Costo Computazionale e Notazione Asintotica

corso di laurea in **Matematica**

Informatica Generale, **Ivano Salvo**

Lezione **4(a)**, **[6/10/23]**

SAPIENZA
UNIVERSITÀ DI ROMA

Costo Computazionale

Saremo interessati a valutare il costo computazionale degli algoritmi (**analisi** [computazionale] **degli algoritmi**).

Usualmente ci concentreremo sul **tempo di calcolo**, o meglio, sul numero di operazioni che necessita un algoritmo.

Occasionalmente, analizzeremo la **quantità di memoria**.

In altri contesti, potrebbe essere interessante valutare **altre risorse computazionali** (numero di processori, numero di messaggi scambiati, etc.)

Le misure di complessità sono sempre **funzione** della **dimensione dell'input** (a volte questo può trarre in inganno) e sono **funzioni positive** e **non-decrescenti** (quasi ovunque).

Ci interesserà (quasi) sempre il **comportamento asintotico**: cioè il costo dell'esecuzione dell'algoritmo al **crescere della dimensione** dell'**input**.

Costo Computazionale: esempio

Un esecutore che **non ha** il predecessore tra le sue **abilità elementari** deve compiere n passi per calcolare n-1.

Quindi la funzione in basso a sinistra che chiama ***n* volte** `pred`, calcolerà almeno $n + n + (n\text{-}1) + (n\text{-}2) + \ldots + 1 = n(n+1)/2 + n$ operazioni di +1 (formula di Gauss).

Il **costo computazionale dipende** dal **modello di calcolo**!

Tuttavia, occorre distinguere la complessità di un **problema** da quella di un **algoritmo**. La somma è cmq lineare!

A destra un algoritmo che fa la somma con $2n$ successori.

```
def somma(int m, int n):
    while n!=0:
        m, n = m+1, pred(n)
    return j
```

```
def somma(int m, int n):
    i = 0
    while i!=n:
        m, i = m+1, i+1
    return j
```

Costo Computazionale: riflessioni

Nel **nostro modello di calcolo**, calcolare la funzione $f(n)$ avrà sempre **complessità almeno proporzionale a** $f(n)$. Perché?

Qual è la complessità di calcolare $m+n$ **dell'algoritmo di somma** delle elementari **per numeri a più cifre** visto prima?

È proporzionale alla **lunghezza della rappresentazione** dei due numeri, che è $\log_b n$, che è **enormente minore** della somma unaria, che è viceversa proporzionale ad n.

Nel seguito, considereremo modelli di calcolo più realistici, in cui le operazioni aritmetiche base (+, *, -) hanno costo **al più logaritmico** rispetto al valore degli operandi (quindi **lineare nella dimensione** della **rappresentazione**)

Caso ottimo, medio, pessimo (1)

Nei problemi visti finora, il **numero di operazioni** è quasi sempre **prevedibile** e dipende **funzionalmente dall'input**, che è costituito da numeri naturali.

Spesso, il **tempo di calcolo** può dipendere dalla **struttura dell'input** e non solo dalla sua dimensione.

⏵ **Esempio**: se cerco un documento in una pila di fogli alla rinfusa, posso decidere di guardarli uno a uno: il tempo dipenderà dal fatto che il documento sia vicino alla cima (caso **ottimo**), in fondo alla pila, o addirittura assente (caso **pessimo**)

Oppure potrei essere interessato al costo **medio**, ossia quale sia il valore atteso della ricerca. Questa dipende da una **distribuzione di probabilità** sulle istanze di input.

⏵ **Esempio**: se i fogli sono alla rinfusa, ma i documenti recenti sono tendenzialmente sopra e io ricerco prevalentemente documenti recenti, il numero medio di fogli analizzati sarà minore di $n/2$. Viceversa se tutto è uniformemente alla rinfusa…

Caso ottimo, medio, pessimo (2)

Quali sono il caso ottimo e pessimo del **programma** che **genera** i **fattori primi** per calcolare il **massimo fattore primo**?

Caso ottimo: n è un numero "molto" composto, con molti fattori primi "piccoli". Ad esempio se $n = 2^k$, allora il programma genererà k volte il fattore primo 2, in sole k **iterazioni.** In questo caso, $k = \log_2 n$.

Caso pessimo: n è un numero primo. In tal caso, l'algoritmo testa la divisibilità per tutti i numeri tra 2 e $\sqrt{n}$ e quindi termina in $\sqrt{n}$ iterazioni.

- **Esempio**: Per **1024** si fanno **10** iterazioni, per **1021** se ne fanno **32** (1021 senza ottimizzazione $p^2 < n$)

Oppure: per 2^k se ne fanno k, se 2^k**-1** è **primo** se ne fanno $2^{k/2}$.

Caso medio: decisamente difficile da determinare, ma correlato al numero medio di fattori primi di n

qui contiamo il numero delle iterazioni, poi entreremo in maggiori dettagli

Notazione asintotica

In molti casi, calcolare il numero esatto delle operazioni necessarie è **molto difficile** o addirittura **impossibile**.

Di norma, ci si accontenta di stabilire l'**ordine di grandezza**.

◗ **Esempio**: l'algoritmo di **somma unaria**, per calcolare $m+n$, esegue $2n$ operazioni di successore e n confronti (per valutare la guardia del ciclo).

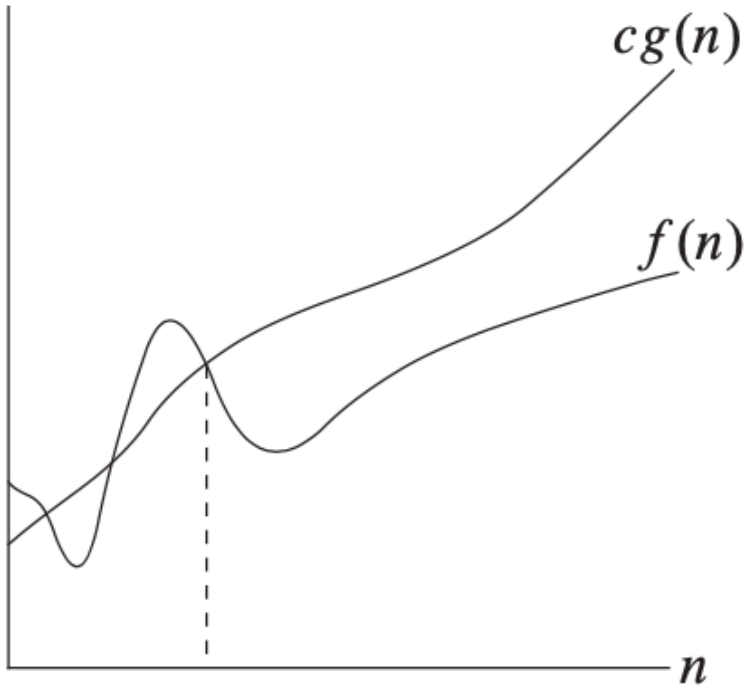
Ci accontenteremo di dire che l'algoritmo di somma unaria è **lineare** in n, dimenticando le costanti moltiplicative.

Questo permette anche di **astrarre sul tipo delle azioni elementari** (fatte in tempo **costante**, ma non **necessariamente uguale**) e sommare il loro numero tra loro.

◗ **Nota storica**: fino a 30 anni fa, eseguire una assegnazione, un confronto, una somma o un prodotto aveva **tempi molto diversi** nei calcolatori e ad esempio era preferibile un algoritmo che minimizzava il numero dei prodotti.

Limite Asintotico Superiore: $\mathcal{O}$

Definizione: Date due funzioni $f(n)$ e $g(n)$ a valori positivi, si dice che $f(n)$ è $\mathcal{O}(g(n))$ (oppure $f(n) \in \mathcal{O}(g(n))$) se esistono due costanti c e n_0 **positive** tali che $0 \leq f(n) \leq c\, g(n)$ per ogni $n \geq n_0$.



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

Nota: $\mathcal{O}(g(n))$ è un **insieme di funzioni**, ma spesso si scrive $f(n) = \mathcal{O}(g(n))$, che si legge "$f(n)$ **è un o grande di** $g(n)$", oppure anche "$g(n)$ **domina (asintoticamente)** $f(n)$"

Esempi di $\mathcal{O}(f(n))$

Esempio: Sia $f(n) = 44n+88$. Abbiamo che:

- $f(n)$ è $\mathcal{O}(n^2)$, infatti è sufficiente trovare n_0 tale che $n^2 > 44n+88$. È sufficiente risolvere questa disequazione, che è soddisfatta per ogni valore maggiore di $n_0 = 42$.

  Si può anche (più semplicemente), **giocare sulla libertà di scegliere la costante** c, per cui preso c=132, è evidente che $132n^2 > 44n+88$ per ogni valore di $n > n_0 = 0$.

- $f(n)$ è anche $\mathcal{O}(n)$, infatti qui, scegliendo ancora c=132, è evidente che $132n > 44n+88$ per ogni valore di $n > n_0 = 1$

In generale saremo interessati a trovare **la più piccola funzione** g tale che $f(n)$ è $\mathcal{O}(g(n))$. Nel nostro caso, $g(n)=n$ è un limite superiore **stretto** per $f(n)$.

Esempi di $\mathcal{O}(f(n))$

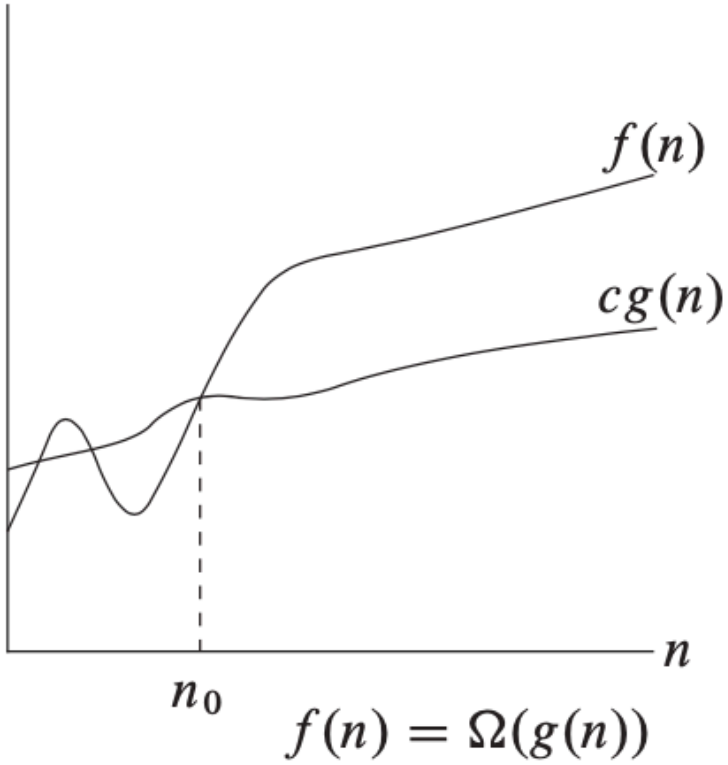
Esempio: Sia $f(n) = n^2+4n$. Abbiamo che:

- $f(n)$ è $\mathcal{O}(n^2)$, infatti $cn^2 \geq n^2+4n$ per ogni n se prendo $c > 5$. Alternativamente, $cn^2 \geq n^2+4n$ implica $(c\text{-}1)\ n^2 \geq 4n$, cioè $(c\text{-}1)\ n \geq 4$ (per $n>0$) che è soddisfatta per $n \geq 4\ (c\text{-}1)$.
- Viceversa $f(n)$ **non è** $\mathcal{O}(n)$ in quanto la disuguaglianza $cn \geq n^2+4n$ diventa falsa per n abbastanza grande. Banalmente per $n > c$.

Morale: avete ampia libertà nel dimostrare che una funzione $f(n)$ è $\mathcal{O}(g(n))$, ma usualmente è meglio non perdersi in dettagli e fare **maggiorazioni "tranquille"**.

Limite Asintotico Inferiore: Ω

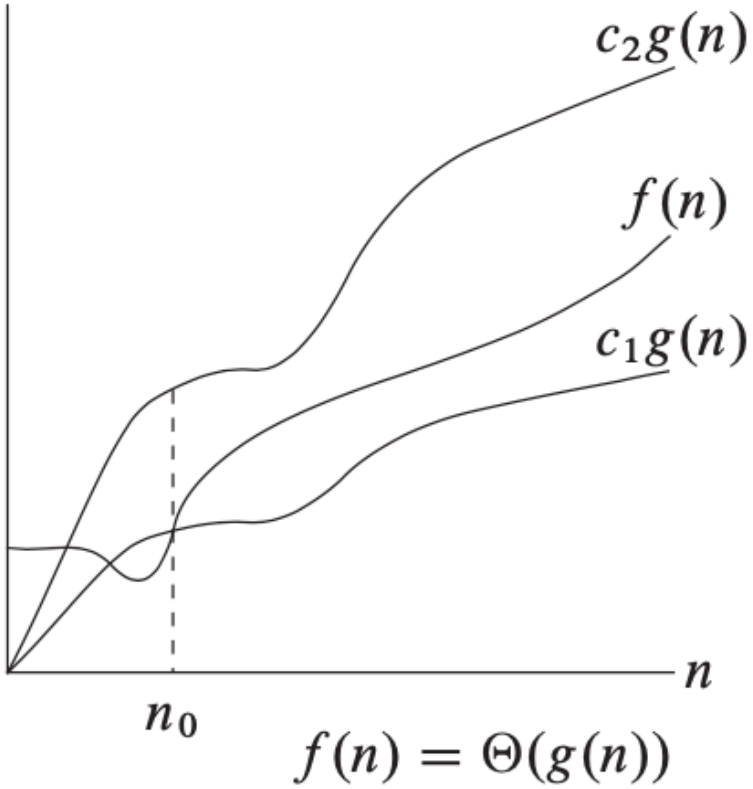
Definizione: Date due funzioni $f(n)$ e $g(n)$ a valori positivi, si dice che $f(n)$ è $\Omega(g(n))$ (oppure $f(n) \in \Omega(g(n))$) se esistono due costanti c ed n_0 **positive** tale che $f(n) \geq c\, g(n)$ per ogni $n \geq n_0$.



Nota: $\Omega(g(n))$ è un **insieme di funzioni**, ma spesso si scrive $f(n) = \Omega(g(n))$, che si legge "$f(n)$ **è un omega grande di** $g(n)$", oppure anche "$g(n)$ **è dominata (asintoticamente)** $f(n)$"

Limite Asintotico Stretto: θ

Definizione: Date due funzioni $f(n)$ e $g(n)$ a valori positivi, si dice che $f(n)$ è $\Theta(g(n))$ (oppure $f(n) \in \Theta(g(n))$) se esistono tre costanti c_1, c_2 ed n_0 **positive** tale che $c_1 g(n) \leq f(n) \leq c_2 g(n)$ per ogni $n \geq n_0$.



Nota: $f(n) = \Theta(g(n))$, che si legge "$f(n)$ **è teta di** $g(n)$", significa che f(n) e g(n) hanno **lo stesso ordine di grandezza** asintotico, **a meno di costanti moltiplicative**.

Significato Pratico di questa Teoria

Immaginiamo di avere 2 calcolatori L e V. L è lento e fa 10^6 operazioni al secondo, mentre V è veloce e arriva a 10^9.

Consideriamo due algoritmi $\mathcal{A}_1$, $\mathcal{A}_2$ che risolvono lo stesso problema con rispettivamente $T(\mathcal{A}_1)=\mathbf{10}n^2$ e $T(\mathcal{A}_2)=\mathbf{100}n\log_2 n$ operazioni.

Dovendo scegliere, mi conviene usare l'algoritmo $\mathcal{A}_1$ sul cacolatore V, oppure $\mathcal{A}_2$ su L?

Calcoliamo i tempi…

Dipende ovviamente dalle dimensioni del problema da risolvere. Facciamo **3** casi: $n=10^3$, $n=10^6$ e infine $n=10^9$ (ricordiamo $\log_2 1000 \approx 10$ e quindi $\log_2 10^{3n} \approx 10n$).

Caso 1: $n=10^3$

$$\mathcal{A}_1 \text{ su } V = \frac{10\cdot(10^3)^2}{10^9}\, sec = \frac{1}{10^2}\, sec = \textbf{.01 sec}$$

$$\mathcal{A}_2 \text{ su } L = \frac{100\cdot 10^3 \log 10^3}{10^6}\, sec = 1\ sec = \textbf{10 sec}$$

Caso 2: $n=10^6$

$$\mathcal{A}_1 \text{ su } V = \frac{10\cdot(10^6)^2}{10^9}\, sec = 10^4 sec = \textbf{2h46min20sec}$$

$$\mathcal{A}_2 \text{ su } L = \frac{100\cdot 10^6 \log 10^6}{10^6}\, sec = 2\cdot 10^3\ sec = \textbf{33min20sec}$$

Caso 1: $n=10^9$

$$\mathcal{A}_1 \text{ su } V = \frac{10\cdot(10^9)^2}{10^9}\, sec = 10^{10}\ sec = \textbf{15 anni e 2 mesi}$$

$$\mathcal{A}_2 \text{ su } L = \frac{100\cdot 10^9 \log 10^9}{10^6}\, sec = 30\cdot 10^5\ sec \approx \textbf{35 giorni}$$

Evoluzione Tecnologica e algoritmi

Immaginiamo di avere 4 algoritmi, $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, e $\mathcal{A}_4$ che risolvono un certo problema P.

Supponiamo che il numero di operazioni sia rispettivamente: $T(\mathcal{A}_1)=\log_{10} n$, $T(\mathcal{A}_2)=n$, $T(\mathcal{A}_3)=n^2$, e $T(\mathcal{A}_4)=10^n$.

Supponiamo che, in un fissato tempo t ciascuno di essi riesca a risolvere istanze di dimensione rispettivamente n_1, n_2, n_3, e n_4. Quanto migliora la situazione il **progresso tecnologico**?

La risposta in tabella!

Il programma esponenziale fa progressi additivi

	v	$\mathbf{10}v$	$\mathbf{100}v$	$\mathbf{1000}v$	kv
$\mathcal{A}_1$	n_1	${n_1}^{10}$	${n_1}^{100}$	${n_1}^{1000}$	${n_1}^{k}$
$\mathcal{A}_2$	n_2	$10 \cdot n_2$	$100 \cdot n_2$	$1000 \cdot n_2$	$k n_1$
$\mathcal{A}_3$	n_3	$3.16 \cdot n_3$	$10 \cdot n_3$	$31.6 \cdot n_3$	$\sqrt{k} \cdot n_3$
$\mathcal{A}_4$	n_4	$n_4 + 1$	$n_4 + 2$	$n_4 + 3$	$n_4 + \log_{10} k$

Algebra della notazione asintotica(1)

- **Relazioni tra** $\mathcal{O}$, Θ, *e* Ω: direttamente dalle definizioni:

$f(n)$ è $\mathcal{O}(g(n))$ se e solo se $g(n)$ è $\Omega(f(n))$

$f(n)$ è $\Theta(g(n))$ **se e solo se** $f(n)$ è $\mathcal{O}(g(n))$ e $f(n)$ è $\Omega(g(n))$

- **Costanti moltiplicative:** ovviamente dalla definizione, si ha anche (in particolare per c=1, quindi sono **riflessive**):

$\forall\, c>0$, $c\,f(n)$ è $\Theta(f(n))$, $\mathcal{O}(f(n))$ e $\Omega(f(n))$

- **Proprietà Transitive:** Valgono le seguenti (**dimostrare**!)

$f(n) \in \mathcal{O}(g(n))$ e $g(n) \in \mathcal{O}(h(n))$ allora $f(n) \in \mathcal{O}(h(n))$

$f(n) \in \Omega(g(n))$ e $g(n) \in \Omega(h(n))$ allora $f(n) \in \Omega(h(n))$

$f(n) \in \Theta(g(n))$ e $g(n) \in \Theta(h(n))$ allora $f(n) \in \Theta(h(n))$

- **Notazione Asintotica e Ordinamento:** c'è un'evidente legame tra notazione $\mathcal{O}$ e $\leq$, notazione Ω e $\geq$, e notazione Θ e $=$, tuttavia, **attenzione**, ci sono funzioni che non stanno in nessuna relazione. **Es:** n e $n^{1+\sin n}$ ($1+\sin n$ oscilla tra 0 e 2).

Algebra della notazione asintotica(2)

- **Somma**: $f(n)+g(n)$ è $\Theta(\max(f(n), g(n))$.

Infatti: $\max(f(n), g(n)) \leq f(n)+g(n) \leq 2 \max(f(n), g(n))$, quindi è $\Theta(\max(f(n),g(n))$. Quindi abbiamo anche:

$$f(n)+g(n) \text{ è } \mathcal{O}(\max(f(n),g(n)) \quad \text{e} \quad f(n)+g(n) \text{ è } \Omega(\max(f(n),g(n))$$

- **Polinomi**: n^k è $\mathcal{O}(n^{k+1})$, ma n^{k+1} non è $\mathcal{O}(n^k)$

Di conseguenza, applicando questa regola e quella della somma (più volte) ho che per due polinomi P e P' valgono:

$$P(n) \in \mathcal{O}(P'(n)) \Leftrightarrow \text{grado}(P) \leq \text{grado}(P')$$
$$P(n) \in \Omega(P'(n)) \Leftrightarrow \text{grado}(P) \geq \text{grado}(P')$$
$$P(n) \in \Theta(P'(n)) \Leftrightarrow \text{grado}(P) = \text{grado}(P')$$
$$P(n) \in \Theta(n^k) \Leftrightarrow \text{grado}(P) = k$$

Algebra della notazione asintotica(3)

◗ **Prodotto**: Se $f_1(n)$ è $\mathcal{O}(g_1(n))$ e $f_2(n)$ è $\mathcal{O}(g_2(n))$ allora $f_1(n)\cdot f_2(n)$ è $\mathcal{O}(g_1(n)\cdot g_2(n))$.

Infatti: se Se $f_1(n)$ è $\mathcal{O}(g_1(n))$ e $f_2(n)$ è $\mathcal{O}(g_2(n))$, allora esistono 4 costanti c_1, c_2, n_1, n_2 **positive** tali che:

$$f_1(n) \leq c_1\, g_1(n) \text{ per ogni } n \geq n_1$$

$$f_2(n) \leq c_2\, g_2(n) \text{ per ogni } n \geq n_2$$

da cui (moltiplicando i due membri, asintoticamente positivi)

$$f_1(n)\cdot f_2(n) \leq c_1 c_2\, g_1(n) \cdot g_2(n) \text{ per ogni } n \geq \max\{n_1, n_2\}$$

Simmetricamente si può far vedere che:

$$f_1(n) \text{ è } \Omega(g_1(n)) \text{ e } f_2(n) \text{ è } \Omega(g_2(n)) \Rightarrow f_1(n)\cdot f_2(n) \text{ è } \Omega\,(g_1(n)\cdot g_2(n))$$

e di conseguenza:

$$f_1(n) \text{ è } \Theta(g_1(n)) \text{ e } f_2(n) \text{ è } \Theta(g_2(n)) \Rightarrow f_1(n)\cdot f_2(n) \text{ è } \Theta\,(g_1(n)\cdot g_2(n))$$

Notazione Asintotica e limiti

Esiste una evidente relazione tra i limiti del rapporto tra due funzioni $f(n)$ e $g(n)$ e le notazioni asintotiche introdotte.

Teorema:

- Se $\lim_{n\to\infty} f(n)/g(n) = k > 0$ allora $f(n) \in \Theta(g(n))$
- Se $\lim_{n\to\infty} f(n)/g(n) = \infty$ allora $f(n) \in \Omega(g(n))$ e $f(n) \notin \mathcal{O}(g(n))$
- Se $\lim_{n\to\infty} f(n)/g(n) = 0$ allora $f(n) \in \mathcal{O}(g(n))$ e $f(n) \notin \Omega(g(n))$

Il converso non è vero perché ad esempio potremo avere che $f(n) \in \Theta(g(n))$ ma il limite non esiste (è finito ma non converge a nessun valore in particolare).

Esercizio: si riesce a dire qualcosa su liminf e limsup?

Alcuni esempi notevoli

I **logaritmi** appartengono **tutti alla stessa classe asintotica**.
Infatti: $\log_a n = \frac{log_b n}{log_b a}$, cioè logaritmi con basi diverse differiscono per una **costante moltiplicativa**.

I logaritmi **"schiacciano" la crescita**: Ad esempio, per ogni costante c, $\Theta(\log n^c) = \Theta(\log n)$, perché $\log n^c = c \log n$.

Ogni **funzione logaritmica è $\mathcal{O}(n^\varepsilon)$ per ogni $\varepsilon > 0$**. È sufficiente calcolare il limite $\lim_{n\to\infty} \log n / n^\varepsilon = 0$ (con de l'Hopital, per esempio, anche **se a rigore noi siamo sugli interi**…)

Viceversa, gli **esponenziali** "espandono" la crescita asintotica: ad es. $2^{2n} > c \cdot 2^n$ per ogni $n > \log_2 c$. Quindi ho che $2^n = \mathcal{O}(2^{2n})$, ma $2^n \notin \Theta(2^{2n})$.

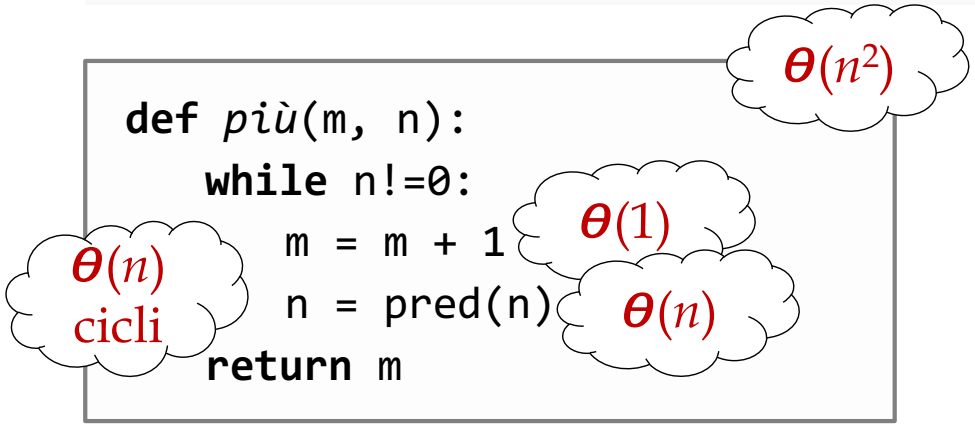
Attenti quindi che le **costanti a esponente contano**!

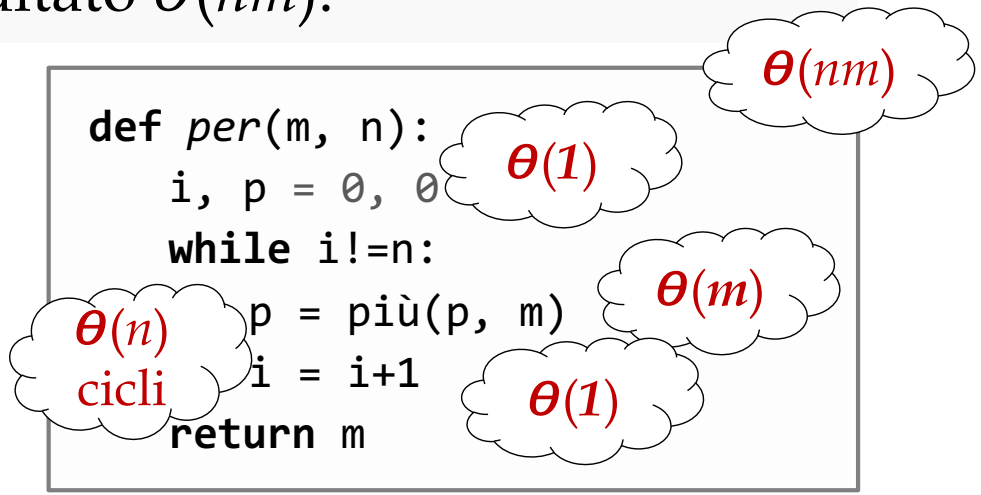
Pseudocodice e complessità

Vediamo come calcolare la complessità degli algoritmi.

a) algoritmo di **somma senza -1**: occorre ricordarsi che `pred(n)` costa un tempo proporzionale a n. Ma n decresce durante il ciclo: si fanno n iterazioni quindi n+(n-1)+…+2+1 operazioni, per un totale di $\frac{n(n+1)}{2}$ operazioni, più n incrementi di m. Il **fattore dominante** è quadratico, quindi $\Theta(n^2)$.

b) **prodotto**: occorre ricordare che `piu(p,m)` può essere fatta in $\Theta(m)$. Il ciclo `while` fa n iterazioni: si deve **moltiplicare** il **numero delle iterazioni** per il **costo di una iterazione**, che in questo caso **è sempre** $\Theta(m)$. Risultato $\Theta(nm)$.





That's all Folks!

corso di laurea in **Matematica**

Informatica Generale, **Ivano Salvo**

Lezione **4(a)** [**6/10/23**]

SAPIENZA
UNIVERSITÀ DI ROMA