Notazione asintotica
I parte

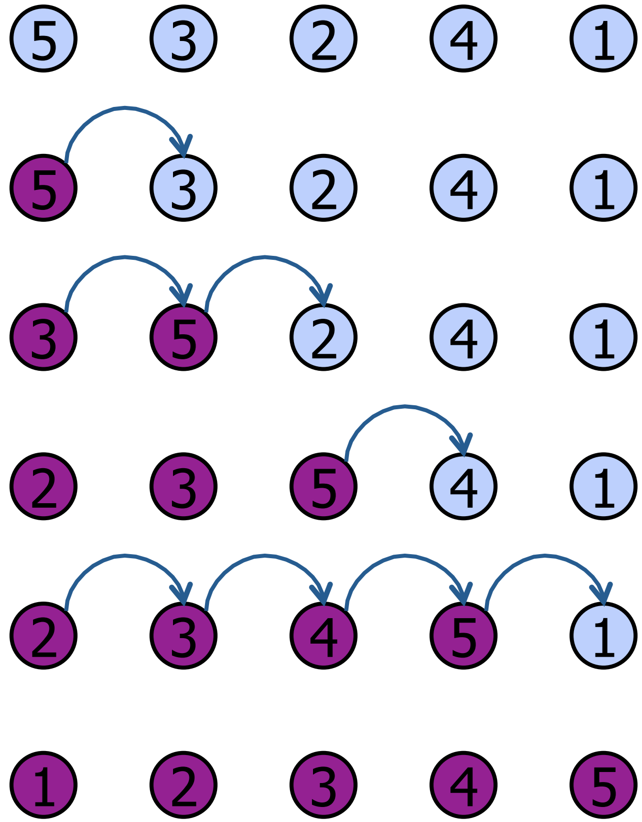
IASD a.a. 2015-16
A. De Bonis

Analisi degli algoritmi

- Esempio:

```
InsertionSort(a):       //n e` la lunghezza di a
 FOR(i=1;i<n;i=i+1){
   elemDaIns=a[i];
   j=i;
   while((j>0)&& a[j-1]>elemDaIns){ //cerca il posto per a[i]
        a[j]=a[j-1];     //shifto a destra gli elementi piu` grandi
        j=j-1;
}
 a[j]=elemDaIns;
}
```

Insertion-sort



Analisi di InsertionSort

InsertionSort(a):	Costo	Num. Volte
FOR(i=1;i<n;i=i+1){	c_1	n
elemDaIns=a[i];	c_2	n-1
j=i;	c_3	n-1
while((j>0)&& a[j-1]>elemDaIns){	c_4	$\sum_{i=1}^{n-1} t_i$
a[j]=a[j-1];	c_5	$\sum_{i=1}^{n-1} (t_i - 1)$
j=j-1;	c_6	$\sum_{i=1}^{n-1} (t_i - 1)$
}		
a[j]=elemDaIns;	c_7	n-1
}		

t_i = numero di iterazioni del while all'i-esima iterazione del for

Analisi di InsertionSort

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=1}^{n-1} t_i + c_5 \sum_{i=1}^{n-1} (t_i - 1) + c_6 \sum_{i=1}^{n-1} (t_i - 1) + c_7(n-1)$$

- Nel caso pessimo $t_i = i + 1$ (elementi in ordine decrescente ➔tutti gli elementi che precedono a[i] sono più grandi di a[i])

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=1}^{n-1} (i+1) + c_5 \sum_{i=1}^{n-1} i + c_6 \sum_{i=1}^{n-1} i + c_7(n-1)$$

Analisi di InsertionSort caso pessimo

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\sum_{i=1}^{n-1}(i+1) + c_5\sum_{i=1}^{n-1} i + c_6\sum_{i=1}^{n-1} i + c_7(n-1)$$

$$= (c_1 + c_2 + c_3 + c_7)n - c_2 - c_3 - c_7 + (c_4 + c_5 + c_6)\sum_{i=1}^{n-1} i + c_4(n-1)$$

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

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

$$= an^2 + bn + c$$

Ordine di grandezza

- Nell'analizzare la complessità di InsertionSort abbiamo operato delle astrazioni
  - Abbiamo ignorato il valore esatto prima delle costanti c_i e poi delle costanti a, b e c.

Ordine di grandezza

- Possiamo aumentare il livello di astrazione considerando solo l'ordine di grandezza
  - Consideriamo solo il temine "dominante"
    - Per InsertionSort: an^2
    - Giustificazione: più **grande** è n, minore è il contributo dato dagli altri termini alla stima della complessità
  - Ignoriamo del tutto le costanti
    - Diremo che InsertionSort ha complessità $O(n^2)$
    - Giustificazione: più grande è n, minore è il contributo dato dalle costanti alla stima della complessità

Ordine di grandezza

- $f(n)=10\ n^2+ 100\ n+ 1000$
- $n=5$: $f(5)= 10x25+100x5+1000=1750$
- $n=10$: $f(10)= 10x100+100x10+1000=3000$
- $n=20$: $f(20)=10x400+100x20+1000=7000$
- $n=100$: $f(100)=10x10000+100x100+1000=111000$
- $n=1000$: $f(1000)=10x10^6+100x1000+1000=10101000$

Efficienza asintotica degli algoritmi

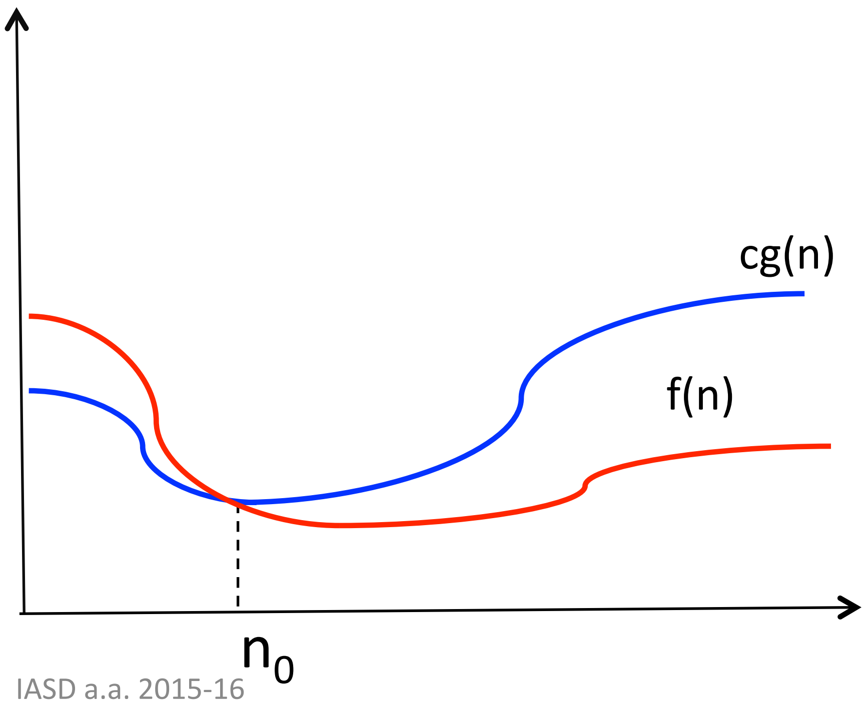
- Per input piccoli può non essere corretto considerare solo l'ordine di grandezza ma per input "abbastanza" grandi è corretto farlo

Notazione asintotica

- Date $f : n \in N \rightarrow f(n) \in R^+$, $g : n \in N \rightarrow g(n) \in R^+$, scriveremo $f(n) = O(g(n))$

$\Leftrightarrow \exists\, c > 0,\ \exists\, n_0$ tale che $f(n) \leq cg(n),\ \forall n \geq n_0$

Informalmente, $f(n) = O(g(n))$ se $f(n)$ non cresce più velocemente di $g(n)$.



Notazione asintotica

- **Esempio**

- $10n^3 + 2n^2 + 7 = O(n^3)$
- Dimostriamolo:
  - Occorre provare che

    $$\exists\, c, n_0 : 10n^3 + 2n^2 + 7 \leq cn^3,\ \forall\, n \geq n_0$$

  - Si ha: $10n^3 + 2n^2 + 7 \leq 10n^3 + n^3 + n^3 = 12n^3,\ \forall\, n \geq 2.$
  - Quindi la diseguaglianza è soddisfatta per $c = 12$ e $n_0 = 2$.
  - Le costanti c ed n_0 non sono uniche….

continua

Notazione asintotica

- $10n^3 + 2n^2 + 7 = O(n^3)$
- `possiamo ottenere una costante c piu` piccola di 12 a patto di prendere un n_0 piu` grande
  - Occorre sempre che $10n^3 + 2n^2 + 7 \leq cn^3, \ \forall n \geq n_0$
  - Notiamo che $2n^2 + 7 \leq 2n^2 + n^2 = 3n^2 \leq n^3, \ \forall n \geq 3$
  - Quindi $10n^3 + 2n^2 + 7 \leq 11n^3, \ \forall n \geq 3$

e la diseguaglianza è soddisfatta anche per $c = 11$ e $n_0 = 3$.

Ordine di grandezza dei polinomi

$$\boxed{a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0 = O(n^k)}$$

Infatti

$$\begin{aligned}
a_k n^k + a_{k-1} n^{k-1} &+ \cdots + a_1 n + a_0 \\
&\leq |a_k| n^k + |a_{k-1}| n^{k-1} + \cdots + |a_1| n + |a_0| \\
&\leq |a_k| n^k + |a_{k-1}| n^k + \cdots + |a_1| n^k + |a_0| n^k \\
&= (|a_k| + |a_{k-1}| + \cdots + |a_1| + |a_0|) n^k \\
&= c n^k
\end{aligned}$$

$c = |a_k|+|a_{k-1}|+\ldots+|a_0|$ ed $n_0=1$

Esempi

$$n^3 + 100n + 200 = O(n^3)$$

$$20n^3 + n^5 + 100n = O(n^5)$$

$$10n^2 + n^{5/2} + 7n = O(n^{5/2})$$

$$10n+3n^7+5n^6+9n^3+34n^2+22n^5+n^{8/3}+4n^{7/2}+23n^{11/2} = O(n^7)$$

Tempo lineare O(n)

Esempio: L'algoritmo Max cerca il massimo nell'array a

- Tempo di esecuzione di Max(a):

$T(n)=c_1+c_2n+(c_3+c_4)(n-1)+c_5=(c_2+c_3+c_4)n+c_1-c_3-c_4+c_5=an+b=O(n)$

  - n è la lunghezza dell'array

Max(a):	Costo	num. volte
max=a[0];	c_1	1
FOR(i=1;i<n;i++)	c_2	n
IF(a[i]>max)	c_3	n-1
max=a[i];	c_4	n-1 (nel caso pessimo)
RETURN max;	c_5	1

Tempo quadratico $O(n^2)$

- Esempio: L'algoritmo Intersect inserisce nell'array c gli elementi che compaiono sia in a che in b. Si assuma che gli elementi in ciascuno degli array a e b siano distinti e che entrambi gli array a e b abbiano dimensione uguale ad n.

```
Intersect(a,b,c):
 k=0;
 FOR(i=0;i<n;i++){              //  eseguito n+1 volte
     FOR(j=0;j<n;j++){          //  eseguito n(n+1) volte
       IF(a[i]=b[j])            //   eseguito n^2 volte
           {c[k]=a[i];k=k+1;}   //eseguito n^2 volte (nel caso pessimo)
```

Tempo di esecuzione $T(n)=O(n^2)$

Tempo logaritmico O(log n)

- Esempio: Algoritmo di ricerca binaria: cerca un numero x in un array a ordinato (x potrebbe anche non essere presente)

```
RicercaBinaria(a,x):
primo=0;
ultimo=n-1;
WHILE(primo<=ultimo)    //al massimo ⌊log n⌋+2 volte
  centro=(primo+ultimo)/2; //il corpo al massimo ⌊log n⌋+1 volte
  IF( x< centro)  ultimo=centro-1;
    ELSE IF( x> centro)  primo=centro+1;
  ELSE RETURN centro;
}
RETURN -1;
```

Tempo di esecuzione $T(n)=O(\log n)$

continua

Tempo logaritmico O(log n)

- Inizialmente spazio di ricerca di n elementi
- Dopo 1 test $\left\lfloor \frac{n}{2} \right\rfloor$ elementi
- Esempio n pari: 1 3 4 5 9 11 14 21. Primo confronto con 5. Nel caso pessimo la ricerca continua in 9 11 14 21 (4 elementi)
- Esempio n dispari: 2 4 6 8 9 11 14 20 23. Primo confronto con 9. La ricerca continua in uno spazio di 4 elementi
- dopo 2 test $\left\lfloor \left\lfloor \frac{n}{2} \right\rfloor \frac{1}{2} \right\rfloor = \left\lfloor \frac{n}{4} \right\rfloor$ elementi,… ,dopo i test $\left\lfloor \left\lfloor \frac{n}{2^{i-1}} \right\rfloor \frac{1}{2} \right\rfloor = \left\lfloor \frac{n}{2^{i}} \right\rfloor$ elementi
- While termina dopo k iterazioni per k tale che $\left\lfloor \frac{n}{2^k} \right\rfloor = 0$

$\left\lfloor \frac{n}{2^k} \right\rfloor = 0 \Rightarrow n < 2^k \Rightarrow k > \log n$. Siccome $\lfloor \log n \rfloor \le \log n < \lfloor \log n \rfloor + 1$,

allora l'intero k piu` piccolo per cui $k > \log n$ e` $\lfloor \log n \rfloor + 1$

Ordine di grandezza del logaritmo

- Dimostriamo che $\log_2 n = O(n)$
- Dobbiamo dimostrare che esistono $c>0$ ed un intero n_0 tali che $\log_2 n \leq cn$ per ogni $n \geq n_0$

- Dimostriamo per induzione che $\log_2 n \leq n$ per ogni $n \geq 1$
- Base: per $n=1$ la disuguaglianza e` soddisfatta perche' si ha $\log_2 1 = 0 \leq 1$
- Passo induttivo: Supponiamo che la disuguaglianza sia soddisfatta per n e dimostriamo che e` soddisfatta per n+1

$$\log_2(n+1) \leq \log_2(n+n) = \log_2(2n) = \log_2 2 + \log_2 n = 1 + \log_2 n$$

Per ipotesi induttiva $\log_2 n \leq n$ per cui $\log_2(n+1) \leq n+1$

Quindi $\log_2 n = O(n)$ con $c=1$ ed $n_0=1$

E se il logaritmo non è in base 2?

Non cambia niente. Sappiamo infatti che

$$\log_a n = (\log_a 2)(\log_2 n),$$

per $a \neq 1$; inoltre abbiamo già provato che $\log_2 n \leq n$, pertanto

$$\log_a n = (\log_a 2)(\log_2 n) \leq (\log_a 2)n \;\Rightarrow\; \log_a n = O(n)$$

E se invece di $\log n$ abbiamo $\log n^a,\ a > 1?$ Non cambia niente. Sappiamo che $\log n^a = a \log n$ e $\log n \leq cn$, per opportuna costante $c > 0$, quindi

$$\log n^a = a \log n \leq acn \quad \text{ovvero} \quad \log n^a = O(n)$$

Logaritmo al confronto con radice

- Si ha

$$\boxed{\forall \text{ costanti } a > 0, b > 0, k > 0 \;\; \text{vale} \;\; \log^a n^b = O(n^k)} \qquad (2)$$

Ad esempio, $\log^5 n^6 = O(\sqrt[3]{n})$

Basta infatti porre $a = 5, b = 6, k = 1/3$ nella (2)

Altro esempio: $\log^2 n^{10} = O(\sqrt[6]{n})$

Basta porre $a = 2, b = 10, k = 1/6$ nella (2)

- Vogliamo dimostrare che $\log^a n^b = O(n^k)$, $\forall\ a > 0,\ b > 0,\ k > 0$ constanti
- Consideriamo prima il caso $a = 1$. In questo caso abbiamo

$$\log n^b = b \log n = b\frac{1}{k}k \log n = \frac{b}{k} \log n^k.$$

- Abbiamo già dimostrato che $\log m \leq m$ per ogni $m \geq 1$, per cui ponendo $m = n^k$ abbiamo $\log n^k \leq n^k$ e di conseguenza

$$\log n^b = b \log n = b\frac{1}{k}k \log n = \frac{b}{k} \log n^k \leq \frac{b}{k} n^k.$$

Si ha quindi che per $c = \frac{b}{k}$ ed $n_0 = 1$, risulta

$$\log n^b \leq cn^k,$$

per ogni $n \geq n_0$. Ne consegue che $\log n^b = O(n^k)$.

- Vogliamo dimostrare che $\log^a n^b = O(n^k)$, $\forall\ a > 0,\ b > 0,\ k > 0$ constanti

- Ora consideriamo il caso in cui a è un qualsiasi numero positivo . Nella slide precedente abbiamo dimostrato che per $c' = \frac{b}{k'}$ ed $n_0 = 1$, risulta

$$\log n^b \leq c' n^{k'},$$

per ogni $n \geq n_0$. Poiché la disuguaglianza vale per ogni k' positivo allora posso prendere $k' = \frac{k}{a}$ e ottengo che per $c' = \frac{b}{k/a}$ ed $n_0 = 1$, risulta

$$\log n^b \leq c' n^{k/a},$$

per ogni $n \geq n_0$.

Ne consegue che

$$(\log n^b)^a \leq (c' n^{k/a})^a = (c')^a (n^{k/a})^a = (c')^a n^k,$$

per ogni $n \geq n_0$.

Prendendo quindi $c = (c')^a$ ed $n_0 = 1$ si ha che $(\log n^b)^a = O(n^k)$.

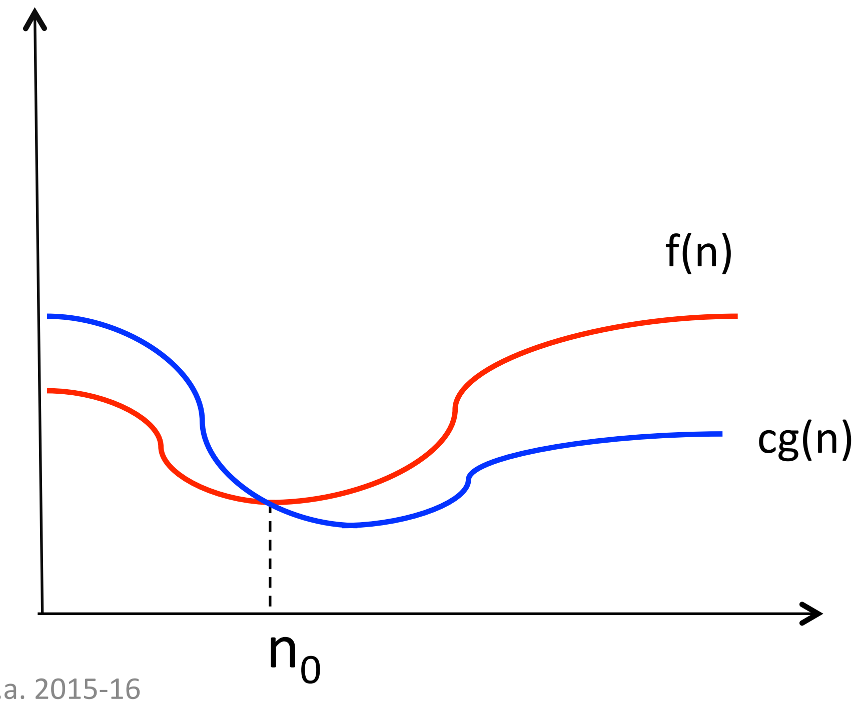
Espressione O	nome
$O(1)$	costante
$O(\log\log n)$	$\log\log$
$O(\log n)$	logaritmico
$O(\sqrt[c]{n}),\ c > 1$	sublineare
$O(n)$	lineare
$O(n \log n)$	$n \log n$
$O(n^2)$	quadratico
$O(n^3)$	cubico
$O(n^k)\ (k \geq 1)$	polinomiale
$O(a^n)\ (a > 1)$	esponenziale

Notazione asintotica
limite inferiore

- Date $f : n \in N \rightarrow f(n) \in R^+$, $g : n \in N \rightarrow g(n) \in R^+$, scriveremo $f(n) = \Omega(g(n))$

$\Leftrightarrow \exists\, c > 0,\ \exists\, n_0$ tale che $f(n) \geq cg(n),\ \forall\, n \geq n_0$

Informalmente, $f(n) = \Omega(g(n))$ se $f(n)$ non cresce più lentamente di $g(n)$.



Notazione asintotica
Limite inferiore

- **Esempio**

- $10n^3 - 2n^2 + 7 = \Omega(n^3)$
- Dimostriamo:
  - Occorre provare che

$$\exists\, c>0, n_0 : 10n^3 - 2n^2 + 7 \geq cn^3,\ \forall\, n \geq n_0$$

  - Dividiamo tutto per n^3: $10 - 2/n + 7/n^3 \geq c$
  - $10 - 2/n + 7/n^3 > 10 - 2 = 8,\ \forall\, n \geq 1.$
  - Quindi la diseguaglianza è soddisfatta per $c \leq 8$ e $n_0 = 1$.

- Vogliamo dimostrare che $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0 = \Omega(n^k)$, $a_k > 0$
- Dobbiamo dimostrare che esistono $c > 0$ ed n_0 tali che

$$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0 \geq c n^k, \quad \forall \; n \geq n_0.$$

Dividendo entrambi i membri della disuguaglianza per n^k otteniamo

$$a_k + \frac{a_{k-1}}{n} + \frac{a_{k-2}}{n^2} + \ldots + \frac{a_0}{n^k} \geq c.$$

Osserviamo che l'espressione a sinistra è maggiore o uguale di

$$\begin{aligned} a_k - \left|\frac{a_{k-1}}{n}\right| - \left|\frac{a_{k-2}}{n^2}\right| - \ldots - \left|\frac{a_0}{n^k}\right| &\geq a_k - \left|\frac{a_{k-1}}{n}\right| - \left|\frac{a_{k-2}}{n}\right| - \ldots - \left|\frac{a_0}{n}\right| \\ \geq a_k - \frac{k}{n} \cdot max_{0 \leq i \leq k-1}\{|a_i|\} &\geq a_k - \frac{k}{n_0} \cdot max_{0 \leq i \leq k-1}\{|a_i|\}, \; \forall \; n \geq n_0 \end{aligned}$$

Se scegliamo un valore di n_0 tale che $a_k - \frac{k}{n_0} \cdot max_{0 \leq i \leq k-1}\{|a_i|\} > 0$ allora possiamo porre $c = a_k - \frac{k}{n_0} \cdot max_{0 \leq i \leq k-1}\{|a_i|\}$ e abbiamo finito.

Notiamo che $a_k - \frac{k}{n_0} \cdot max_{0 \leq i \leq k-1}\{|a_i|\} > 0$ per $n_0 > \frac{k}{a_k} \cdot max_{0 \leq i \leq k-1}\{|a_i|\}$.

L'intero n_0 più piccolo per cui vale ciò è $n_0 = \lfloor \frac{k}{a_k} \cdot max_{0 \leq i \leq k-1}\{|a_i|\} \rfloor + 1$.

Abbiamo trovato i valori c ed n_0 desiderati.