

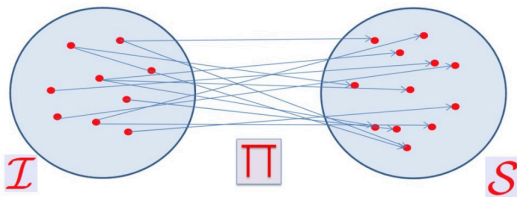
Dati e Algoritmi: A. Pietracaprina

Nozioni di Base

Argomenti trattati

- Nozioni di: problema computazionale, modello di calcolo, algoritmo, pseudocodice, struttura dati
- Analisi di complessità: analisi al caso pessimo, analisi asintotica, ordini di grandezza
- Analisi di correttezza: tecniche di dimostrazione, induzione, invarianti
- Algoritmi ricorsivi e loro analisi

Problema Computazionale



Un *problema computazionale* è costituito da

- un insieme \mathcal{I} di ISTANZE (i *possibili input*)
- un insieme \mathcal{S} di SOLUZIONI (i *possibili output*)
- una *relazione* Π che a ogni istanza $i \in \mathcal{I}$ associa una o più soluzioni $s \in \mathcal{S}$.

Oss. Π è un sottoinsieme del prodotto cartesiano $\mathcal{I} \times \mathcal{S}$

Esempi

Somma di Interi (\mathbb{Z})

- $\mathcal{I} = \{(x, y) : x, y \in \mathbb{Z}\};$
- $\mathcal{S} = \mathbb{Z};$
- $\Pi = \{((x, y), s) : (x, y) \in \mathcal{I}, s \in \mathcal{S}, s = x + y\}.$

Ad es: $((1, 9), 10) \in \Pi; \quad ((23, 6), 29) \in \Pi \quad ((13, 45), 31) \notin \Pi$

Ordinamento di array di interi

- $\mathcal{I} = \{A : A = \text{array di interi}\};$
- $\mathcal{S} = \{B : B = \text{array ordinati di interi}\};$
- $\Pi = \{(A, B) : A \in \mathcal{I}, B \in \mathcal{S}, B \text{ contiene gli stessi interi di } A\}.$

Ad es.

$$\begin{aligned}(< 43, 16, 75, 2 >, < 2, 16, 43, 75 >) &\in \Pi \\(< 7, 1, 7, 3, 3, 5 >, < 1, 3, 3, 5, 7, 7 >) & \\(< 13, 4, 25, 17 >, < 11, 27, 33, 68 >) &\end{aligned}$$

Ordinamento di array di interi (ver.2)

- $\mathcal{I} = \{A : A = \text{array di interi}\};$
- $\mathcal{S} = \{P : P = \text{permutazioni}\};$
- $\Pi = \{(A, P) : A \in \mathcal{I}, P \in \mathcal{S}, P \text{ ordina gli interi di } A\}.$

Ad es.

$$\begin{aligned}(< 43, 16, 75, 2 >, < 4, 2, 1, 3 >) &\in \Pi \\(< 7, 1, 7, 3, 3, 5 >, < 2, 4, 5, 6, 1, 3 >) &\in \Pi \\(< 7, 1, 7, 3, 3, 5 >, < 2, 5, 4, 6, 1, 3 >) &\in \Pi \\(< 13, 4, 25, 17 >, < 1, 2, 4, 3 >) &\end{aligned}$$

Osservazioni

- istanze diverse possono avere la stessa soluzione (ad es., somma)
- un'istanza può avere diverse soluzioni (ad es., ordinamento ver. 2)

Esercizio

Specificare come problema computazionale \square la verifica se due insiemi finiti di oggetti da un universo U sono disgiunti oppure no.

$$I \equiv \{(A, B) : A, B \subseteq U, A, B \text{ finiti}\}$$

$$S \equiv \{true, false\}$$

$$\Pi \equiv \{((A, B), s) \mid \text{se } A \cap B = \emptyset \text{ allora } s = true, \text{ se } A \cap B \neq \emptyset \text{ allora } s = false\} \quad s \in S, (A, B) \in I$$

Esercizio

Specificare come problema computazionale \square la ricerca dell'inizio e della lunghezza del più lungo segmento di 1 consecutivi in una stringa binaria.

$$I \equiv \{A : A \text{ è una stringa binaria}\}$$

$$S \equiv \{(i, l) : i, l \in \mathbb{N}_0\}$$

$$\Pi \equiv \{(A, (i, l)) : i \text{ la casella di inizio del segmento di 1 consecutivi più numeroso, } l \text{ la lunghezza del segmento di 1 consecutivi più lungo}\}$$

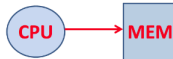
Algoritmo e modello di calcolo

Definizione

Algoritmo: procedura computazionale ben definita che trasforma un dato *input* in un *output* eseguendo una sequenza finita di *operazioni elementari*.

L'algoritmo fa riferimento a un *modello di calcolo*, ovvero un'astrazione di computer che definisce l'insieme di operazioni elementari.

Modello di calcolo RAM ¹ (*Random Access Machine*)



- *input, output, dati intermedi (e programma): in memoria*
- *operazioni elementari: assegnamento, operazioni logiche, operazioni aritmetiche, indicizzazione di array, return di un valore da parte di un metodo, ecc.*

¹Diverso da *Random Access Memory*!

Un algoritmo A *risolve un problema computazionale* $\Pi \subseteq \mathcal{I} \times \mathcal{S}$ se:

- ① A calcola una funzione da \mathcal{I} a \mathcal{S} , e quindi,
 - riceve come input istanze $i \in \mathcal{I}$
 - produce come output soluzioni $s \in \mathcal{S}$
- ② Dato $i \in \mathcal{I}$, A produce in output $s \in \mathcal{S}$ tale che $(i, s) \in \Pi$.

Se Π associa più soluzioni a una istanza i , per tale istanza A ne calcola una (quale, dipende da come è stato progettato).

Pseudocodice

Per semplicità e facilità di analisi, descriviamo un algoritmo utilizzando uno **pseudocodice** strutturato come segue:

Algoritmo *nome* (^{input}*parametri*) → è una firma (signature) dell'algoritmo

Input: *breve descrizione dell'istanza di input*

Output: *breve descrizione della soluzione restituita in output*

descrizione chiara dell'algoritmo tramite costrutti di linguaggi di programmazione e, se utile ai fini della chiarezza, anche tramite linguaggio naturale, dalla quale sia facilmente determinabile la sequenza di operazioni elementari eseguita per ogni dato input.

USARE SEMPRE QUESTA STRUTTURA PER LO PSEUDOCODICE!!

Per maggiori dettagli fare riferimento alla dispensa sulla **Specificazione di Algoritmi in Pseudocodice**, disponibile sul Moodle del corso.

per cose semplici che sarebbero lunghe da scrivere con i costrutti del linguaggio di programmazione

Esempio: trasposta di una matrice $n \times n$ di interi

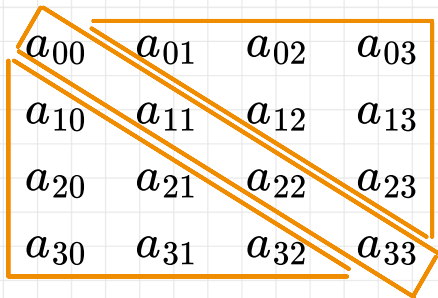
Problema computazionale: $\Pi \subseteq I \times S$

$I \equiv \{A : \text{matrice } n \times n \text{ di interi}\}$

$S \equiv \{B \text{ matrice } n \times n \text{ di interi}\}$

$\Pi \equiv \{(A, B) : A \in I, B \in S, B = A^t\}$

Algoritmo: Idea per $n=4$



A 4x4 matrix of elements a_{ij} is shown. A double orange line runs from the top-left corner (a_{00}) to the bottom-right corner (a_{33}), defining the upper triangular region. The elements are arranged as follows:

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

Scambio ciascuna entry a_{ij} del triangolo superiore (a_{ij} : $i=0 \dots n-2$, $j>i$) con a_{ji} (l'omologa del triangolo inferiore)

Algoritmo transpose (A)

input matrice A nxn di interi a_{ij} , $0 \leq i, \leq n$

output matrice A^t

```
for i ← 0 to n-2 do{  
    for j ← i+1 to n-1 do{  
        scambia  $a_{ij}$  con  $a_{ji}$   
    }  
}  
return A
```

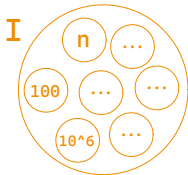
Taglia di una istanza

L'analisi di un algoritmo (ad es., la determinazione della sua complessità) viene solitamente fatta **partizionando le istanze in gruppi in base alla loro taglia (size)**, in modo che le istanze di un gruppo siano tra loro *confrontabili*.

La taglia di un'istanza è espressa da uno o più valori che ne caratterizzano la grandezza.

es: se ho n elementi da ordinare, la taglia è n
Nel caso di ordinamento fa riferimento alla dimensione fisica dell'istanza

La taglia serve per dividere in gruppi omogenei da analizzare separatamente



- **Ordinamento di un array:**

Taglia n = # di elementi dell'array

- **Trasposta di una matrice:**

matrice $n \times m$

- Taglia $x = n \times m$
- Taglia (n, m) = # righe e # colonne
- (se $n=m$) Taglia n = # di righe/colonne

Strutture dati

Le strutture dati sono usate dagli algoritmi per organizzare e accedere in modo sistematico ai dati di input e ai dati generati durante l'esecuzione.

Definizione

Una *struttura dati* è una collezione di *oggetti* corredata di *metodi* di accesso e/o modifica.

Due *livelli di astrazione*:

- 1 *Livello logico*: specifica l'organizzazione logica degli oggetti della collezione, e la relazione input-output di ciascun metodo (a questo livello si parla di *Abstract Data Type* o ADT)
- 2 *Livello fisico*: specifica il layout fisico dei dati e la realizzazione dei metodi tramite algoritmi.

Ad esempio in Java:

- *Livello logico*: (gerarchia di) *interfacce*
- *Livello fisico*: (gerarchia di) *classi*.

Entro 8 ottobre

Esercizio

Siano A e B due array di n ed m interi, rispettivamente. Scrivere un algoritmo in pseudocodice per calcolare il seguente valore:

$$v = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (A[i] \cdot B[j]).$$

Esercizio

Sia A un array di n interi distinti, ordinato in senso crescente, e x un valore intero. Scrivere due algoritmi in pseudocodice che implementano la ricerca binaria per trovare x in A . Entrambi gli algoritmi restituiscono l'indice i tale che $A[i] = x$, se x appare in A , e -1 altrimenti. Il primo algoritmo deve essere iterativo e usare un solo ciclo, e il secondo algoritmo deve essere ricorsivo.

Analisi degli algoritmi

L'analisi di un algoritmo A mira a studiarne l'*efficienza* e l'*efficacia*. In particolare, essa può valutare i seguenti aspetti:

Complessità

- tempo
- spazio

Correttezza

- terminazione (se termina o "va in loop")
- soluzione del problema computazionale

Noi ci concentreremo su:

- ① complessità: tempo
- ② correttezza: soluzione del problema computazionale

COMPLESSITÀ IN TEMPO

OBIETTIVO:

Stimare il tempo di esecuzione (*running time*) di un algoritmo, al fine di:

- valutarne l'efficienza
- poterlo confrontare con altri algoritmi per lo stesso problema.

REQUISITI PER LA COMPLESSITÀ IN TEMPO

La **complessità in tempo** deve:

- ① riguardare tutti gli input
- ② permettere di confrontare algoritmi (senza necessariamente determinare il tempo di esecuzione esatto)
- ③ essere derivabile dallo pseudocodice

Stimare sperimentalmente il tempo di esecuzione di un algoritmo (ad es., in Java con `System.currentTimeMillis()`) è utile, ma **non soddisfa requisiti**. Perché?

- Richiede di implementare l'algoritmo con un programma e l'impatto dell'implementazione può influenzare il confronto tra algoritmi
- Non si possono considerare tutti gli input (se infiniti)
- La stima dipende dall'ambiente HW/SW

APPROCCIO CHE SODDISFA I REQUISITI

- **Analisi al caso pessimo** (*worst-case*) in funzione della taglia dell'istanza (req. 1,2).
- **Conteggio operazioni elementari nel modello RAM** (req. 2,3)
- **Analisi asintotica** (per semplificare il conteggio)

Osservazione: esiste anche l'analisi al caso medio (*average case*), e l'analisi probabilistica

Sia A un algoritmo che risolve $\Pi \subseteq \mathcal{I} \times \mathcal{S}$.

Definizione

La **complessità (in tempo) al caso peggio** di A è una funzione $t_A(n)$ definita come *il massimo numero di operazioni elementari che A esegue per risolvere una istanza di taglia n .*

La definizione rispetta i tre requisiti definiti sopra

In altre parole, se chiamiamo $t_{A,i}$ il numero di operazioni eseguite da A per l'istanza i , abbiamo che

$$t_A(n) = \max\{t_{A,i} : \text{istanza } i \in \mathcal{I} \text{ di taglia } n\}.$$

Difficoltà nel calcolo di $t_A(n)$

Determinare $t_A(n)$ per ogni n è arduo, se non impossibile, perché:

- È difficile identificare l'istanza peggiore di taglia n .
- È difficile contare il max numero di operazioni richieste per risolvere tale istanza peggiore (il conteggio richiederebbe anche una specifica dettagliata del set di operazioni elementari del modello RAM).

È necessario determinare esattamente $t_A(n)$?

No, per le seguenti ragioni:

- Il tempo di esecuzione, che la complessità vuole stimare, dipende da tanti fattori che è impossibile quantificare in modo preciso.
- Le diverse operazioni elementari del modello RAM possono avere impatto diverso sui tempi di esecuzione a seconda delle architetture.

⇒ ci accontentiamo di **limiti superiori** e **limiti inferiori** a $t_A(n)$.

Esempio: arrayMax

Algoritmo arrayMax(A)

Input: array $A[0 \div n - 1]$ di $n \geq 1$ interi

Output: max intero in A

currMax $\leftarrow A[0]$;

for $i \leftarrow 1$ **to** $n - 1$ **do**

if (currMax $< A[i]$) **then** currMax $\leftarrow A[i]$;

return currMax

Taglia dell'istanza: n (ragionevole)

Stima di $t_{\text{arrayMax}}(n)$

È facile vedere che per una qualsiasi istanza di taglia n :

- al di fuori del ciclo **for** arrayMax esegue un numero costante (rispetto a n) di operazioni.
- in ciascuna delle $n - 1$ iterazioni del ciclo **for** si esegue un numero costante di operazioni.

⇒ esistono costanti $c_1, c_2, c_3, c_4 > 0$ tali che per ogni n

$$t_{\text{arrayMax}}(n) \leq c_1 n + c_2 \quad (\text{limite superiore})$$

$$t_{\text{arrayMax}}(n) \geq c_3 n + c_4 \quad (\text{limite inferiore})$$

Dobbiamo stimare le costanti c_1, c_2, c_3, c_4 ?

No, per le stesse ragioni per cui non è necessario stimare esattamente la complessità

Analisi Asintotica

Si ricorre quindi alla **analisi asintotica**, ignorando

- **fattori moltiplicativi costanti** (rispetto alla taglia dell'istanza)
- **termini additivi non dominanti**

Nel caso di `arrayMax` possiamo affermare che

- $t_{\text{arrayMax}}(n)$ è *al più* proporzionale a n (limite superiore)
- $t_{\text{arrayMax}}(n)$ è *almeno* proporzionale a n (limite inferiore)

Dalle due affermazioni consegue che:

$t_{\text{arrayMax}}(n)$ è **proporzionale a n** (limite stretto)

Per esprimere efficacemente affermazioni come quelle fatte sopra si usano gli **ordini di grandezza**: $O(\cdot)$, $\Omega(\cdot)$, $\Theta(\cdot)$, $o(\cdot)$.

Ordini di grandezza

$f(n), g(n)$ funzioni da \mathbb{N} a $\mathbb{R}^+ \cup \{0\}$.

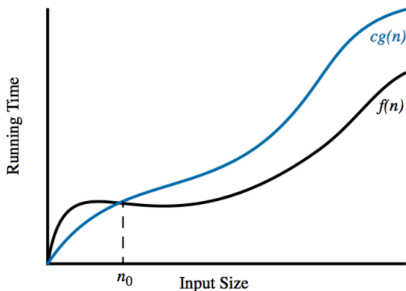
Definizione

$f(n) \in O(g(n))$ se $\exists c > 0$ e $\exists n_0 \geq 1$, costanti rispetto a n , tali che

$$f(n) \leq cg(n), \forall n \geq n_0$$

In parole: $f(n)$ è al più proporzionale a $g(n)$

O anche: $f(n)$ non cresce asintoticamente più di $c \cdot g(n)$



Esempi

$f(n)$	$O(\cdot)$	c	n_0
$3n + 4$ per $n \geq 1$	$O(n)$	4	4
$n + 2n^2$ per $n \geq 1$	$O(n^2)$	3	1
2^{100} per $n \geq 1$	$O(1)$	2^{100}	1
$c_1n + c_2$ per $n \geq 1$, $c_1, c_2 > 0$	$O(n)$	$c_1 + c_2$	1

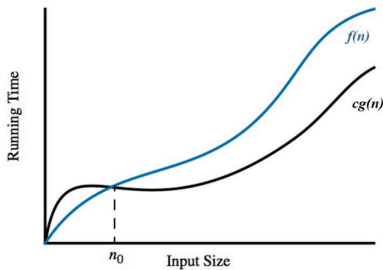
$$3n + 4 \in O(n^5) \quad c = 7 \quad n_0 = 1$$

Definizione

$f(n) \in \Omega(g(n))$ se $\exists c > 0$ e $\exists n_0 \geq 1$, costanti rispetto a n , tali che

$$f(n) \geq cg(n), \forall n \geq n_0$$

In parole: $f(n)$ è almeno proporzionale a $g(n)$



Esempi

$f(n)$	$\Omega(\cdot)$	c	n_0
$3n + 4$ per $n \geq 1$	$\Omega(n)$	1	1
$n + 2n^2$ per $n \geq 1$	$\Omega(n^2)$	1	1
2^{100} per $n \geq 1$	$\Omega(1)$	2^{100}	1
$c_3n + c_4$ per $n \geq 1$, $c_3, c_4 > 0$	$\Omega(n)$	1	1

Definizione

$f(n) \in \Theta(g(n))$ se

$$f(n) \in O(g(n)) \quad \text{e} \quad f(n) \in \Omega(g(n)).$$

In parole: $f(n)$ è (**esattamente**) proporzionale a $g(n)$

Applicando la definizione agli esempi dei lucidi precedenti otteniamo:

- $f(n) = 3n + 4 \in \Theta(n)$
- $f(n) = n + 2n^2 \in \Theta(n^2)$
- $f(n) = 2^{100} \in \Theta(1)$
- $t_{\text{arrayMax}}(n) \in \Theta(n)$

Definizione

$f(n) \in o(g(n))$ se

$$\lim_{n \rightarrow +\infty} f(n)/g(n) = 0.$$

In parole: $f(n)$ è *asintoticamente* più piccola (cresce meno) di $g(n)$

Esempi

- $f(n) = 100n$ per $n \geq 1 \Rightarrow f(n) \in o(n^2)$
- $f(n) = 3n/(\log_2 n)$ per $n \geq 1 \Rightarrow f(n) \in o(n)$

Useremo o-piccolo per dire che un algoritmo è asintoticamente migliore di un altro

Proprietà degli ordini di grandezza

① $\max\{f(n), g(n)\} \in \Theta(f(n) + g(n))$, per ogni $f(n), g(n) : \mathbb{N} \rightarrow \mathbb{R}^+ \cup \{0\}$.

② $\sum_{i=0}^k a_i n^i \in \Theta(n^k)$, se $a_k > 0$, k, a_i costanti, e $k \geq 0$.

Ad es.: $(n+1)^5 \in \Theta(n^5)$ (cioè tengo il termine con l'esponente più grande)

③ $\log_b n \in \Theta(\log_a n)$, se $a, b > 1$ sono costanti.

Oss. La proprietà deriva dalla relazione $\log_b n = (\log_a n) * (\log_b a)$ e, grazie a essa, *la base dei logaritmi, se costante, si omette negli ordini di grandezza*, a meno che il logaritmo non sia all'esponente.

④ $n^k \in o(n^a)$, se $k > 0, a > 1$ sono costanti.

⑤ $(\log_b n)^k \in o(n^h)$ se b, k, h sono costanti, con $b > 1$ e $h, k > 0$.

Strumenti matematici

- **(Parte bassa)** $\forall x \in \mathbb{R}$, si definisce $\lfloor x \rfloor =$ più grande intero $\leq x$.

Ad es.: $\lfloor 3/2 \rfloor = 1$; $\lfloor 3 \rfloor = 3$.

- **(Parte alta)** $\forall x \in \mathbb{R}$, si definisce $\lceil x \rceil =$ più piccolo intero $\geq x$.

Ad es.: $\lceil 3/2 \rceil = 2$; $\lceil 3 \rceil = 3$.

- **(Modulo)** $\forall x, y \in \mathbb{Z}$, con $y \neq 0$, si definisce $x \bmod y =$ resto della divisione intera x/y . (% in Java).

Ad es.: $39 \bmod 7 = 4$; $80 \bmod 4 = 0$.

- (Sommatorie notevoli) $\forall n \in \mathbb{Z}$ e $a \in \mathbb{R}$, con $n \geq 0$ e $a > 0$ vale

$$\sum_{i=0}^n i = \frac{n(n+1)}{2} \in \Theta(n^2)$$

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \in \Theta(a^n) \quad \text{per } a > 1$$

$$\sum_{i=1}^n a^i = \frac{a^{n+1} - 1}{a - 1} - 1 \in \Theta(a^n) \quad \text{per } a > 1$$

Analisi di complessità in pratica

Dato un algoritmo A e detta $t_A(n)$ la sua complessità al caso pessimo si cercano limiti asintotici superiori e/o inferiori a $t_A(n)$.

Limite Superiore (Upper Bound)

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

Si prova argomentando che per ogni n “abbastanza grande” e per ciascuna istanza di taglia n l'algoritmo esegue $\leq cf(n)$ operazioni, con c costante (non serve determinare c)

Limite Inferiore (Lower Bound)

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

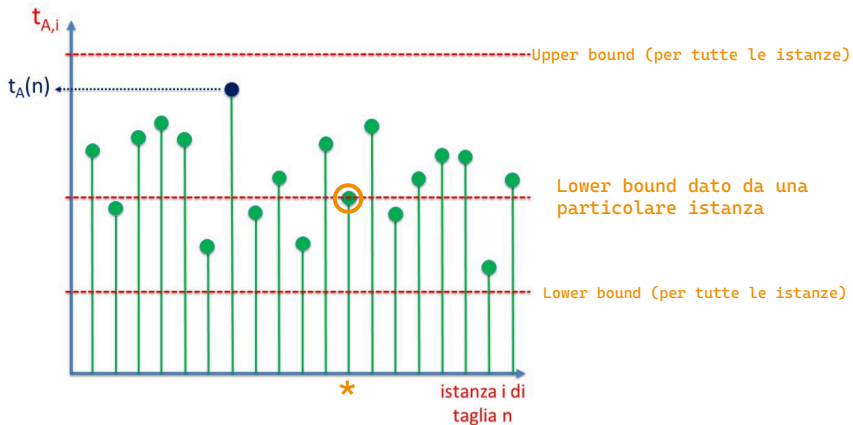
Si prova argomentando che per ogni n “abbastanza grande”, esiste una istanza di taglia n per la quale l’algoritmo esegue $\geq cf(n)$ operazioni, con c costante (non serve determinarla)

In alcuni casi è comodo argomentare che per *ciascuna* istanza di taglia n l’algoritmo esegue $\geq cf(n)$ operazioni

Sia che si provi $t_A(n) \in O(f(n))$ o che si provi $t_A(n) \in \Omega(f(n))$

- $f(n)$ deve essere più vicino possibile alla complessità vera (*tight bound*).
- $f(n)$ deve essere più semplice possibile \Rightarrow no costanti, no termini additivi di ordine inferiore, ma *solo termini essenziali*!

Limiti superiori e inferiori



Terminologia per complessità [GTG14, Par. 4.2]

- logaritmica: $\Theta(\log n)$, base 2 o costante > 1
- lineare: $\Theta(n)$
- quadratica: $\Theta(n^2)$
- cubica: $\Theta(n^3)$
- polinomiale: $\Theta(n^c)$, $c > 0$ costante
- esponenziale: $\Omega(a^n)$, $a > 1$ costante
- polilogaritmica: $\Theta((\log n)^c)$, $c > 0$ costante

Esempio: Prefix Averages ([GTG14] pp. 164-165)

Si consideri il seguente problema computazionale: dato un array di n interi $X[0 \dots n-1]$ calcolare un array $A[0 \dots n-1]$ dove

$$A[i] = \left(\sum_{j=0}^i X[j] \right) \frac{1}{i+1}, \text{ per } 0 \leq i < n.$$

Vedremo adesso 2 algoritmi: 1 banale e inefficiente, e 1 più furbo ed efficiente.

Per entrambi gli algoritmi vale la seguente specifica di input-output:

Input: $X[0 \dots n-1]$ array di n interi

Output: $A[0 \dots n-1]$: $A[i] = (\sum_{j=0}^i X[j]) / (i+1)$ per $0 \leq i < n$

Algoritmo prefixAverages1

for $i \leftarrow 0$ to $n-1$ do

$a \leftarrow 0$;

 for $j \leftarrow 0$ to i do

$a \leftarrow a + X[j]$;

$A[i] \leftarrow \frac{a}{i+1}$;

return A

i+1 iterazioni con $\Theta(1)$
operazioni ciascuna

- Fuori dal for esterno: $\Theta(1)$ operazioni
- Per ciascuna iterazione i del for esterno ($i=0, \dots, n-1$)
 - $\Theta(i+1)$ operazioni nel for interno
 - $\Theta(1)$ altre operazioni

Complessità: $t_{pA1}(n) \in \Theta\left(1 + \sum_{i=0}^{n-1} i + 1\right) = \Theta\left(\sum_{i=0}^{n-1} i\right) = \Theta\left(\frac{(n-1)n}{2}\right) = \Theta(n^2)$

Algoritmo prefixAverages2

```
 $s \leftarrow 0;$   
for  $i \leftarrow 0$  to  $n-1$  do  
     $s \leftarrow s + X[i];$   
     $A[i] \leftarrow \frac{s}{i+1};$   
return  $A$ 
```

n interazioni con $\Theta(1)$
operazioni ciascuna

- Fuori dal for: $\Theta(1)$ operazioni
- Interazioni i del for ($i=0, \dots, n-1$). $\Theta(1)$ operazioni

Complessità: $t_{pA2}(n) \in \Theta\left(1 + \sum_{i=0}^{n-1} 1\right) = \Theta(n)$

Possiamo affermare che $t_{pA2}(n) \in o(t_{pA1}(n))$, cioè è migliore

Somma di Interi (Z)

- $\mathcal{I} = \{(x, y) : x, y \in Z\};$
- $\mathcal{S} = Z;$
- $\Pi = \{((x, y), s) : (x, y) \in \mathcal{I}, s \in \mathcal{S}, s = x + y\}.$

Ad es: $((1, 9), 10) \in \Pi;$ $((23, 6), 29) \in \Pi$ $((13, 45), 31) \notin \Pi$

Ordinamento di array di interi

- $\mathcal{I} = \{A : A = \text{array di interi}\};$
- $\mathcal{S} = \{B : B = \text{array ordinati di interi}\};$
- $\Pi = \{(A, B) : A \in \mathcal{I}, B \in \mathcal{S}, B \text{ contiene gli stessi interi di } A\}.$

Ad es.

$$(< 43, 16, 75, 2 >, < 2, 16, 43, 75 >) \in \Pi$$

$$(< 7, 1, 7, 3, 3, 5 >, < 1, 3, 3, 5, 7, 7 >)$$

$$(< 13, 4, 25, 17 >, < 11, 27, 33, 68 >)$$

Ordinamento di array di interi (ver.2)

- $\mathcal{I} = \{A : A = \text{array di interi}\};$
- $\mathcal{S} = \{P : P = \text{permutazioni}\};$
- $\Pi = \{(A, P) : A \in \mathcal{I}, P \in \mathcal{S}, P \text{ ordina gli interi di } A\}.$

Ad es.

$$(< 43, 16, 75, 2 >, < 4, 2, 1, 3 >) \in \Pi$$

$$(< 7, 1, 7, 3, 3, 5 >, < 2, 4, 5, 6, 1, 3 >) \in \Pi$$

$$(< 7, 1, 7, 3, 3, 5 >, < 2, 5, 4, 6, 1, 3 >) \in \Pi$$

$$(< 13, 4, 25, 17 >, < 1, 2, 4, 3 >)$$

Esercizio

Specificare come problema computazionale Π la verifica se due insiemi finiti di oggetti da un universo U sono disgiunti oppure no.

$$I \equiv \{(A, B) : A, B \subseteq U, A, B \text{ finiti}\}$$
$$S \equiv \{true, false\}$$

Esercizio

Specificare come problema computazionale Π la ricerca dell'inizio e della lunghezza del più lungo segmento di **1** consecutivi in una stringa binaria.

$$I \equiv \{A : A \text{ è una stringa binaria}\}$$

$$S \equiv \{(i, l) : i, l \in \mathbb{N}_0\}$$

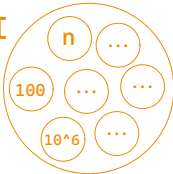
$$\Pi \equiv \{(A, (i, l)) : i \text{ la casella di inizio del segmento di 1 consecutivi più numeroso, } l \text{ la lunghezza del segmento di 1 consecutivi più lungo}\}$$

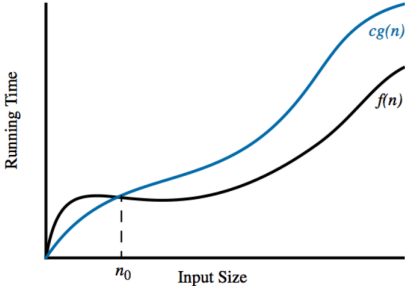


A 4x4 grid of elements a_{ij} is shown, enclosed in a thick orange border. A diagonal line, also in orange, runs from the top-left corner to the bottom-right corner, passing through the elements a_{11} , a_{22} , and a_{33} . The elements are arranged in rows and columns, with the first row containing a_{00} , a_{01} , a_{02} , and a_{03} , and the last row containing a_{30} , a_{31} , a_{32} , and a_{33} .

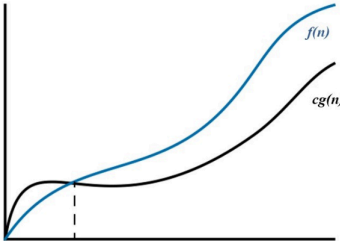
a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

I





Running Time



n_0

Input Size

$f(n)$

$cg(n)$

