

Algoritmo (per numeri di fibonacci)

$$F_n = \begin{cases} 1 \\ F_{n-1} + F_{n-2} \end{cases}$$

$$n = 1, 2$$

$$n \geq 3$$

formula di Binet

$$x^2 = x + 1 \rightarrow x^2 - x - 1 = 0$$

$$\begin{cases} \Phi = \frac{1 + \sqrt{5}}{2} \approx 1.6... \\ \hat{\Phi} = \frac{1 - \sqrt{5}}{2} \approx -0.6... \end{cases}$$

sezione aurea
complemento della sezione aurea

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

Dimostrazione tramite passo induttivo

$$n=1 \rightarrow F_1 = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2} \right) = \frac{1}{\sqrt{5}} \left(\frac{2\sqrt{5}}{2} \right) = 1 \checkmark$$

$$n=2 \rightarrow F_2 = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^2 - \left(\frac{1-\sqrt{5}}{2} \right)^2 \right)$$

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

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

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

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

per ipotesi induttiva allora

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

$$\frac{1}{\sqrt{5}} \left[\left(\Phi^{n-1} + \Phi^{n-2} \right) - \left(\hat{\Phi}^{n-1} + \hat{\Phi}^{n-2} \right) \right]$$

...

$$\frac{1}{\sqrt{5}} \left(\Phi^n - \hat{\Phi}^n \right) \quad ? \quad \begin{array}{l} \text{come ci arrivo?} \\ \text{dimostro se è possibile} \\ \text{che} \end{array}$$

$$\left\{ \begin{array}{l} \Phi^n = \Phi^{n-1} + \Phi^{n-2} \\ \hat{\Phi}^n = \hat{\Phi}^{n-1} + \hat{\Phi}^{n-2} \end{array} \right.$$

se sono entrambe
vere la dimostrazione
è finita

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

$$\left\{ \begin{array}{l} \Phi^2 = \Phi + 1 \\ \hat{\Phi}^2 = \hat{\Phi} + 1 \end{array} \right.$$

\rightarrow per definizione di Φ , avere

$x^2 = x + 1$ la dimostrazione
è verificata

pseudocodice:

Fib(int n) \rightarrow int

if $n \leq 2$ then return 1;

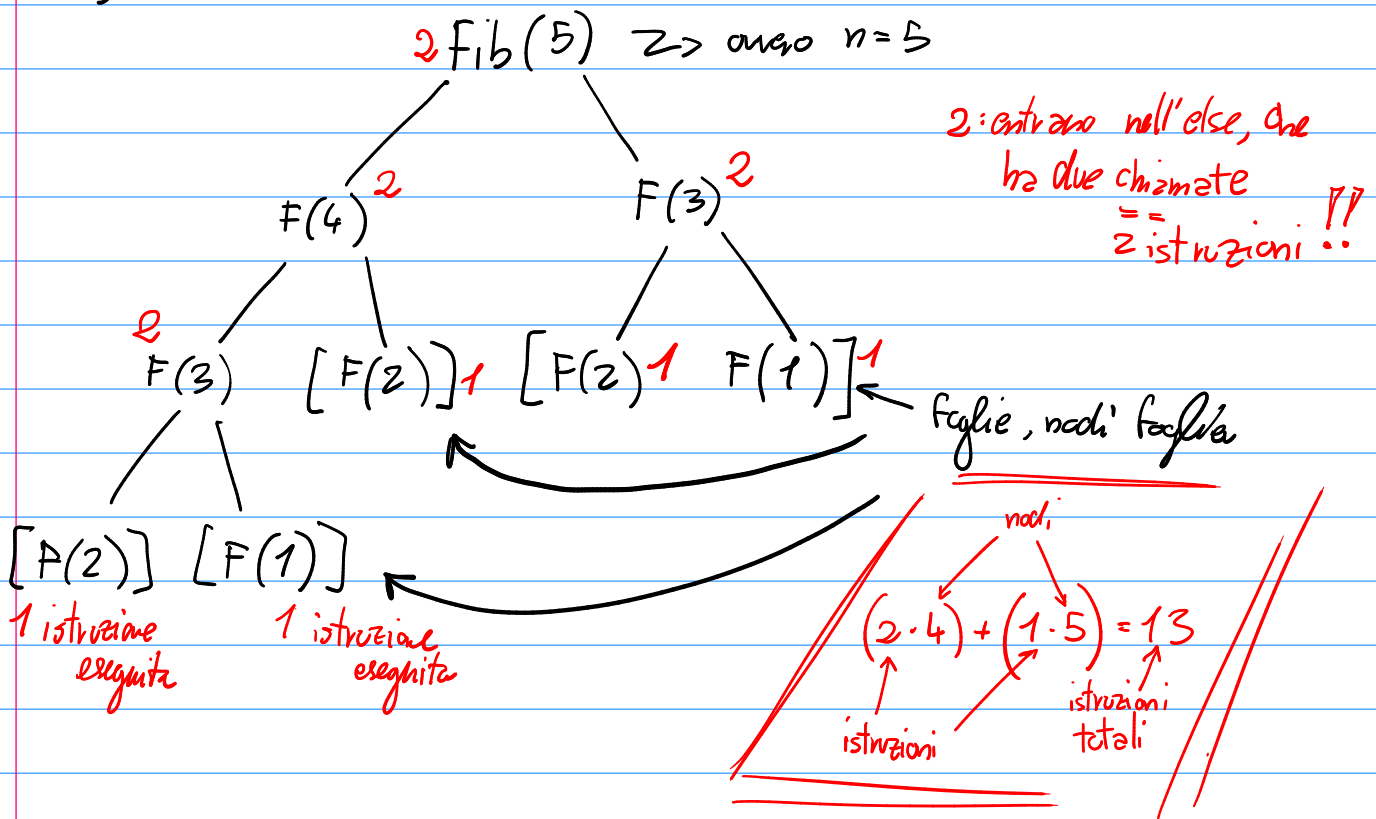
else return Fib(n-1) + Fib(n-2)

↑
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

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

\uparrow non foglie \nwarrow $f = \text{foglie}$

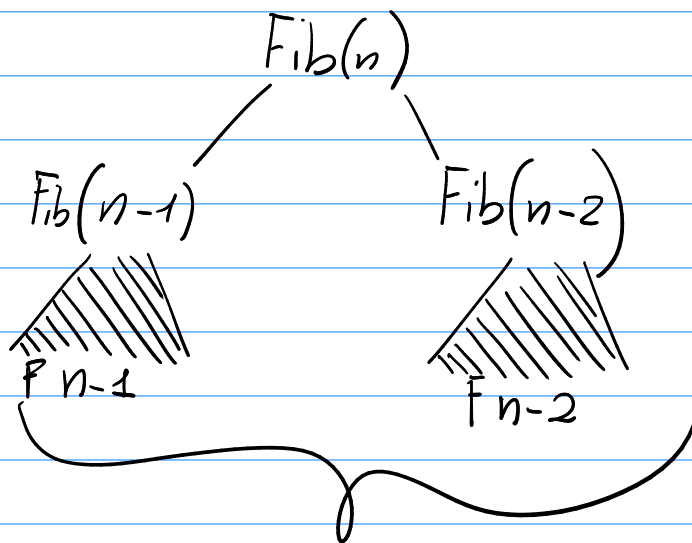
Proprietà 1 Sia T_n l'albero delle ricorrenze relativo alla chiamata $\text{Fib}(n)$.

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

$$f(T_n) = F_n$$

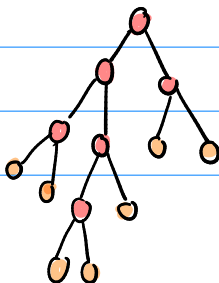
Dim: induttiva su n (meglio con il disegno)

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

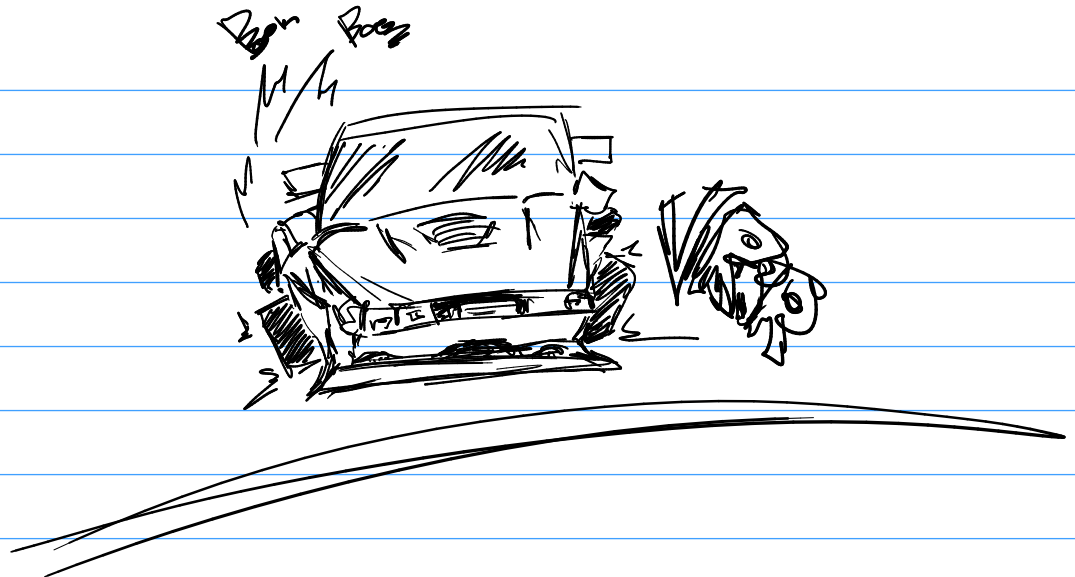


F_n = iesimo numero di Fibonacci

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



\bullet = $i(T_n) = f(T_n) - 1$ (albero binario)
 \circ = $F(n)$ (caso fibonacci)



$\text{Fib}(\text{int } n) \rightarrow \text{int}$

if $n \leq 2$ then return 1.

else return $\text{Fib}(n-1) + \text{Fib}(n-2)$

ricordo

$T(n) =$ 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 \Rightarrow prof ut f

formula generale

$$T(n) = c \cdot (T_n) + f(T_n)$$

↑
costi non
ricorsive

↑
foglio

$$a) \quad f(T_n) = F_n$$

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

riprenendo
pagine
successive

quindi posso semplificare la formula generale con:

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

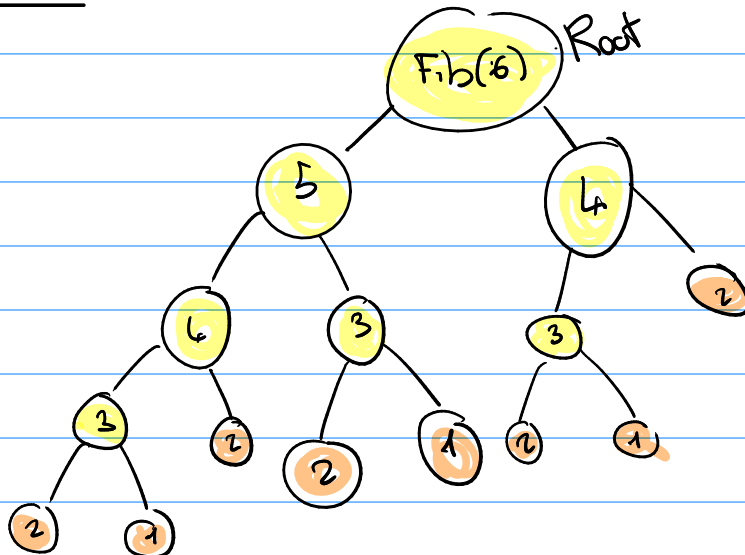
complessità cresce "come i numeri di Fibonacci"

$$T(n) \approx F_n \quad \forall n \geq 6$$

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

Ind. su n (verifichiamo)

Base : $n = 6, 7$



$$n \geq 8$$

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

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

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

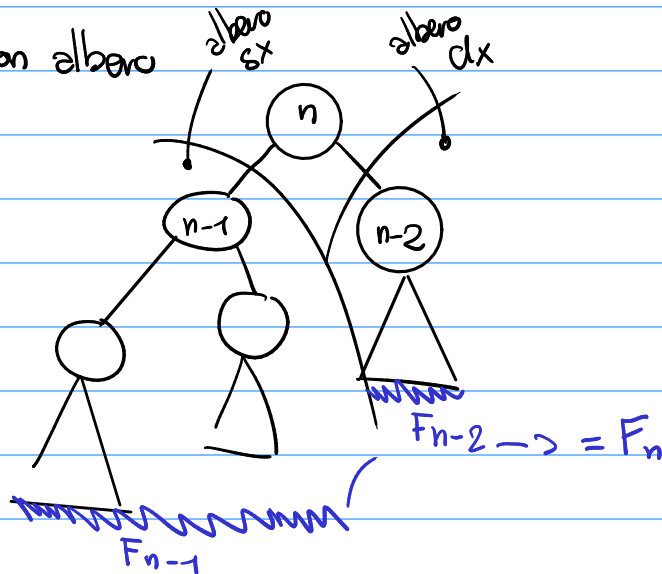
riprendo formule

$$f(T_n) = F_n \quad ? \quad \text{numero figli} = F_n ?$$

base $n=1,2 \rightarrow$ non ci sono nodi foglia del

$$n \geq 3$$

\rightarrow ipotesi con albero




totale nodi foglia = somma numero nodi foglia sx e dx

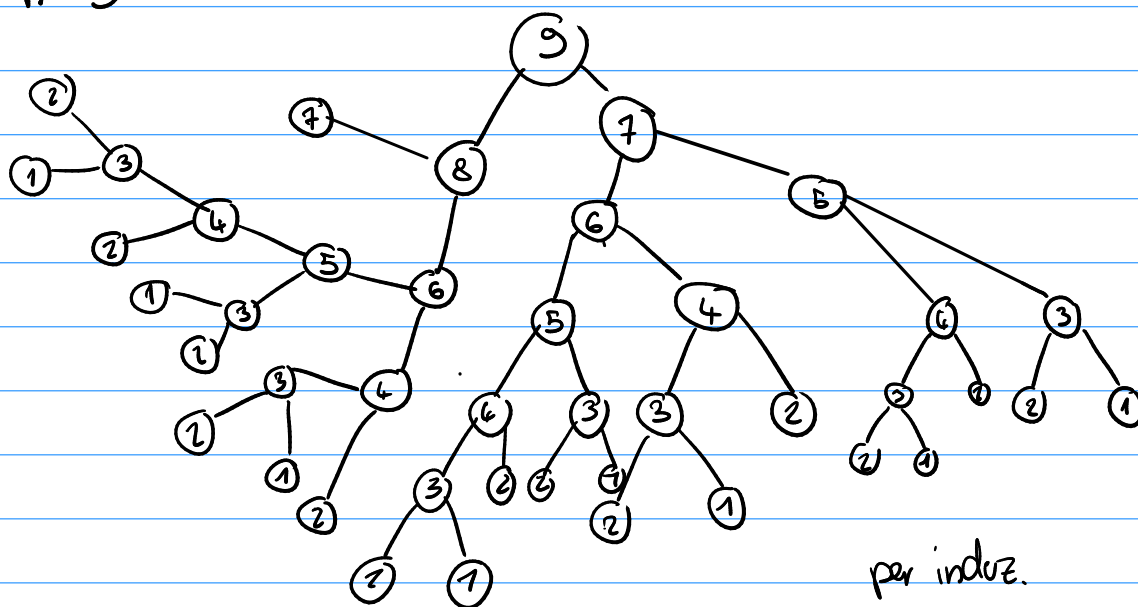
Se T è un albero binario dove ogni vertice interno ha esattamente 2 figli:
allora $i(T) = f(T) - 1$

Dimostrazione (ind sulla grandezza dell'albero)

Base $n = 1, 2$

 non ha nodi figli!

$n = 9$



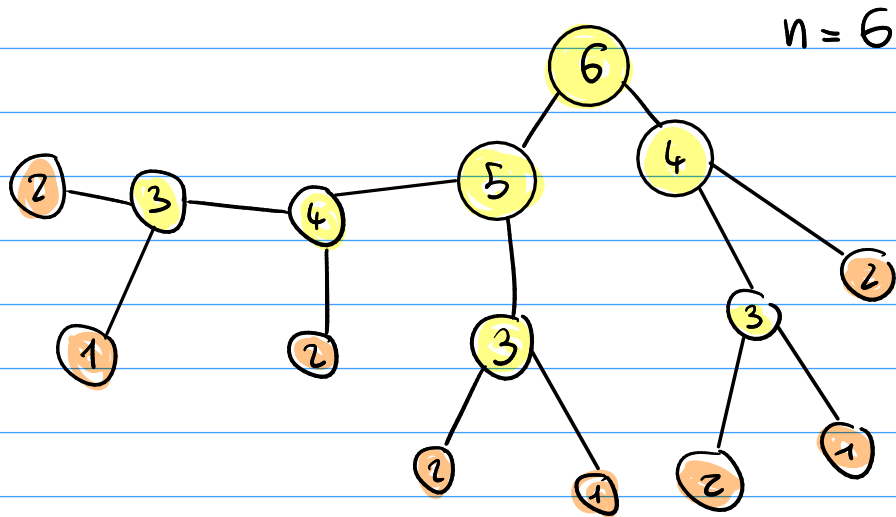
per induz.
 $n = n$ nodi

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

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

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

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



$\text{Fib3}(\text{int } n) \rightarrow \text{int}$

1. allocare 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 $F[n]$;

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

| 4
4
3

1, 2, 8

$= 2 \cdot n \rightarrow 2$ é non considerabile $\Rightarrow n$

non come degli appalti! qua dentro, dev' essere così!!

(consiglio del prof, non imparare a memoria!

asymptotic efficiency of an algorithm

dato un n input abbastanza grande, è possibile studiare l'efficienza in confronto ad un altro perché ciò 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 $A[j] > \text{key}$

$A[j+1] = A[j]$

$j = j - 1$

$A[j+1] = \text{key}$

$i = 2$

1	2	3	4	5	6
5	2	6	6	1	3

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

key = 2

$j = 1$

$i = 3$

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

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

key = 4

$j = 2$

$i = 4$

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

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

key = 6

$j = 3$

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 |

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 = j - 1

A[j+1] = Key

algoritmo

concluso

istru. n.p.

C₁ n

C₂ n-1

C₃ n-1

C₄ $\sum_{i=2}^n t_i$

C₅ $\sum_{i=2}^n (t_i - 1)$

C₆ $\sum_{i=2}^n (t_i - 1)$

C₇ n-1

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

$$C_1(n) + C_2(n-1) + C_3(n-1) + C_4(n-1) + C_7(n-1)$$

| =

$$(C_1 + C_2 + C_3 + C_4 + C_7)n - (C_2 + C_3 + C_4 + C_7)$$

ordine + grande

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

formule formule bla bla bla

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

$$n=3$$

$$1+2+3 = 6 \checkmark$$

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

inde questa

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

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

$$C_1(n) + C_2(n-1) + C_3(n-1) + C_4\left(\frac{n(n+1)}{2}-1\right) + C_5\left(\frac{n(n-1)}{2}\right) + C_6\left(\frac{n(n-1)}{2}\right) + C_7(n-1)$$

| =

$$\left(\frac{C_4}{2} + \frac{C_5}{2} + \frac{C_6}{2}\right)n^2 + \left(C_1 + C_2 + C_3 + \frac{C_4}{2} - \frac{C_5}{2} - \frac{C_6}{2} + C_7\right)n - (C_2 + C_3 + C_4 + C_7)$$

$$T(n) \text{ (worst-case)} = n^2 \checkmark$$

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

quando un alg. è migliore di un altro?

quando l'ordine di crescita di $\Theta()$ è 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 $L[i] \leq R[j]$

$A[k] = L[i]$

$i = i + 1$

else $A[k] = R[j]$

$j = j + 1$

$k = k + 1$

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

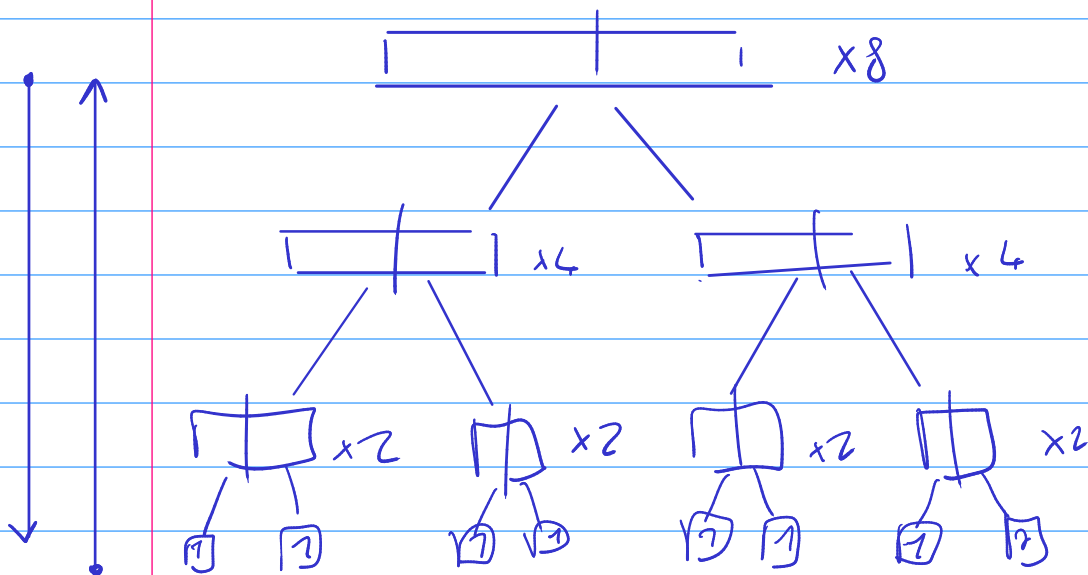
```

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

$$T(n): \begin{cases} \Theta(1) & \text{if } n < n_0 \text{ (sono arrivati ad un numero solo rimasto)} \\ D(n) + aT(n/b) + C(n) \end{cases}$$

↑
tempo per dividere
i due sottovettori

↑
tempo per unire
i due sottovettori

tempo per risolvere
i sottoproblemi

↓
 $\Theta(1)$
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 esplicito l'istituzione (c, n₀)"

dato quello scritto sopra

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

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

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

$$n_3 = \max \{n_1, n_2\} \quad \text{quindi } \forall n \geq n_3 \text{ risultato vero:}$$

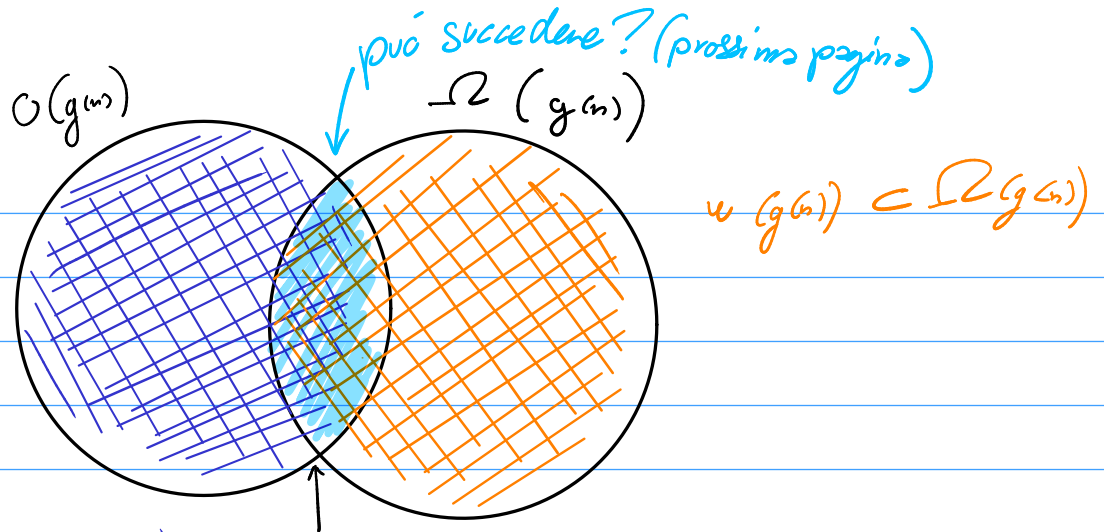
$$\begin{array}{l} \text{- sic } f(n) \leq c_1 g(n) \\ \text{- sic } g(n) \leq c_2 h(n) \end{array} \quad \left. \vphantom{\begin{array}{l} \text{- sic } f(n) \leq c_1 g(n) \\ \text{- sic } g(n) \leq c_2 h(n) \end{array}} \right\} \rightarrow f(n) \leq c_1 g(n) \leq c_3 h(n)$$

\downarrow
 $c_3 = c_1 c_2 \quad // > 0$

1) Utilizzando la stessa teoria, trovare che

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

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



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

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

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

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

per ogni,
non "esiste in"
come le altre formule

"per n abbastanza
grande"

strettamente
minore!

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

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

per ogni,
non "esiste in"
come le altre formule

"per n abbastanza
grande"

strettamente
maggiore!

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

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

verificandolo:

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

$$\overset{\text{per definizione}}{\downarrow} \quad f(n) = o(g(n)) \quad \wedge \quad f(n) = \Omega(g(n))$$

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

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

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

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

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

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

~~~~~~~~~

Proprietà

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

$$\{a_n\}_{n \in \mathbb{N}}$$

$$l \in \mathbb{R}$$

= se si impicciolisce,  
il limite non è  
infinito

$$l = \lim_{n \rightarrow \infty} a_n \iff \forall \varepsilon > 0 \exists n_0 \in \mathbb{N} \Rightarrow \forall n \geq n_0$$

$$|a_n - l| < \varepsilon$$

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

esercizio di esempio

$$\log(n) = O(\sqrt{n}) \longrightarrow \lim_{n \rightarrow \infty} \frac{\log n}{\sqrt{n}} \stackrel{?}{=} 0 \quad \begin{array}{l} \text{bella} \\ \text{domanda} \\ \text{da fare} \end{array}$$

$$\lim_{n \rightarrow \infty} \frac{\log n}{\sqrt{n}} = \frac{\infty}{\infty} = \text{da fare} \quad \frac{1/n}{1/2 n^{-1/2}} = \frac{2}{\sqrt{n}} = 0$$

io vorrei solo vivere a colori con un computer e un  
voglio piangere, i miei occhi si stanno chiudendo, merda!

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

$$\begin{array}{lcl} \text{Block 1} & O(f(n)) & \\ \text{Block 2} & O(g(n)) & \Rightarrow O(f(n) + g(n)) \end{array}$$

immagina, in pseudocode:

$$\begin{array}{lcl} \text{if } \langle \text{cond} \rangle \text{ then} & O(f(n)) & \\ \quad \text{Rama then} & O(g(n)) & \left. \vphantom{\begin{array}{l} O(f(n)) \\ O(g(n)) \end{array}} \right\} O(f(n) + \\ \text{else} & O(h(n)) & \max \{O(g(n)), O(h(n))\} \\ \quad \text{Rama else} & & \end{array}$$

$$\begin{array}{lcl} \text{for } \langle \text{cond} \rangle \text{ to } k & & \\ \quad \text{Block for } O(f(n)) & \left. \vphantom{\begin{array}{l} \text{for } \langle \text{cond} \rangle \text{ to } k \\ \text{Block for } O(f(n)) \end{array}} \right] \rightarrow O(k f(n)) \end{array}$$

"i for persone essere zitti"

$$\begin{array}{lcl} \text{for } \langle \text{cond} \rangle \text{ to } k & & \\ \quad \text{for } \langle \text{cond} \rangle \text{ to } j & \nearrow & O(k j f(n)) \\ \quad \quad \text{Block for } O(f(n)) & & \end{array}$$

$$\text{while } \langle \text{cond} \rangle \text{ do } \rightarrow O(f(n))$$

$$\text{Block } \rightarrow O(g(n))$$

$N(n) = \max$  numero di iterazioni

$$O(N(n) (f(n) + g(n)))$$

↓  
 loop  
 max  
 times

check cond  
 while scope  
 code

## Metodo dell'iterazione

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

complessità algoritmo  
 di  
 ricorrenza binaria

metodo che come esecuzione ha quelle di provare con poche iterazioni la presenza di un pattern per il calcolo della complessità

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

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

$$= \left[ \left[ T\left(\frac{n}{8}\right) + C \right] + C \right] + C = T\left(\frac{n}{2^3}\right) + 3C$$

$$= T\left(\frac{n}{2^k}\right) + kC$$

$$\frac{n}{2^k} = 1 \quad ? \quad n = 2^k = \log_2 n$$

$$T\left(\frac{n}{2^{\lg_2 n}}\right) + \lg_2 n c$$

$$| =$$

$$T(1) + \lg_2 n c$$

$$| =$$

$$O(\lg_2 n)$$

$$T(n) = \begin{cases} 1 & n=1 \\ gT\left(\frac{n}{3}\right) + n & n>1 \end{cases}$$

$$g\left(gT\left(\frac{n}{3^2}\right) + \frac{n}{3}\right) + n = g^2\left(T\left(\frac{n}{3^2}\right)\right) + 4n$$

$$| =$$

$$g^2\left[gT\left(\frac{n}{3^3}\right) + \frac{n}{3^2}\right] + 4n$$

$$| =$$

$$g^3 T\left(\frac{n}{3^3}\right) + \underbrace{g_n + 4n}_{3^2 n + 3n + n}$$

$$g^k T\left(\frac{n}{3^k}\right) + n \sum_{i=0}^{k-1} 3^i$$

$$n(3^2 + 3^1 + 3^0)$$

do recurrence!  $[q \neq 1]$

$$\sum_{i=0}^k q^i = \frac{q^{k+1} - 1}{q - 1}$$

$\frac{n}{3^k} = 1?$

$$g^k T\left(\frac{n}{3^k}\right) + n \frac{3^k - 1}{2}$$

$O(\lg_3 n)$