuzione!

$$\begin{cases} T(n) = 2 + T(n-1) + T(n-2) & \text{quando } n \geq 3 \\ 1 & \text{quando } n = 1,2 \end{cases}$$

ricorsiva come l'algoritmo ... prof wtf

formula generale

foglie

$$\boxed{T(n) = 2 \cdot i(T_n) + f(T_n)}$$

nodi non foglie

a) $f(T_n) = F_n$

b) $i(T_n) = f(T_n) - 1$

riprendo pagine successive

quindi posso semplificare la formula generale con:

$$T(n) = 2(F_n - 1) + F_n = 3F_n - 2$$

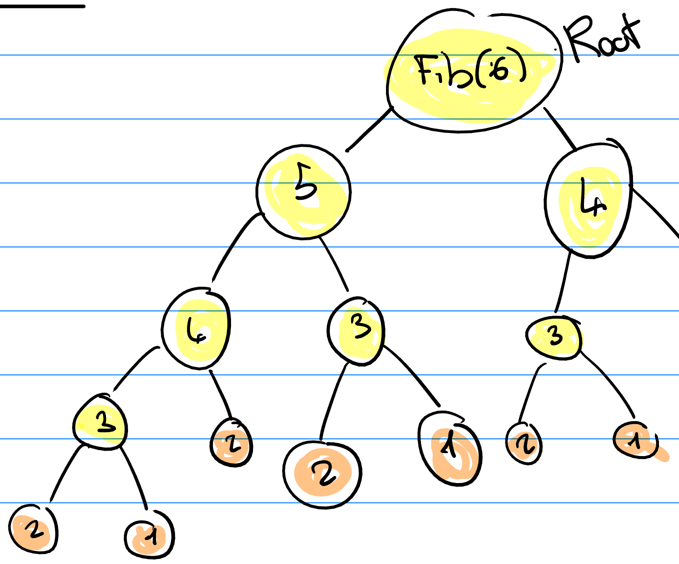
complessità cresce "come i numeri di Fibonacci"

$T(n) \simeq F_n \qquad \forall n \geq 6$

$F_n \geq 2^{n/2}$

Ind. su n (verifichiamo)

Base : $n = 6, 7$



$n \geq 8$

$$F_n = \underbrace{F_{n-1}}_{2^{(n-1)/2}} + \underbrace{F_{n-2}}_{2^{(n-2)/2}}$$

$$\geq_1 2^{(n-1)/2} + 2^{(n-2)/2}$$

$$2^{\frac{n}{2}}\left(\frac{1}{\sqrt{2}} + \frac{1}{2}\right) \geq 2^{n/2}$$

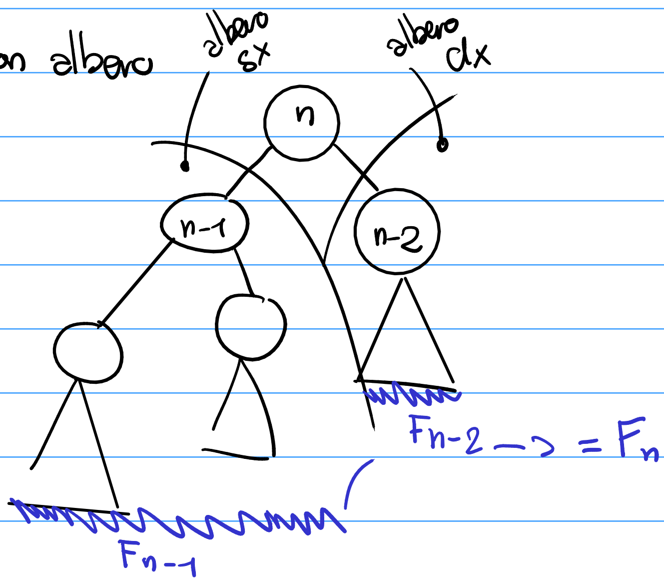
riprendo formule

$f(T_n) = F_n$ ?  numero figli $= F_n$ ?

<u>base</u> $n = 1, 2$ --> non ci sono nodi foglia lol

$n \geq 3$

↳ ipotesi con albero

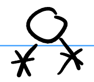



totale nodi foglia = somma numero nodi foglia sx e dx

Se T é un albero binario dove ogni vertice interno ha esattamente 2 figli:
allora $i(T) = f(T) - 1$

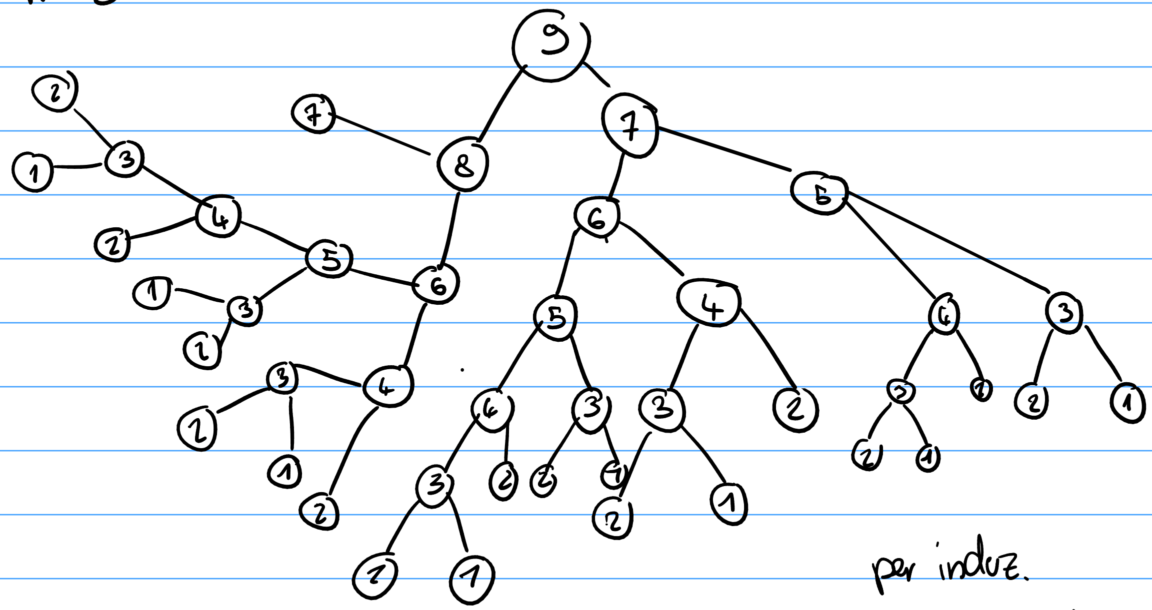
## Dimostrazione (ind sulla grandezza dell'albero)

<u>Base</u> $n = 1, 2$



non ha nodi figli!

$n = 9$



per induz.
$n = n$ nodi

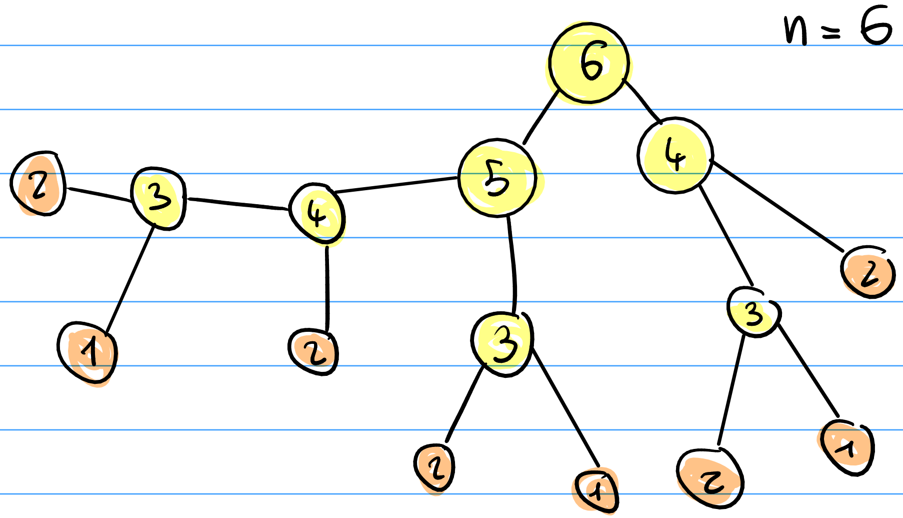
$$i(T) = f(T) - 1$$

$$i(T) = \begin{cases} i(T) = i(T') + 1 \\ f(T) = f(T') + 1 \end{cases}$$

$$i(T) = \begin{cases} i(T) = i(T') + 1 = f(T') \\ f(T) = f(T') + 1 \end{cases}$$

la domanda del secolo per algoritmi complessi: l'algoritmo finirà?

$n = 6$



Fib3(int n) —> int

1. alloca spazio per un array F di n interi
2. F[1] = F[2] = 1;
3. for i=3 to n
4. F[i] = F[i-1] + F[i-2];
5. return F[n];

$n = 3 \quad + (n-2) + (n-1)$

4 3

1,2,5

$= 2 \cdot n \longrightarrow 2$ è non considerabile $\Rightarrow n$

mancano degli appunti qua dentro, devo recuperarli!!

(consigli del prof, non imparare a memoria!

## asimptotic efficiency of an algoritm

dato un n input abbastanza grande, é possibile studiarne l'efficienza in confronto ad un altro perché ció annulla l'interesse verso le parti che crescono con velocità di ordine minore.

Analisi insertion-sort:

n is the number of elements!

```
insertion-sort(A, n)        // A = [] an array of numbers
    for i=2 to n
        key = A[i]
        j = i-1
        while j>0 and A[j] > key
            A[j+1] = A[j]
            j -= 1
        A[j+1] = key
```

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| i=2 | 5 | 2 | 4 | 6 | 1 | 3 |

key = 2
j = 1

| 2 | 5 | 4 | 6 | 1 | 3 |
|---|---|---|---|---|---|

i=3

| 2 | 5 | 4 | 6 | 1 | 3 |
|---|---|---|---|---|---|

key = 4
j = 2

| 2 | 4 | 5 | 6 | 1 | 3 |
|---|---|---|---|---|---|

i=4

| 2 | 4 | 5 | 6 | 1 | 3 |
|---|---|---|---|---|---|

key = 6
j = 3

| 2 | 4 | 5 | 6 | 1 | 3 |
|---|---|---|---|---|---|

// 6 > tutti i suoi precedenti!

$i = 5$

| | | | | | |
|---|---|---|---|---|---|
| 2 | 4 | 5 | 6 | 1 | 3 |

key = 1
$j = 4$

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 4 | 5 | 6 | 3 |

$i = 6$

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 4 | 5 | 6 | 3 |

key = 3
$j = 5$

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |

algoritmo concluso

```
insertion-sort(A, n)
    for i=2 to n
        key = A[i]
        j = i-1
        while j>0 and A[j] > key
            A[j+1] = A[j]
            j -= 1
        A[j+1] = key
```

| | istru. | rip. |
|---|---|---|
| | $c_1$ | $n$ |
| | $c_2$ | $n-1$ |
| | $c_3$ | $n-1$ |
| | $c_4$ | $\sum_{i=2}^{n} t_i$ |
| | $c_5$ | $\sum_{i=2}^{n} (t_i - 1)$ |
| | $c_6$ | $\sum_{i=2}^{n} (t_i - 1)$ |
| | $c_7$ | $n-1$ |

best case = quando array già ordinato, per cui $t_i = 1$!

$$c_1(n) + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$

$=$

$$(c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

$\downarrow$

ordine + grande

lineare tempo di esecuzione $= n$ quindi $T(n) = n$ (best case) formula famosa bla bla bla

$$\frac{n(n+1)}{2} = \sum_{i=1}^{n} i$$

$n = 3$

$1+2+3 = 6$ ✓

$\frac{3 \cdot 4}{2} = \frac{12}{2} = 6$

anche questa

$$\sum_{i=2}^{n} (i-1) = \frac{n(n-1)}{2}$$

per cui il peggior scenario, ovvero ordinato in ordine decrescente:

$$c_1(n) + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$

$$=$$

$$\left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7)$$

$T(n)$ (worst-case) $= n^2$ ✓

notazione per worst-case = Theta $n^2$ = $\Theta(n^2)$

quando un alg. é migliore di un altro?

quando l'ordine di crescita di $\Theta()$ é minore!

## Divide and conquer method (resolving recursion)

merge (A, p, q, r) // $p \leq q < r$

```
nL = q-p+1
nR = r-q
let L[0 : nL-1] and R[0 : nR-1] be new arrays
for i=0 to nL-1
    L[i] = A[p+i]
for j=0 to nR-1
    R[j] = A[q+j+1]
i = 0
j = 0
k = p
while i < nL and j < nR
    if L[i] ≤ R[j]
        A[k] = L[i]
        i += 1
    else A[k] = R[j]
        j += 1
    k = k+1
while i < nL
    A[k] = L[i]
    i = i+1
```

k = k + 1

while j < nR

A[k] = R[j]

j = j + 1

k = k + 1

merge sort (A, p, r)

if p ≥ r

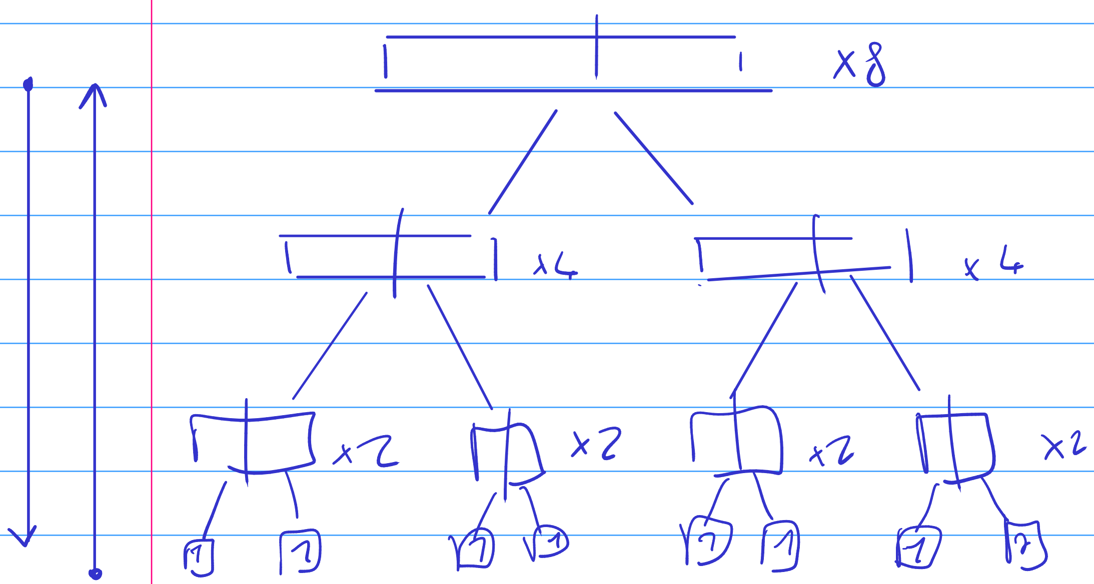
return

q = ⌊(p + r)/2⌋

merge sort (A, p, q)

merge sort (A, q+1, r)

merge (A, p, q, r);




to complete the merge sort call, the T(n) is

$$T(n): \begin{cases} \Theta(1) & \text{if } n < n_0 \\ D(n) + aT(n/b) + C(n) \end{cases}$$

(sono arrivato ad un numero solo rimasto)

tempo per dividere i due sottovettori

tempo per risolvere i sottoproblemi

tempo per unire i due sottovettori

$\Theta(1)$ è costante

$\Theta(n)$ dipende da $n$

quindi abbiamo

$$T(n) = 2T(n/2) + \Theta(n)$$

Ipotesi

1) $f(n) = O(g(n))$

2) $g(n) = O(h(n))$

Tesi $f(n) = O(h(n))$ // prima capisco l'intuizione (c.prof)

$\downarrow$

dato quello scritto sopra

$$\exists\, c_1 > 0 \ \exists\, n_1 \in \mathbb{N} \ni' \forall n \geq n_1,\ f(n) \leq c_1 g(n)$$

$$\exists\, c_2 > 0 \ \exists\, n_2 \in \mathbb{N} \ni' \forall n \geq n_2,\ g(n) \leq c_2 h(n)$$

$$\exists\, c_3 > 0 \ \exists\, n_3 \in \mathbb{N} \ni' \forall n \geq n_3,\ f(n) \leq c_3 h(n)$$

$n_3 = \max\{n, n_2\}$ quindi $\forall n \geq n_3$ risultano vere:

- sia $f(n) \leq c_1 g(n)$
- sia $g(n) \leq c_3 h(n)$

$\longrightarrow$ $f(n) \leq c_1 g(n) \leq c_3 h(n)$

$\downarrow$

$c_3 = c_1 c_2$ // $> 0$

1) Utilizzando la stessa teoria, trovare che

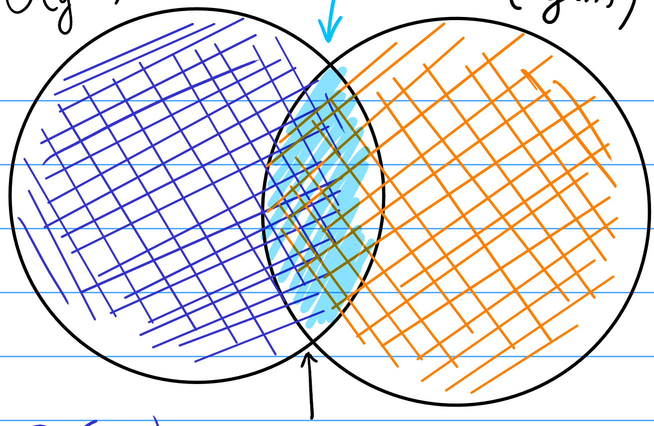
$$f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$$

$$f(n) = \Theta(g(n)) \iff g(n) = \Theta(f(n))$$

può succedere? (prossima pagina)

$O(g(n))$ $\Omega(g(n))$

$\omega(g(n)) \subset \Omega(g(n))$



$o(g(n)) \subset O(g(n))$

$\Theta(g(n))$ se e soltanto se, ricade appartiene ad entrambi.

$o \neq O \rightarrow o(g(n))$ = insieme delle funzioni che

$$\{f(n) \mid \underbrace{\forall c>0}_{\text{per ogni, non "esiste un" come le altre formule}} \underbrace{\exists n_0 \in \mathbb{N} \ni' \forall n>n_0}_{\text{"per n abbastanza grande"}} \quad \underbrace{f(n) < c\, g(n)}_{\text{strettamente minore!}}\}$$

$\omega \neq \Omega \rightarrow \omega(g(n))$ insieme delle funzioni che

$$\{f(n) \mid \underbrace{\forall c>0}_{\text{per ogni, non "esiste un" come le altre formule}} \underbrace{\exists n_0 \in \mathbb{N} \ni' \forall n>n_0}_{\text{"per n abbastanza grande"}} \quad \underbrace{f(n) > c\, g(n)}_{\text{strettamente maggiore!}}\}$$

? 1) $o(g(n)) \cap \Omega(g(n)) = \emptyset$ ?

2) $\omega(g(n)) \cap O(g(n)) = \emptyset$ ?

verifichiamolo:

1) procediamo per assurdo. Supponiamo per assurdo che sia falsa

$=$

$$f(n) = o(g(n)) \wedge f(n) = \Omega(g(n))$$

per definizione

per definizione

$$\forall c>0\ \exists n_0 \in \mathbb{N} \ni' \forall n > n_0\ f(n) < g(n)$$

$$\exists c_1 > 0\ \exists n_0' \in \mathbb{N} \ni' \forall n > n_0'\ f(n) \geq c_1 g(n)$$

dalla prima proprietà, essendo che vale $\forall c > 0$:

$$\exists n_0 \ni' \forall n > n_0,\ f(n) < c_1 g(n)$$

$n_2 = \max\{n_0', n_0\}$

$\hookrightarrow$ $n \geq n_2$ le proprietà valgono entrambe !!

Proprietà

$$f(n) = o(g(n)) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$\{a_n\}_{n \in \mathbb{N}}$ $\quad l \in \mathbb{R}$

= si rimpicciolisce, il limite non è a infinito

$$l = \lim_{n \to \infty} a_n \iff \forall \varepsilon > 0 \ \exists n_0 \in \mathbb{N} \ni \forall n \geq n_0 \ |a_n - l| < \varepsilon$$

$$f(n) = \omega(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = +\infty$$

eserazio di esempio

$$\log(n) = O(\sqrt{n}) \longrightarrow \lim_{n \to \infty} \frac{\log n}{\sqrt{n}} \overset{?}{\neq} 0$$

bella domanda da fare

$$\lim_{n \to \infty} \frac{\log n}{\sqrt{n}} = \frac{\infty}{\infty} \overset{\text{derivate}}{=} (\ln \text{ per convenienza}) \quad \frac{1/n}{1/2\, n^{-1/2}} = \frac{2}{\sqrt{n}} = 0$$

io volevo solo vivere a colori cosa cazzo è successo
voglio piangere, i miei occhi si stanno chiudendo, merda!

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = l \ , \ ]0, 1[ \ , \ \underline{f(n) = \Theta(g(n))}$$