18.09.2023

Numeri di Fibonacci:

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

Quindi: 1, 1, 2, 3, 5, 8, 13 ...

SUPPONIAMO DI SCRIVERE UNA PROCEDURA CHE PRENDA UN INTERO n E RESTITUISCA IL PROSSIMO NUMERO DI FIBONACCI

1) Binet scoprì che l'n-esimo numero di Fibonacci può essere scritto come:

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

2) Sappiamo che $ax^2 + bx + c = 0 \longrightarrow x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

3) Se ho $x^2 - x - 1 = 0 \longrightarrow x^2 = x + 1$ (come per Fibonacci!)

$a = 1$
$b = -1$
$c = 1$

$$x_{1,2} = \frac{1 \pm \sqrt{5}}{2}$$

Quindi $\varphi = x_1 = \frac{1 + \sqrt{5}}{2}$

$\hat{\varphi} = x_2 = \frac{1 - \sqrt{5}}{2}$

N.B. φ è un irrazionale: la somma di due irrazionali dà un intero.

4) Fib1 (int n)
return $\frac{1}{\sqrt{5}} \left(\varphi^n - \hat{\varphi}^n \right)$; ⇝ PRIMO ALGORITMO PER CALCOLARE IL N° DI FIBONACCI

Domande da porsi:

(1) CONVERGE? (cioè si ferma prima o poi?)
(2) RISOLVE IL PROBLEMA? CON QUALI CONDIZIONI?
(3) GRADO DI COMPLESSITÀ? (quante istruzioni vengono eseguite?)

Il nostro algoritmo converge ed esegue un'istruzione, quindi ha grado di complessità basso.

DIMOSTRO CHE FUNZIONA TRAMITE INDUZIONE

Base → n = 1,2
Passo ind. n ≥ 3 — Assumo che vada fino a (n-1) e lo dimostro per n] INDUZIONE

Per n = 1

$$F_1 = \frac{1}{\sqrt{5}} (\varphi - \hat{\varphi}) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2} \right) = \frac{1}{\sqrt{5}} \left(\frac{2\sqrt{5}}{2} \right) = 1 \quad \text{OK}$$

Per $n = 2$ (controllo anche per 2 perché φ è di 2 sol)

$$F_1 = \frac{1}{\sqrt{5}}\left(\frac{(1+\sqrt{5})^2}{4} - \frac{(1-\sqrt{5})^2}{4}\right) = \frac{1}{\sqrt{5}}\left(\frac{(1+5+2\sqrt{5})-(1+5-2\sqrt{5})}{4}\right) = \frac{1}{\sqrt{5}}\cdot\sqrt{5} = 1$$

Passo induttivo, $n = 3$

~~PER DEFINIZIONE~~, $F_n = F_{n-1} + F_{n-2}$

Suppongo vero fino a $n-1$, so che $n-1$ è calcolabile e quindi posso procedere

IPOTESI INDUTTIVA: Vale per F_{n-1} e per F_{n-2}, quindi posso sostituire

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

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

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

$$= \frac{1}{\sqrt{5}}\left[\varphi^{n-2}\varphi^2 - \hat{\varphi}^{n-2}\hat{\varphi}^2\right] = \frac{1}{\sqrt{5}}\left(\varphi^n - \hat{\varphi}^n\right)$$ SICURAMENTE CORRETTA

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

19.09.2013

PSEUDOCODICE

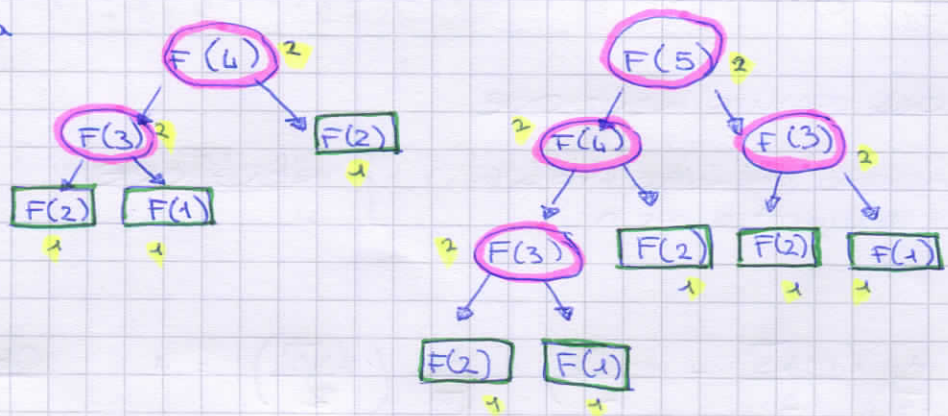
```
Fib2 (int n) → int
    if (n ≤ 2) then return 1
    then return Fib2(n-1) + Fib2(n-2)
```

COSA FA NEL CASO...?

(n = 1)	1
(n = 2)	1
(n = 3)	4
(n = 4)	7
(n = 5)	11

→ NUMERO DI ISTRUZIONI ESEGUITE DALL'ALGORITMO

- nodi foglia
- nodi interni
- n° istr.



TEMPO IMPIEGATO
$T(n) = 2 \cdot (Tn) + f(Tn)$

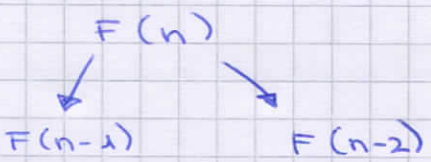
$i(Tn)$ = n° nodi interni
$f(Tn)$ = n° foglie

QUINDI:
Congettura → $f(Tn) = Fn$? $\forall\ n \geq 1$

↳ Ovvero: è vero che il numero di foglie equivale al numero di Fibonacci?

PROVO INDUTTIVAMENTE:

Per $n=1$, $n=2$ vale
Passo induttivo: $n \geq 3$. Ipotesi induttiva: fino a $(n-1)$ vale.
Perciò vale anche per n.



Se è vera per $F(n-1)$ allora lo sarà per $n-2$!

Congettura → $i(Tn) = f(Tn) - 1$? $\forall\ n \geq 1$

↳ È vero che il numero di nodi interni equivale al numero di foglie scalato di 1?

PROVO

Base: $n=1, 2$ vale
(come sopra, vale per $n-1$ quindi vale)

$i(T') = F(T') - 1$
$i(T) = i(T') + 1$
$f(T) = f(T') + 1$
$i(T) = i(T') + 1$
$= f(T')$
$= f(T) - 1$

QUINDI "TEMPO IMPIEGATO" È:

$T(n) = 2 \cdot (Tn) + Fn$

PER CASA

(1)
Dimostrare $Fn \geq 2^{\frac{n}{2}}$ $\forall\ n > 6$

(2)
Calcolare Fib(3)

↳ L'algoritmo!

ALGORITMO: insieme di istruzioni definite passo passo per essere eseguite meccanicamente e produrre un determinato risultato.

Misura di tempo → linee di codice

Relazione di ricorrenza: è tempo richiesto da una routine è uguale al tempo speso all'interno della routine più il tempo speso per le chiamate ricorsive.

Programmazione dinamica: risolvo il sottoproblema, memorizzo il risultato e lo uso poi

PROVO A DIMOSTRARE

$F_n \geq 2^{\frac{n}{2}} \quad \forall n \geq 6$

Base $n = 6$

$F_6 \geq 2^{6/2}$

$8 \geq 2^3$

$8 \geq 8$ OK

Passo $(n = 7)$

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

$F_{n-1} \geq 2^{\frac{n-1}{2}}$

$F_{n-2} \geq 2^{\frac{n-2}{2}}$

$F_n = 2^{\frac{n-1}{2}} + 2^{\frac{n-2}{2}} = 2^{\frac{n}{2}} \cdot 2^{-\frac{1}{2}} + 2^{\frac{n}{2}} \cdot 2^{-1} = 2^{\frac{n}{2}} \left(2^{-\frac{1}{2}} + 2^{-1}\right) =$

$= 2^{\frac{n}{2}} \underbrace{\left(\frac{1}{\sqrt{2}} + \frac{1}{2}\right)}_{>1} > 2^{\frac{n}{2}}$

$T(n) = 3F_n - 2$ Questo algoritmo non funziona bene. È ricorsivo, quindi ogni volta richiesta F_n. La soluzione è riscriverlo iterativamente.

Fib3 ITERATIVO:

1) Fib3(int n) → int — Alloca spazio (in modo dinamico) per F_n (array di n interi)

2) F[1] := F[2] := 1 ① ×
3) for each i = 3 to n do
4) F[i] = F[i-1] + F[i-2] ② ↝ verrà eseguito n-2 volte [i=k → n-(k+1)]
5) return F[n] ③

$T(n) = ③ + n-1 + n-2 = 2n$ (soluzione iterativa)

Fib4(int n) → int

a ← 1 b ← 1 ①
for i = 3 to n
c ← a + b ②
a ← b ③
b ← c ④
return b ⑤

$T(n) = 4n - 5 \simeq 4n$

T(n) → numero di linee di codice mandate in esecuzione

Fib4 è l'algoritmo Fib3 modificato per richiedere meno spazio in memoria. Ogni chiamata del ciclo usa i due valori precedenti di Fn, quindi basta usare 2 variabili al posto dell'array. (Quindi 2n (algoritmo precedente) per 2 = 4n)

24.09.2013

NOTAZIONE ASINTOTICA

DEFINIZIONE DI O-GRANDE

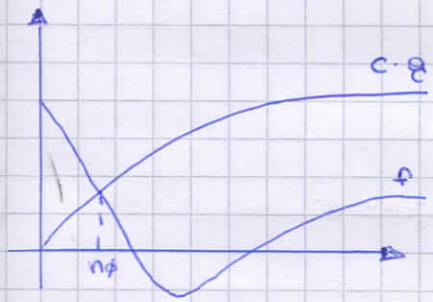
$$O(g(n)) = \{ f(n) \mid \exists c > 0, \exists n_0 \in \mathbb{N} \ni \forall n \geq n_0 : f(n) \leq c \cdot g(n) \}$$ (cioè per n sufficientemente grande)

$\rightsquigarrow \lim_{n \to \infty} \sup \left| \frac{f(n)}{g(n)} \right| < \infty$

Il suo obbiettivo è di caratterizzare il comportamento di una funzione per argomenti elevati in modo semplice ma rigoroso, per poter confrontare il comportamento di più funzioni tra loro.

In pratica, posso rappresentare due algoritmi come due funzioni [sempre positive]

ESEMPIO



N.B! Se scrivo $f(x) = O(g(x))$ sto dicendo che $f(x)$ è dell'ordine di $g(x)$, non sto affermandone l'uguaglianza, in quanto $O(g(x))$ rappresenta una CLASSE di funzioni

→ studio come aumentano il tempo e la memoria a seconda dell'algoritmo

ORDINI DI FUNZ. PIÙ COMUNI

$O(1)$ costante
$O(\log(n))$ logaritmica
$O(n^2)$ quadratica
$O(n)$ lineare

P.ES.

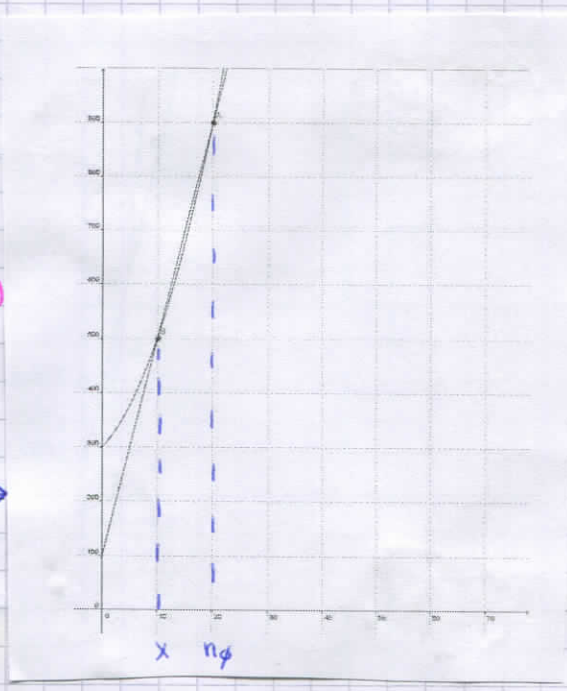
$$\underbrace{40n + 100}_{f(n)} \overset{?}{=} O(\underbrace{n^2 + 10n + 300}_{g(n)}) \quad \text{se pongo } c=1$$

$$40n + 100 = n^2 + 10n + 300$$

$$40n + 100 - n^2 - 10n - 300 = 0$$

$$n^2 - 30x + 200 = 0$$

$$n_{1,2} = \frac{30 \pm \sqrt{100}}{2} \quad \begin{matrix} \nearrow 20 \\ \searrow 10 \end{matrix}$$



DEFINIZIONE DI Ω-GRANDE

$$\Omega(g(n)) = \{ f(n) \mid \exists c > 0, \exists n_0 \in \mathbb{N} \ni \forall n \geq n_0 : f(n) \geq c \cdot g(n) \} \rightsquigarrow \lim_{n \to \infty} \sup \left| \frac{f(n)}{g(n)} \right| < \infty$$

PROPRIETÀ

└▷ $f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$] ★

GIUSTO ▷ " $f(n)$ ha ordine minore/ug. di $g(n)$ se $g(n)$ ha ordine maggiore o uguale a $f(n)$ "

DIMOSTRO ⭐

(⇒)

Ipotesi: $f(n) = O(g(n))$ cioè $\{\exists c>0, \exists n_0 \in \mathbb{N} \ni \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$ definiz. di O-grande

Tesi: $g(n) = \Omega(f(n))$ cioè $\{\exists c'>0, \exists n'_0 \in \mathbb{N} \ni \forall n \geq n'_0 : c' f(n) \leq g(n)\}$

Se prendo $c' = \frac{1}{c}$ $\rightsquigarrow$ $c' \cdot f(n) \leq g(n)$ $\rightsquigarrow$ $\frac{1}{c} \cdot f(n) \leq g(n)$

DEFINIZIONE DI Θ-GRANDE

$$\Theta(g(n)) = \{\exists c_1, c_2 > 0, n_0 \in \mathbb{N} \ni \forall n \geq n_0 : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\}$$

$\{$ $\rightsquigarrow$ $0 < \lim_{n\to\infty} \inf \left|\frac{f(n)}{g(n)}\right| \leq \lim_{n\to\infty} \sup \left|\frac{f(n)}{g(n)}\right| < \infty$

"$f(n)$ ha stesso ordine di grandezza di $g(n)$"

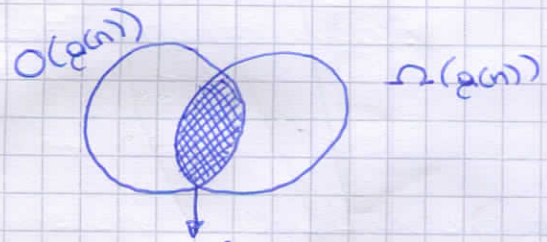
PROPRIETÀ

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

"$f(n)$ ha stesso ordine di $g(n)$ quando $f(n)$ ha ordine minore/uguale di $g(n)$ e, allo stesso tempo, maggiore/uguale di $g(n)$"

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

"$f(n)$ ha stesso ordine di $g(n)$ se $g(n)$ ha stesso ordine di $f(n)$"



ESEMPIO

$$\sqrt{n+10} = \Theta(\sqrt{n})$$

1) $\sqrt{n+10} = O(\sqrt{n})$] DEVO DIMOSTRARE
2) $\sqrt{n+10} = \Omega(\sqrt{n})$]

(1) $\exists c>0, \exists n_0 \in \mathbb{N} \ni \forall n \geq n_0 : \sqrt{n+10} \leq c\sqrt{n}$? ← DEFINIZIONE

$\sqrt{n+10} \leq c \cdot \sqrt{n} \iff (\sqrt{n+10})^2 \leq (c \cdot \sqrt{n})^2$ così tolgo la radice

$\sqrt{n+10} \leq c \cdot \sqrt{n} \iff n+10 \leq c^2 \cdot n$

$n + 10 - c^2 \cdot n \leq 0$ porto tutto da una parte

$n - c^2 \cdot n \leq 10$

$-n + c^2 \cdot n \geq 10$ cambio il segno (quindi giro ≤ che diventa ≥)

$n \cdot (c^2 - 1) \geq 10$

$(c^2 - 1) \geq \emptyset$

$c > 1$

Raccolgo. Noto che $(c^2-1) \geq \emptyset$ perché è un quadrato e c è maggiore di zero per definizione.

c è maggiore stretto di 1 perché se fosse uguale non funzionerebbe più la definizione, che ci dice di prendere un $c > \emptyset$

Quindi $n \geq \frac{10}{c^2-1}$

Perciò se per esempio prendo $c = \sqrt{2}$ è > 1, avrei $n_\emptyset = 10$

2) $\exists\ c > \emptyset,\ \exists\ n_\emptyset \in \mathbb{N} \ni \forall n \geq n_\emptyset : \sqrt{n+10} \geq c \cdot \sqrt{n}$ ← DEFINIZIONE

$\sqrt{n+10} \geq c \cdot \sqrt{n}$

$(\sqrt{n+10})^2 \geq (c \cdot \sqrt{n})^2$

$n + 10 \geq c^2 \cdot n$

$n + 10 - c^2 n \geq \emptyset$

$n - c^2 \cdot n \geq -10$

$c^2 \cdot n - n \leq 10$

$n(c^2 - 1) \leq 10$

$c^2 - 1 \leq \emptyset \rightarrow c^2 \leq 1 \rightarrow c \leq 1$ ma è comunque positivo, perché $c > 0$ per definiz. perciò $0 < c \leq 1$

Potrei per esempio scegliere $c = 1$ e avrei $n_\emptyset = 1$

~~PER CASA~~

$\frac{1}{2} n^2 - 3n = \Theta(n^2)$

1) $\frac{1}{2} n^2 - 3n = O(n^2)$

2) $\frac{1}{2} n^2 - 3n = \Omega(n^2)$

] Perché Θ-grande è fatta di O-grande e Ω grande

DEFINIZ. DI O-GRANDE

1) $\exists\ c > \emptyset,\ \exists\ n_\emptyset \in \mathbb{N} \ni \forall n \geq n_\emptyset : \frac{1}{2} n^2 - 3n \leq c \cdot n^2$ ←

$\frac{1}{2} n^2 - 3n \leq c \cdot n^2$

$\frac{1}{2} n^2 - 3n - c \cdot n^2 \leq \emptyset$] → divido per n

$\frac{1}{2} n - 3 - c \cdot n \leq \emptyset$

$\frac{1}{2} n - c \cdot n \leq 3$] → Raccolgo n

$n(\underbrace{\frac{1}{2} - c}_{\leq \emptyset}) \leq 3$

$\frac{1}{2} - c < \emptyset \Rightarrow c > \frac{1}{2}$

2) $\exists c>0, \exists n_0 \in \mathbb{N} \Rightarrow \forall n \geq n_0 : \frac{1}{2} n^2 - 3n \geq c \cdot n^2$ ← DEFINIZIONE Ω-GRANDE

$$\frac{1}{2} n^2 - 3n \geq c \cdot n^2$$

$$\frac{1}{2} n^2 - 3n - c \cdot n^2 \geq 0$$ → divido per n

$$\frac{1}{2} n - 3 - c \cdot n \geq 0$$

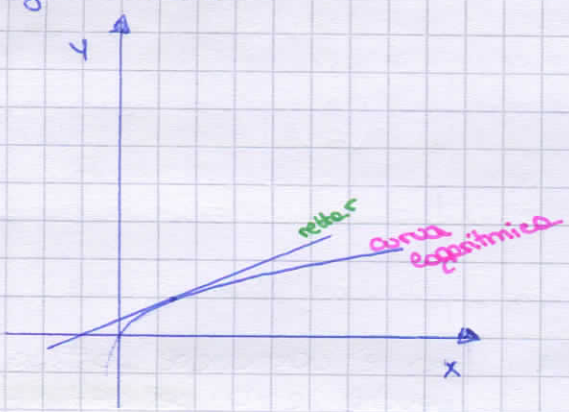
$$n\left(\frac{1}{2} - c\right) \geq 3$$

$$n \geq \frac{3}{\left(\frac{1}{2} - c\right)}$$

→ Qui lo posso fare perché so che $\left(\frac{1}{2} - c\right)$ è positivo. Nel punto 1) non si poteva.

ESERCIZI COL PROF. 26.09.2012

$\log n = O(n)$



OSSERVAZIONE GRAFICA:

(★) Comunque io prenda un punto sulla curva logaritmica la retta tangente passante per quel punto sarà sempre al di sopra della curva.

DEDUCO CHE, DATO (★), AVRÒ:

$\log x \leq x - 1 < x$ (questo perché x dà una retta)

perciò $\log x < x$

La definizione di O-grande dice che $\{\exists c>0, \exists n_0 \in \mathbb{N} \Rightarrow \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$, perciò se ho che $\log x$ è $f(n)$ e x è $g(n)$, $c=1$, è vero che $\log n = O(n)$.

ESEMPIO. $n = O(n^2)$

$n \leq 2^n$ → $n \cdot \log n = O(n^2)$ dove n è una funzione

ESERCIZIO 2

$f(n) = n\sqrt{n}$

$g(n) = 9^{\log_3 n} = n^2$ ←

$9^{\log_3 n} = (3^2)^{\log_3 n} = 3^{2\log_3 n} = 3^{\log_3 n^2} = n^2$

$n\sqrt{n} = O(n^2)$ Sappiamo che $n \leq n^2$. $\sqrt{n} < n$. Perciò $n\sqrt{n} \leq n^2$

ESERCIZIO 3

$n! = O(n^n)$

Sappiamo che:

$n! = n \cdot n-1 \cdot n-2 \ldots$

Perciò $n! \leq n^n$

$n^n = \underbrace{n \cdot n \cdot n \cdot n \cdot \ldots \cdot n}_{n\text{-volte}}$

ESERCIZIO 4

Dove $f(n) = n!$
$g(n) = 2^n$

$n! = \Omega(2^n)$

$n! = 1 \cdot 2 \cdot \ldots \cdot n \geq \underbrace{1 \cdot 2 \cdot 2 \cdot \ldots \cdot 2}_{n-1} = 2^{n-1}$

$\forall n \geq 1 \quad n! \geq 2^{n-1} = \left(\frac{1}{2}\right) 2^n \qquad f(n) \geq c \cdot g(n)$

c

ESERCIZIO 5

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

$\log n! = \log \prod_{i=1}^{n} i = \sum_{i=1}^{n} \log i \leq n \log n$ → definizione di O-grande

Il logaritmo del prodotto è la somma dei logaritmi

$n!$ potrebbe essere un fattoriale grande. Per calcolarlo posso usare la FORMULA DELL'APPROSSIMAZIONE DI STIRLING:

$$n! = \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \left(1 + \underbrace{\Theta(n)}_{f(n) = \Theta(n)}\right)$$

dove e = esponenziale

$f(n) + g(n) = \Theta(\max\{f(n), g(n)\})$

$\forall n \in \mathbb{N} \qquad h(n) = \max\{f(n), g(n)\}$

1) $\forall n \geq 1 \qquad f(n) + g(n) \leq 2\, h(n)$

$f(n) = \max\{f(x), g(x)\} \quad \vee \quad g(n) = \max\{f(x), g(x)\}$

$f(n) + g(n) \geq \max\{f(n), g(n)\}$ Ω

ESERCIZIO 6

$a, b \in \mathbb{R} \qquad b > 0 \qquad (n+a)^b = \Theta(n^b)$

$\exists\, c_1, c_2 > 0, \exists\, n_0 \in \mathbb{N} \Rightarrow \forall n \geq n_0 : c_1 \cdot n^b \leq (n+a)^b \leq c_2 \cdot n^b$ DEFINIZ. DI Θ

Considero due casi: $a > 0$ (I)
$a < 0$ (II)

I) $a > 0$

$c_1 \cdot n^b \leq (n+a)^b \leq c_2 \cdot n^b$

$c_1^{1/b} \cdot n \leq n + a \leq c_2^{1/b} \cdot n$ → divido gli esponenti per b

Considero solo $(n+a)^b \leq c_2 \cdot n^b$ perché ho $a > 0$

$n + a \leq c_2^{1/b} \cdot n$

$n + a - c_2^{1/b} \cdot n \leq 0$

$n - c_2^{1/b} \cdot n \leq -a$

$-n + c_2^{1/b} \cdot n \geq a$

$n\left(c_2^{1/b} - 1\right) \geq a$

$n \geq \frac{a}{c_2^{1/b} - 1}$

$c_2^{1/b} - 1 > 0 \Rightarrow c_2^{1/b} > 1, \quad c_2 > 1$

$n \geq a \qquad n_0 = \lceil a \rceil$

Se per esempio prendo $c_1 = 1$, $c_2 = 2^b$ ho $n_0 = \lceil a \rceil$ ← "parte intera superiore"

II) $a < 0$

Devo trovare c_2. Se c_2 fosse $= 1$, avrei n. n è sempre maggiore di $n - a$ (perché a in qst caso è negativo, quindi ho $(n + (-a))$)

$c_1^{1/b} \cdot n \leq n + a$

$-c_1^{1/b} \cdot n + n \geq -a$

$n\underbrace{(1 - c_1^{1/b})}_{>0} \geq -a$

$1 - c_1^{1/b} > 0$

$c_2 > 1 > c_1^{1/b}$

$c_2 > 1 > c_1$

$a, b \in \mathbb{R}, \quad b > 0$

Se prendo $c_1 = \left(\frac{1}{2}\right)^b$, $c_2 = 1$, $n_0 = \lceil 2|a| \rceil$

DEFINIZIONE DI O-PICCOLO

$$o(g(n)) = \{ f(n) \mid \forall c > 0, \exists n_0 \in \mathbb{N} \ni \forall n \geq n_0 : 0 \leq f(n) < c \cdot g(n) \}$$

PROPRIETÀ

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

DEFINIZIONE DI ω-PICCOLO

$$\omega(g(n)) = \{ f(n) \mid \forall c > 0, \exists n_0 \in \mathbb{N} \ni \forall n \geq n_0 : 0 \leq c \cdot g(n) < f(n) \}$$

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