

Algoritmi (modulo di laboratorio)

Corso di Laurea in Matematica

Roberto Cordone

DI - Università degli Studi di Milano



- Lezioni: Martedì 8.30 - 10.30 in aula 8 Mercoledì 10.30 - 13.30 in aula 309
Giovedì 16.30 - 18.30 in aula 307 Venerdì 10.30 - 12.30 in aula 4
- Ricevimento: su appuntamento (Dipartimento di Informatica)
- E-mail: roberto.cordone@unimi.it
- Pagina web: <http://homes.di.unimi.it/~cordone/courses/2024-algo/2024-algo.html>
- Sito Ariel: <https://mgoldwurmasd.ariel.ctu.unimi.it>

Lezione 3: Complessità computazionale

Milano, A.A. 2023/24

Complessità di un algoritmo (costo)

Applicare un algoritmo significa

- eseguire una sequenza finita di operazioni elementari (**passi**)
- manipolare una sequenza finita di simboli (**celle di memoria**), che include I all'inizio, S alla fine e risultati parziali nei passi intermedi

Per risolvere un'istanza I con un algoritmo A , quindi si paga un costo

- **temporale** $T_A(I)$, pari al **numero di passi eseguiti**
- **spaziale** $S_A(I)$, pari al **numero massimo di celle usate in un passo**

È intuitivo che questi costi (**complessità**) dipendono

- dalle operazioni elementari disponibili (modello computazionale)
- dai simboli disponibili (alfabeto)

ma si dimostra che la dipendenza non è fortissima

Comunque, useremo quasi sempre la macchina RAM

Confronto fra algoritmi (1)

Ora vogliamo definire il costo degli algoritmi che risolvono un problema P in modo che **A sia meglio di A' quando impiega meno tempo** (o spazio)

Per una singola istanza I è facile:

$$A(I) \preceq A'(I) \Leftrightarrow T_A(I) \leq T_{A'}(I)$$

Vorremmo estendere il confronto da singole istanze all'intero problema P , istituendo una **relazione di ordine debole** dotata di:

- 1 **riflessività**: $A \preceq A$
- 2 **transitività**: $A \preceq A'$ e $A' \preceq A'' \Rightarrow A \preceq A''$
- 3 **completezza**: $A \not\preceq A' \Rightarrow A' \preceq A$

per ogni terna di algoritmi A , A' e A'' che risolvono P

Confronto fra algoritmi (2)

Ci sono tre definizioni naturali per la relazione d'ordine $A \preceq A'$

① **su tutte le istanze:** $T_A(I) \leq T_{A'}(I)$ per ogni $I \in \mathcal{I}_P$

- è molto complicata da verificare
- l'ordine non è quasi mai completo

A sarà migliore su alcune istanze, A' su altre

② **nel caso medio:** $E[T_A(I)] \leq E[T_{A'}(I)]$

- richiede di considerare **tutte le istanze**
- richiede una **distribuzione di probabilità delle istanze**
- richiede **calcoli complicati**

È una buona definizione, ma complicata e in parte arbitraria

③ **nel caso pessimo:** $\max_{I \in \mathcal{I}_P} T_A(I) \leq \max_{I \in \mathcal{I}_P} T_{A'}(I)$

- spesso è facile identificare le istanze peggiori
- fornisce un **limite superiore**, che è un'informazione comunque utile
- in alcuni problemi il **caso pessimo è abbastanza frequente** e quindi la **complessità nel caso pessimo è simile a quella nel caso medio** (ad es., l'insuccesso in una ricerca)

È una definizione sbilanciata, ma utile in pratica

Complessità e dimensione (1)

Caso medio e caso pessimo hanno però un difetto fondamentale:

$$\sup_{I \in \mathcal{I}_P} T_A(I) = +\infty$$

cioè **non esiste un tempo massimo su \mathcal{I}_P e spesso neppure medio**, perché **il problema include infinite istanze, senza limite sul tempo di risoluzione**

Si può legare il tempo $T_A(I)$ alla **dimensione $|I|$** dell'istanza I , definita

- secondo la teoria, come **numero di simboli della codifica di I**
(criterio di *costo logaritmico*)
- in pratica, attraverso un indice dal significato concreto (o più indici)
 - se il problema riguarda insiemi, il **numero di elementi (n)**
 - se il problema riguarda relazioni (grafi), il **numero di elementi/nodi (n) e/o il numero di coppie in relazione/archi (m)**
(criterio di *costo uniforme*)

Complessità e dimensione (2)

Definita la dimensione di ogni istanza

- si considerano le istanze di ogni **dimensione n fissata**:

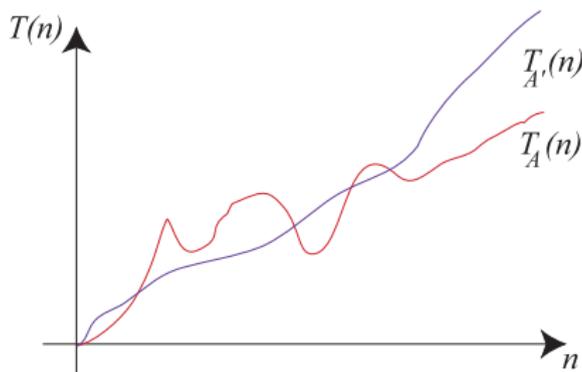
$$\mathcal{I}_P^{(n)} = \{I \in \mathcal{I}_P : |I| = n\}$$

- si determina il **caso medio** o il **caso pessimo** per ciascuna dimensione

$$T_A(n) = \frac{\sum_{I \in \mathcal{I}_P^{(n)}} T_A(I)}{|\mathcal{I}_P^{(n)}|} \text{ oppure } T_A(n) = \max_{I \in \mathcal{I}_P^{(n)}} T_A(I) \text{ per ogni } n \in \mathbb{N}$$

- si confrontano le funzioni $T_A(n)$ per ogni dimensione n

Ma anche le funzioni $T_A(n)$ sono ordinate molto raramente



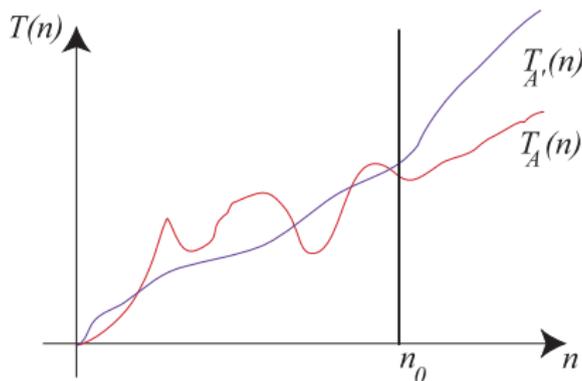
Complessità asintotica

Conta di più risolvere in fretta le istanze “grandi” che quelle “piccole”
Impiegare due giorni anziché uno è peggio che due secondi anziché uno

$$A \preceq A' \Leftrightarrow T_A(n) \leq T_{A'}(n) \text{ per ogni } n \geq n_0$$

A è meglio di A' quando usa meno tempo (o spazio)

- sulla peggior istanza di dimensione n
- per ogni valore di $n \geq n_0$, per un opportuno valore di $n_0 \in \mathbb{N}$



Notazione Θ

Data una **funzione approssimante** $f(n)$

$$T(n) \in \Theta(f(n))$$

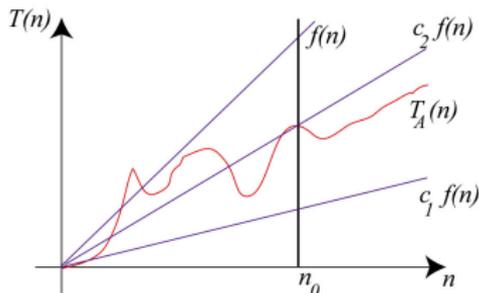
significa formalmente che

$$\exists c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N} : c_1 f(n) \leq T(n) \leq c_2 f(n) \text{ for all } n \geq n_0$$

dove c_1 , c_2 e n_0 sono indipendenti da n

$T(n)$ è “chiusa a sandwich” fra $c_1 f(n)$ e $c_2 f(n)$

- per qualche valore “piccolo” di c_1
- per qualche valore “grande” di c_2
- per ogni valore “grande” di n
- per qualche definizione di “piccolo” e “grande”



Asintoticamente, $f(n)$ stima $T(n)$ a meno di un fattore moltiplicativo:

- per istanze grandi, il tempo di calcolo è almeno e al massimo proporzionale ai valori della funzione $f(n)$

Notazione O

Data una **funzione approssimante** $f(n)$

$$T(n) \in O(f(n))$$

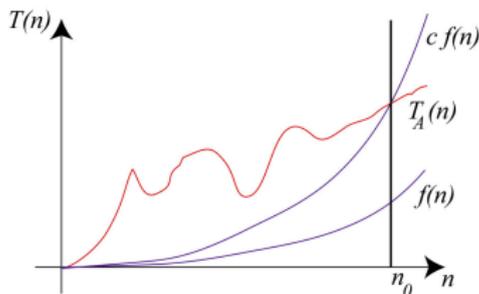
significa formalmente che

$$\exists c \in \mathbb{R}^+, n_0 \in \mathbb{N} : T(n) \leq c f(n) \text{ for all } n \geq n_0$$

dove c , e n_0 sono indipendenti da n

$T(n)$ è “dominata” da $cf(n)$

- per qualche valore “grande” di c
- per ogni valore “grande” di n
- per qualche definizione di “grande”



Asintoticamente, $f(n)$ stima $T(n)$ per eccesso a meno di un fattore moltiplicativo:

- per istanze grandi, il tempo di calcolo è al massimo proporzionale ai valori della funzione $f(n)$

Notazione Ω

Data una **funzione approssimante** $f(n)$

$$T(n) \in \Omega(f(n))$$

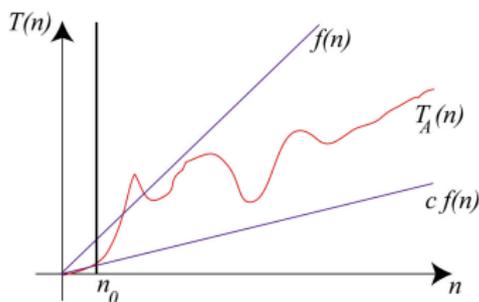
significa formalmente che

$$\exists c > 0, n_0 \in \mathbb{N} : T(n) \geq c f(n) \text{ for all } n \geq n_0$$

dove c e n_0 sono indipendenti da n

$T(n)$ “domina” $cf(n)$

- per qualche valore “piccolo” di c
- per ogni valore “grande” di n
- per qualche definizione di “piccolo” e “grande”



Asintoticamente, $f(n)$ stima $T(n)$ per difetto a meno di un fattore moltiplicativo:

- per istanze grandi, il tempo di calcolo è almeno proporzionale ai valori della funzione $f(n)$

Perché ignorare le costanti moltiplicative?

Il tempo di calcolo effettivo è il prodotto del numero di operazioni elementari T_A per il tempo γ richiesto da ciascuna

$$T_{\text{eff}} = T_A \gamma$$

Il tempo γ richiesto per un'operazione elementare

- dipende dalla tecnologia
- non è rigorosamente uguale per tutte le operazioni

Se cambia la specifica macchina usata, ma non la sua struttura

- il tempo γ richiesto per ciascuna operazione può cambiare
- il numero T_A di operazioni elementari rimane uguale

Un'analisi che ignora i fattori moltiplicativi è valida per tutte le macchine che aderiscono allo stesso modello computazionale

Impatto pratico della complessità

Supponiamo di avere

- un giorno di calcolo a disposizione (86 400 secondi)
- due macchine, il cui tempo medio per operazione elementare è pari, rispettivamente, a $\gamma_1 = 1\mu\text{s}/\text{oper.}$ e $\gamma_2 = 1\text{ ns}/\text{oper.}$
- algoritmi di complessità diversa per lo stesso problema

La massima dimensione trattabile nel tempo disponibile è

$$\bar{n} = \max \{ n \in \mathbb{N} : T_A(n) \gamma \leq 8.64 \cdot 10^{10} \mu\text{s} = 1 \text{ giorno} \}$$

$T_A(n)$	n	$n \log_2 n$	n^2	n^3	2^n	3^n
$\gamma = 1 \mu\text{s}/\text{op}$	$8.64 \cdot 10^{10}$	$2.75 \cdot 10^9$	$2.94 \cdot 10^5$	$4.42 \cdot 10^3$	36.33	22.92
$\gamma = 1 \text{ ns}/\text{op}$	$8.64 \cdot 10^{13}$	$8.29 \cdot 10^{11}$	$9.30 \cdot 10^6$	$4.42 \cdot 10^4$	46.30	29.21

Se ne deduce facilmente che

- un algoritmo migliore surclassa una macchina migliore
- una macchina più veloce è utile solo se si usa un algoritmo veloce

Esercizio 1

Dimostrare che $T(n) = 3n^2 + 7n + 8 \in \Theta(n^2)$, cioè che

$$\exists c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N}^+ : c_1 n^2 \leq 3n^2 + 7n + 8 \leq c_2 n^2 \quad \forall n \geq n_0$$

Il procedimento è semplice

- 1 si fa un'ipotesi sul valore di c_1 e c_2 ,
basata su regole generali, intuizione o semplici tentativi
- 2 si ricava n_0 in modo da rispettare la tesi

Poniamo $c_1 = 3$ e $c_2 = 4$:

- la prima disuguaglianza diventa

$$0 \leq 7n + 8 \quad \text{che vale per ogni } n \in \mathbb{N}^+ \text{ (dunque } n \geq 1)$$

- la seconda disuguaglianza diventa

$$7n + 8 \leq n^2 \quad \text{che vale per ogni } n \geq 8$$

Di conseguenza, $c_1 = 3$, $c_2 = 4$ e $n_0 = 8$ soddisfano la definizione

Esercizio 2

Dimostrare che $T(n) = 3n^2 - 7n + 2 \notin \Theta(n)$, cioè che

$$\nexists c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N}^+ : c_1 n \leq 3n^2 - 7n + 2 \leq c_2 n \quad \forall n \geq n_0$$

Questo equivale a

$$\exists n \geq n_0 : c_1 n > 3n^2 - 7n + 2 \text{ oppure } 3n^2 - 7n + 2 > c_2 n \quad \forall c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N}^+$$

Si deve trovare una n funzione di c_1 , c_2 e n_0 che soddisfi la tesi

Basta soddisfare una delle due disuguaglianze: scegliamo la seconda

$$3n^2 - (7 + c_2)n + 2 > 0$$

Se $(7 + c_2)^2 - 24 = \phi(c_2) < 0$, la disuguaglianza vale per ogni $n \in \mathbb{N}$.

Altrimenti, vale per $n < \frac{7+c_2-\sqrt{\phi(c_2)}}{6}$ oppure per $n > \frac{7+c_2+\sqrt{\phi(c_2)}}{6}$

Scegliamo la seconda disuguaglianza, e la combiniamo con $n \geq n_0$:

$$n = \max \left(n_0, \left\lceil \frac{7 + c_2 + \sqrt{\phi(c_2)}}{6} \right\rceil + 1 \right)$$

Si dimostri che:

- $n^2 \in \Theta(n^2 + 4n + 3)$
- $2n^2 + 3n \in \Theta(n^2)$
- $6n^2 + 2n \in \Theta(n^2)$
- $n^2 \in O(n^2/4 - 2)$
- $2n^2 + 3n \in O(n^3)$
- $n^4 \in O(2^n)$
- $n \log_2 n \in O(n^2)$
- $n^2 \in \Omega(n^2 + 2n + 5)$
- $3n^2 - 2n \in \Omega(n^2)$
- $3n^5 \in \Omega(n^4)$
- $4n^2 \notin \Theta(n^3)$
- $n^2 \notin O(10^6 n)$
- $3^n \notin O(2^n)$
- $2n^2 - 3n \notin \Omega(n \log_2 n)$

Proprietà fondamentali (1)

Riflessività

- $f(n) \in \Theta(f(n))$
- $f(n) \in O(f(n))$
- $f(n) \in \Omega(f(n))$

Simmetria (solo per Θ)

- $g(n) \in \Theta(f(n)) \Leftrightarrow f(n) \in \Theta(g(n))$

Simmetria trasposta (fra O e Ω)

- $g(n) \in O(f(n)) \Leftrightarrow f(n) \in \Omega(g(n))$

Antisimmetria

- $\begin{cases} f(n) \in O(g(n)) \\ f(n) \in \Omega(g(n)) \end{cases} \Leftrightarrow f(n) \in \Theta(g(n))$

Sono facili da ricordare, associando mentalmente Θ a $=$, O a \leq e Ω a \geq

Transitività

- $\begin{cases} f(n) \in \Theta(g(n)) \\ g(n) \in \Theta(h(n)) \end{cases} \Rightarrow f(n) \in \Theta(h(n))$
- $\begin{cases} f(n) \in O(g(n)) \\ g(n) \in O(h(n)) \end{cases} \Rightarrow f(n) \in O(h(n))$
- $\begin{cases} f(n) \in \Omega(g(n)) \\ g(n) \in \Omega(h(n)) \end{cases} \Rightarrow f(n) \in \Omega(h(n))$

Sono facili da ricordare, associando mentalmente Θ a $=$, O a \leq e Ω a \geq

Non vale la completezza: esistono funzioni non confrontabili

(sono rare e "strane": per esempio, n e $n^{1+\sin(\frac{\pi}{2}n)}$)

Complessità asintotica e limiti

Un'altra terna di relazioni fra funzioni si basa sul concetto di limite

- $T(n) \in o(f(n))$ quando $\lim_{n \rightarrow +\infty} \frac{T(n)}{f(n)} = 0$
- $T(n) \sim f(n)$ quando $\lim_{n \rightarrow +\infty} \frac{T(n)}{f(n)} = 1$
- $T(n) \in \omega(f(n))$ quando $\lim_{n \rightarrow +\infty} \frac{T(n)}{f(n)} = +\infty$

Queste relazioni sono legate alle precedenti, ma sono più restrittive

- $T(n) \in o(f(n)) \Rightarrow T(n) \in O(f(n))$
- $T(n) \sim f(n) \Rightarrow T(n) \in \Theta(f(n))$
- $T(n) \in \omega(f(n)) \Rightarrow T(n) \in \Omega(f(n))$

Per dimostrarlo, basta applicare la definizione di limite

$$T(n) \in O(f(n)) \Leftrightarrow \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N} : T(n) \leq c f(n) \text{ for all } n \geq n_0$$

$$T(n) \in o(f(n)) \Leftrightarrow \forall \epsilon \in \mathbb{R}^+, \exists n_\epsilon \in \mathbb{N} : T(n) \leq \epsilon f(n) \text{ for all } n \geq n_\epsilon$$

Sono facili da ricordare, associando mentalmente o a $<$ e ω a $>$

I fattori moltiplicativi costanti non sono significativi

- $f(n) \in O(h(n)) \Leftrightarrow c f(n) \in O(h(n))$ per ogni $c \in \mathbb{R}^+$
- $f(n) \in \Theta(h(n)) \Leftrightarrow c f(n) \in \Theta(h(n))$ per ogni $c \in \mathbb{R}^+$
- $f(n) \in \Omega(h(n)) \Leftrightarrow c f(n) \in \Omega(h(n))$ per ogni $c \in \mathbb{R}^+$

Aggiungere o sottrarre termini dominati non ha effetti significativi

- $f(n) \in O(h(n)) \Leftrightarrow (f(n) + c g(n)) \in O(h(n))$
- $f(n) \in \Theta(h(n)) \Leftrightarrow (f(n) + c g(n)) \in \Theta(h(n))$
- $f(n) \in \Omega(h(n)) \Leftrightarrow (f(n) + c g(n)) \in \Omega(h(n))$

per ogni $c \in \mathbb{R}$ e per ogni $g(n) \in o(f(n))$

Queste proprietà

- non valgono per le corrispondenti relazioni fra numeri
- consentono di usare approssimanti semplici per classificare le funzioni

Dai principi di sostituzione deriva che

- i fattori moltiplicativi e la base dei logaritmi si possono ignorare

$$\log_b f(n) = \frac{\log_a f(n)}{\log_a b}$$

- di un polinomio si può considerare solo il termine direttore

$$(c_r n^r + c_{r-1} n^{r-1} + \dots + c_1 n + c_0) \in \Theta(n^r)$$

Inoltre

- tutte le funzioni limitate appartengono a $\Theta(1)$
- si possono ignorare gli arrotondamenti all'intero per le funzioni che non convergono asintoticamente a 0

Approssimanti di uso comune

Le **approssimanti più usate** per la complessità $T(n)$ di un algoritmo sono

- **polilogaritmiche**: $(\log n)^r$ con $r \geq 0$
- **potenze**: n^r con $r > 0$
- **esponenziali**: r^n con $r > 1$

oppure **prodotti di tali approssimanti**

Le approssimanti fondamentali si dominano in ordine lessicografico stretto

$$(\log n)^r \in o(n^s) \quad \text{per ogni } r \geq 0 \text{ e } s > 0$$

$$n^r \in o(s^n) \quad \text{per ogni } r > 0 \text{ e } s > 1$$

Ovviamente le funzioni con esponenti o basi più grandi dominano le altre

Gli algoritmi con queste approssimanti sono debolmente ordinati

(quindi, non tutti gli algoritmi, ma quasi)

La **complessità asintotica di un algoritmo nel caso pessimo** fornisce una misura del tempo di calcolo dell'algoritmo attraverso i seguenti passaggi

- 1 misuriamo il tempo col **numero T di operazioni elementari eseguite** (così la misura è indipendente dallo specifico meccanismo usato)
- 2 scegliamo un valore n che misuri la **dimensione di un'istanza** (per es., il numero di elementi dell'insieme, di righe o colonne della matrice, di nodi o archi del grafo)
- 3 troviamo il **tempo di calcolo massimo o medio** per ogni dimensione n

$$T_A(n) = \frac{\sum_{I \in \mathcal{I}_P^{(n)}} T_A(I)}{|\mathcal{I}_P^{(n)}|} \text{ o } T_A(n) = \max_{I \in \mathcal{I}_P^{(n)}} T_A(I) \quad n \in \mathbb{N}$$

(questo riduce la complessità a una funzione $T : \mathbb{N} \rightarrow \mathbb{N}$)

- 4 **approssimiamo $T(n)$ con una funzione $f(n)$ più semplice**, di cui interessa solo l'andamento per $n \rightarrow +\infty$ (è più importante che l'algoritmo sia efficiente su dimensioni grandi)
- 5 **raccogliamo le funzioni in classi con la stessa approssimante semplice** (la relazione di approssimazione è una relazione di equivalenza)

Algoritmi iterativi e sommatorie

Un **algoritmo iterativo** ripete le stesse operazioni più volte su dati diversi

```
For  $i := 1$  to  $n$  do  
  Operazioni( $i, I$ )
```

Sia $f(i)$ la **complessità** di Operazioni(i, I)

(in generale dipende anche da $|I|$, ma qui semplifichiamo)

La complessità dell'intero ciclo è

$$F(n) = \sum_{i=1}^n f(i)$$

Come approssimarla asintoticamente rispetto al numero di iterazioni n ?

Teorema: si possono sostituire gli addendi con un'approssimante

$$f(i) \in \Theta(g(i)) \Rightarrow \sum_{i=1}^n f(i) \in \Theta\left(\sum_{i=1}^n g(i)\right)$$

Dimostrazione: *si veda a pag. 20 delle dispense*

Quindi studiamo le sommatorie di approssimanti fondamentali

Stima mediante minorazione e maggiorazione

Vogliamo trovare un'espressione asintotica per la complessità

$$F(n) = \sum_{i=1}^n f(i)$$

Quasi sempre è possibile ipotizzare che $f(i)$ sia

- 1 non negativa: $f(i) \geq 0$ per ogni $i \in \mathbb{N}$
- 2 non decrescente: $f(i+1) \geq f(i)$ per ogni $i \in \mathbb{N}$

Sotto queste ipotesi valgono le banali stime per difetto e per eccesso:

$$f(n) \leq F(n) \leq n f(n)$$

e quindi

$$F(n) \in \Omega(f(n)) \quad F(n) \in O(n f(n))$$

Tipicamente

- le funzioni esponenziali cadono in $\Theta(f(n))$
- le potenze e le funzioni polilogaritmiche cadono in $\Theta(n f(n))$

Esistono tecniche per ottenere stime più precise

Approssimanti esponenziali: somma geometrica

Per le somme di esponenziali esiste una nota soluzione in forma chiusa

$$\sum_{i=0}^n r^i = \frac{r^{n+1} - 1}{r - 1} \Rightarrow F(n) \in \Theta(f(n))$$

Dimostrazione:

$$r F(n) = r + r^2 + \dots + r^n + r^{n+1}$$

$$F(n) = 1 + r + \dots + r^{n-1} + r^n$$

$$(r - 1) F(n) = r^{n+1} - 1$$

16		32	64
4	8		
1	2		

$$\sum_{i=0}^6 2^i = 2^7 - 1$$

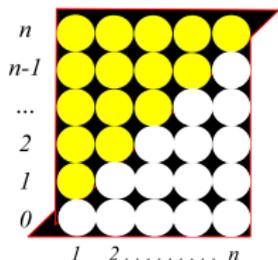
Approssimante lineare: somma aritmetica

Per le somme di funzioni lineari, esiste una nota soluzione in forma chiusa

$$F(n) = \sum_{i=1}^n i = \frac{n(n+1)}{2} \Rightarrow F(n) \in \Theta(nf(n))$$

Dimostrazione:

$$\begin{aligned} 2F(n) &= 1 + 2 + \dots + (n-1) + n + \\ &\quad n + (n-1) + \dots + 2 + 1 = \\ &(n+1) + (n+1) + \dots + (n+1) + (n+1) = n(n+1) \end{aligned}$$

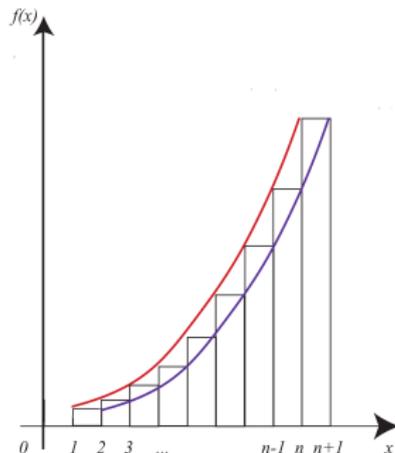


$$\sum_{i=1}^5 i = \frac{5(5+1)}{2}$$

Per le potenze, c'è un'espressione esatta, ma anche una stima semplice

Stima mediante integrali

Sia $f(x)$ una funzione **non decrescente**



La somma è pari all'area sottesa dal grafico a scaletta

$$F(n) = \sum_{i=1}^n f(i)$$

ed è compresa fra le aree sottese dai grafici di $f(x)$ e $f(x-1)$

$$f(1) + \int_2^{n+1} f(x-1) dx \leq F(n) \leq \int_1^n f(x) dx + f(n)$$

$$f(1) + \int_1^n f(x) dx \leq F(n) \leq \int_1^n f(x) dx + f(n)$$

Per le funzioni non crescenti, basta scambiare $f(1)$ e $f(n)$

$$f(i) = i^r \quad \text{e} \quad F(n) = \sum_{i=1}^n i^r$$

La stima con l'integrale è

$$1^r + \int_1^n x^r dx \leq F(n) \leq \int_1^n x^r dx + n^r$$

da cui

$$1 + \frac{n^{r+1} - 1}{r+1} \leq F(n) \leq \frac{n^{r+1} - 1}{r+1} + n^r$$

e quindi $F(n) \in \Theta(n^{r+1})$

Per le somme di potenze, $F(n) \in \Theta(nf(n))$ come nel caso lineare

Stima mediante decomposizione

Stime per difetto e per eccesso si possono eseguire anche

- decomponendo la somma
- approssimando separatamente le singole parti

Applichiamo questa tecnica alle funzioni polilogaritmiche:

$$\begin{aligned} F(n) &= \sum_{i=1}^n (\log i)^r = \sum_{i=1}^{n/2} (\log i)^r + \sum_{i=n/2+1}^n (\log i)^r > \\ &> \frac{n}{2} (\log 1)^r + \frac{n}{2} \left(\log \frac{n}{2}\right)^r \end{aligned}$$

Siccome per n abbastanza grande vale $\log \frac{n}{2} > \frac{\log n}{2}$:

$$F(n) > 0 + \frac{n (\log n)^r}{2^r}$$

Quindi $F(n) \in \Omega(n (\log n)^r)$ e siccome in generale $F(n) \in O(n (\log n)^r)$

$$F(n) \in \Theta(n (\log n)^r), \text{ cioè ancora } F(n) \in \Theta(nf(n))$$

In generale

- se $f(i)$ contiene esponenziali, si maggiorano gli altri singoli termini

$$F(n) = \sum_{i=1}^n \left(r^i i^a (\log i)^b \right) \leq \sum_{i=1}^n \left(r^i n^a (\log n)^b \right) = n^a (\log n)^b \sum_{i=1}^n r^i$$

e si conclude che $F(n) \in \Theta(f(n))$

- se $f(i)$ non contiene esponenziali, si decompone la somma

$$F(n) = \sum_{i=1}^n \left(i^a (\log i)^b \right) \geq \sum_{i=n/2+1}^n \left(i^a (\log i)^b \right) \geq \frac{n}{2} \left(\left(\frac{n}{2} \right)^a \left(\log \frac{n}{2} \right)^b \right)$$

e si conclude che $F(n) \in \Theta(nf(n))$