



Algoritmi e Strutture Dati

Modelli di calcolo e analisi di algoritmi

Università di Camerino

Corso di Laurea in Informatica
6 CFU

I periodo didattico

Docente: Emanuela Merelli

algoritmi e strutture dati



Il termine Algoritmo

Il termine *Algoritmo* significa procedimento di calcolo, deriva dal termine latino medievale *algorismus*, che a sua volta deriva dal nome del matematico usbeco Abu Ja'far Mohammad ibn-Musa *al-Khwarizmi*, vissuto nel IX (?) secolo d.C.



algoritmi e strutture dati



Algoritmo

Un algoritmo è un **procedimento matematico di calcolo**, è **ben definito** e viene eseguito per giungere alla risoluzione di un problema computazionale; prende come input un insieme di valori e produce come output un insieme di valori

Più precisamente, un procedimento di calcolo esplicito e descrivibile con un numero **finito di regole** che conduce al risultato dopo un **numero finito di operazioni**, cioè di applicazioni delle regole;

algoritmi e strutture dati



Perché parliamo di Algoritmi

La teoria degli algoritmi ha iniziato a stabilizzarsi agli inizi del XX secolo, mentre ...

le **tecniche di progettazione** di algoritmi e di **analisi di correttezza** e di **efficienza** si sono evolute nella seconda metà del XX secolo grazie alla diffusione dei calcolatori elettronici

Ovunque si impieghi un calcolatore elettronico occorre definire algoritmi corretti e efficienti che ne utilizzino al massimo le potenzialità (tempo computazione e spazio di memorizzazione)

Esempi di algoritmi efficienti:

- controllo dei voli aerei
- regolazione reattori nucleari
- reperimento d'informazioni da archivi
- smistamento di comunicazioni telefoniche
- gioco degli scacchi
- controllo della produzione di una catena di montaggio
- ...

algoritmi e strutture dati



Gli Algoritmi e i Programmi

Gli algoritmi vengono descritti tramite **programmi**, che si avvalgono di istruzioni e costrutti dei **linguaggi di programmazione** per essere eseguiti da calcolatori elettronici

Le proprietà degli algoritmi sono talmente **fondamentali**, **generali** e **robuste**, da essere **indipendenti** dalle caratteristiche di specifici linguaggi di programmazione o di particolari calcolatori elettronici

algoritmi e strutture dati



Programmi

I programmi sono formulazioni **concrete** di algoritmi **astratti** che si basano su particolari rappresentazioni dei **dati**, e utilizzano operazioni di manipolazione dei dati, messe a disposizione da uno specifico **linguaggio di programmazione**

algoritmi e strutture dati



I Dati

La nozione di algoritmo è inscindibile da quella di dato.

Per risolvere un problema computazionale, occorre organizzare ed elaborare dati.

Un algoritmo può essere visto come un manipolatore di dati

algoritmi e strutture dati



Le Strutture Dati

Gli algoritmi a fronte di dati d'ingresso che descrivono il problema producono dati d'uscita come risultato del problema

E' fondamentale che i **dati** siano ben organizzati e **strutturati** in modo che il calcolatore li possa elaborare efficientemente

algoritmi e strutture dati



"Clever" e "Efficient"

Obiettivo:

Studiare i modi più **appropriati** di organizzare i dati di un problema al fine di realizzare un algoritmo **efficiente**

Domanda:

- Che cosa intendiamo per **appropriato** "clever"?
- Che cosa intendiamo per **efficiente** "efficient"?

algoritmi e strutture dati



Appropriato e Efficiente

Liste, pile, code	Insert
Heaps	Delete
Alberi binari di ricerca	Find
B-trees	Merge
Tabelle Hash	Shortest path
Grafi	

Data Structures

Algorithms

algoritmi e strutture dati



Quale è l'algoritmo descritto da questa procedura?

...

```

for i ← 2 to length[A]
  do key ← A[i]
    Δ Si inserisce A[i] nella sequenza ordinata A[1..j-1]
    j ← i-1
    while j > 0 e A[j] > key
      do A[j+1] ← A[j]
      j ← j-1
    A[j+1] ← key

```

pseudocodice

algoritmi e strutture dati



Problema dell'ordinamento

DEF:

Data una sequenza di numeri n $\langle a_1, a_2, \dots, a_n \rangle$
 Trovare una permutazione $\langle a'_1, a'_2, \dots, a'_n \rangle$ della
 sequenza data tale che $a'_1 < a'_2 < \dots < a'_n$

Input: $\langle a_1, a_2, \dots, a_n \rangle$

Output: $\langle a'_1, a'_2, \dots, a'_n \rangle$ oppure
 $\langle a_{\pi(1)}, a_{\pi(2)}, \dots, a_{\pi(n)} \rangle$ dove

π è una opportuna permutazione degli indici
 $1, \dots, n$

algoritmi e strutture dati



Istanza del problema

Input: (31,41,59,26,41,58)

Output: (26,31,41,41,58,59)

La scelta del migliore algoritmo dipende

- numero di elementi da ordinare
- da quanto gli elementi siano già ordinati
- dispositivo di memoria (metodo d'accesso)

algoritmi e strutture dati



Correttezza

Un algoritmo si dice corretto se, per ogni istanza di input, termina (si ferma) con l'output corretto

Un algoritmo corretto risolve il problema computazionale dato

algoritmi e strutture dati



Efficienza

Un algoritmo si dice efficiente se, per ogni istanza di input, determina l'output corretto nel minor tempo possibile

Un algoritmo efficiente risolve il problema computazionale in un **tempo accettabile**

algoritmi e strutture dati



Insertion sort

L'algoritmo di ordinamento *insertion sort* risulta **efficiente** nel caso in cui il numero di elementi (n) da ordinare è piccolo

...

for $i \leftarrow 2$ to $\text{lenght}[A]$

do $\text{key} \leftarrow A[i]$

pseudocodice

Δ Si inserisce $A[i]$ nella sequenza ordinata $A[1..i-1]$

$j \leftarrow i-1$

while $j > 0$ e $A[j] > \text{key}$

do $A[j+1] \leftarrow A[j]$

$j \leftarrow j-1$

$A[j+1] \leftarrow \text{key}$

algoritmi e strutture dati



Idea per ordinare...

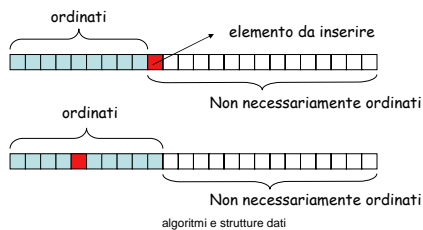


Figure 2.1 Sorting a hand of cards using insertion sort.
algoritmi e strutture dati



Idea per ordinare...

- Ad ogni passo si ha una sottosequenza ordinata in cui inserisco un nuovo elemento dell'input:



algoritmi e strutture dati



Algoritmo: Insertion Sort

la posizione i è indicata

30 (10) 40 20

10 30 (40) 20

10 30 40 (20)

10 20 30 40

algoritmi e strutture dati



Insertion Sort

- Insertion-sort(A)
- 1. for $i=2$ to length(A)
- do key = A[i]
- {insert A[i] in A[1,...,i-1]}
- j = i-1
- while $j>0$ and $A[j]>key$
- do A[j+1] = A[j]
- j=j-1
- A[j+1] = key

algoritmi e strutture dati



Istanza A = {5,2,4,6,1,3}

$i=2$ 5 2 4 6 1 3

$i=3$ 2 5 4 6 1 3

$i=4$ 2 4 5 6 1 3

$i=5$ 2 4 5 6 1 3

$i=6$ 1 2 4 5 6 3

$i=7$ 1 2 3 4 5 6

algoritmi e strutture dati



An Example: Insertion Sort

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

variabili locali

dimensione del problema

passaggio dei parametri

algoritmi e strutture dati



Analisi di Algoritmi

Analizzare un algoritmo vuol dire determinare le risorse necessarie all'algoritmo.

spazio di memoria

e

tempo computazionale

algoritmi e strutture dati



Analisi di Insert-Sort

Il tempo computazionale impiegato dalla procedura Insertion-Sort dipende

- dalla *dimensione input*
- dallo *ordinamento implicito nella sequenza*

Il tempo di esecuzione viene espresso in funzione della **dimensione del problema**, cioè dell'input

algoritmi e strutture dati



Dimensione dell'input

Numero degli elementi dell'input

n

la dimensione dell'array per l'ordinamento

Nota1: se moltiplichiamo due matrici $n \times n$ sarà n^2

Nota2: Il tempo e lo spazio saranno funzioni in n ($TIME(n)$, $SPACE(n)$)

algoritmi e strutture dati



Modello di calcolo

Prima di analizzare un algoritmo abbiamo bisogno di stabilire quale sarà la tecnologia di riferimento utilizzata per eseguire gli algoritmi quando saranno realizzati come programmi

Assumiamo di utilizzare

Mono-Processore + RAM (Random Access Memory)

assenza di concorrenza e parallelismo

algoritmi e strutture dati



Tempo di esecuzione

Numero di operazioni elementari o "passi" eseguiti per il calcolo dell'output

passo \cong una linea di pseudocodice

Hp:

Ogni passo riferito ad una linea i , è eseguito in un tempo costante c_i

algoritmi e strutture dati



Insertion Sort

Statement	Effort
<code>InsertionSort(A, n) {</code>	
<code>for i = 2 to n {</code>	c_1
<code>key = A[i]</code>	c_2
<code>j = i - 1;</code>	c_3
<code>while (j > 0) and (A[j] > key) {</code>	c_4
<code>A[j+1] = A[j]</code>	c_5
<code>j = j - 1</code>	c_6
<code>}</code>	0
<code>A[j+1] = key</code>	c_7
<code>}</code>	0

algoritmi e strutture dati



Calcolo del Tempo d'esecuzione

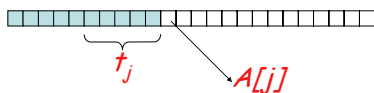
Indichiamo con n_i il numero di volte che un passo i viene eseguito

$$T(n) = c_1 n_1 + c_2 n_2 + \dots + c_k n_k = \sum_{i=1}^k c_i n_i$$

algoritmi e strutture dati



Insertion Sort: analisi del costo computazionale



t_j numero di elementi maggiori di $A[j]$
(t_j il numero di volte che il test del ciclo while viene eseguito)

dipende dai dati in input
 caso ottimo: $t_j = 0$ ($t_j = 1$)
 caso pessimo: $t_j = j-1$ ($t_j = j$)
 caso medio: $t_j = (j-1)/2$ ($t_j = j/2$)

algoritmi e strutture dati



Insertion Sort: analisi del costo computazionale

Insertion-sort(A)

1.	for j=2 to n	C_1	n
2.	do key = A[j]	C_2	$n-1$
3.	i = j-1	C_3	$n-1$
4.	while i>0 and A[i]>key	C_4	$\sum_{j=2}^n t_j$
5.	do A[i+1] = A[i]	C_5	$\sum_{j=2}^n (t_j-1)$
6.	i=i-1	C_6	$\sum_{j=2}^n (t_j-1)$
7.	A[i+1] = key	C_7	$n-1$

algoritmi e strutture dati
 t_j numero di elementi maggiori di A[j]



$$T(n) = c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 \sum t_j + (c_5 + c_6) \sum (t_j - 1) + c_7 (n-1)$$

- t_j numero di elementi maggiori di A[j] - dipende dai dati in input - (t_j numero di test del ciclo while)
- caso ottimo: $t_j = 0 \quad t_j = 1 \quad T(n) = an + b$; lineare
- caso pessimo: $t_j = j-1 \quad t_j = j \quad T(n) = an^2 + bn + c$; quadratico
- caso medio: $t_j = (j-1)/2 \quad t_j = j/2 \quad ???$

algoritmi e strutture dati



Tempo computazionale per il caso ottimo è una funzione lineare

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7) = an + b$$

dove a e b sono costanti

$T(n)$ è una funzione lineare

algoritmi e strutture dati



Tempo computazionale per il caso pessimo è una funzione quadratica

$$\sum_{i=2}^n (i-1) = \frac{n(n-1)}{2}$$

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

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

algoritmi e strutture dati



Tempo computazionale per il caso pessimo è una funzione quadratica

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

$$T(n) = an^2 + bn + c$$

algoritmi e strutture dati



Analisi del Caso Peggior e Caso Medio

Il tempo d'esecuzione nel caso peggiore di un algoritmo rappresenta un limite superiore (upper-bound) sui tempi d'esecuzione per qualsiasi input

La sua conoscenza garantisce che l'algoritmo non impiegherà mai tempi maggiori

Caso medio è approssimativamente negativo quanto il caso peggiore.

Come determinare il caso medio?

algoritmi e strutture dati



Analisi Asintotica

algoritmi e strutture dati



Analisi asintotica

Obiettivo: semplificare l'analisi del tempo di esecuzione di un algoritmo prescindendo dai dettagli implementativi o di altro genere. **Classificare le funzioni in base al loro comportamento asintotico.**

Astrazione: come il tempo di esecuzione cresce *asintoticamente* in funzione della dimensione dell'input.

Asintoticamente **non significa per tutti gli input.**

Esempio: input di piccole dimensioni.

algoritmi e strutture dati



"O grande"

- limite superiore asintotico
 - $f(n) = O(g(n))$, se esistono due costanti c e n_0 , l.c.
 - $0 \leq f(n) \leq cg(n)$ per $n \geq n_0$
 - $f(n)$ e $g(n)$ sono funzioni non negative
- notazione O si usa per dare un limite superiore ad una funzione a meno di un fattore costante
 - O viene usata nell'analisi del caso pessimo

algoritmi e strutture dati


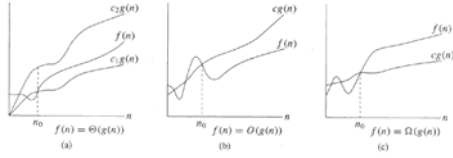




Figure 3.1 Graphic examples of the Θ , O , and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. (a) Θ -notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0 , c_1 , and c_2 such that to the right of n_0 , the value of $f(n)$ always lies between $c_1g(n)$ and $c_2g(n)$ inclusive. (b) O -notation gives an upper bound for a function to within a constant factor. We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that to the right of n_0 , the value of $f(n)$ always lies on or below $cg(n)$. (c) Ω -notation gives a lower bound for a function to within a constant factor. We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that to the right of n_0 , the value of $f(n)$ always lies on or above $cg(n)$.

algoritmi e strutture dati



“ Ω grande”

- Limite inferiore asintotico
 - $f(n) = \Omega(g(n))$ se esistono due costanti c e n_0 , i.e.
 $0 \leq cg(n) \leq f(n)$ per $n \geq n_0$
- Usato per
 - tempo di esecuzione nel *caso ottimo*;
 - limiti inferiori di complessità;
 - Esempio: il limite inferiore per la ricerca in array non ordinati è $\Omega(n)$.

algoritmi e strutture dati


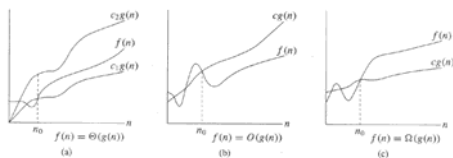



Figure 3.1 Graphic examples of the Θ , O , and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. (a) Θ -notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0 , c_1 , and c_2 such that to the right of n_0 , the value of $f(n)$ always lies between $c_1g(n)$ and $c_2g(n)$ inclusive. (b) O -notation gives an upper bound for a function to within a constant factor. We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that to the right of n_0 , the value of $f(n)$ always lies on or below $cg(n)$. (c) Ω -notation gives a lower bound for a function to within a constant factor. We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that to the right of n_0 , the value of $f(n)$ always lies on or above $cg(n)$.

algoritmi e strutture dati



“Θ”

- “tight bound” = approssimazione *stretta*.
- $f(n) = \Theta(g(n))$ se esistono c_1, c_2 e n_0 , t.c.
 $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ per $n \geq n_0$

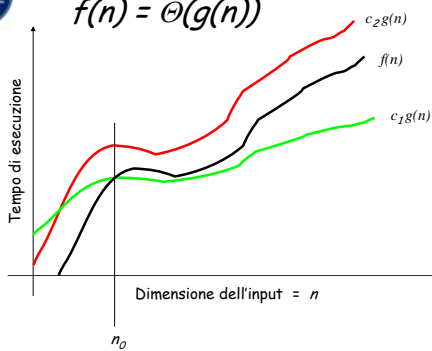
Teorema: Per ogni coppia $f(n)$ e $g(n)$, $f(n) = \Theta(g(n))$ se e solo se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$

- $O(f(n))$ spesso usato erroneamente al posto di $\Theta(f(n))$

algoritmi e strutture dati



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



algoritmi e strutture dati



“o piccolo” e “ω piccolo”

- $f(n) = o(g(n))$

Analogo stretto di O grande

- Per ogni $c > 0$, esiste $n_0 > 0$, t.c. $f(n) < c g(n)$ per $n \geq n_0$
- Usato per confrontare tempi di esecuzione. Se $f(n) = o(g(n))$ diciamo che $g(n)$ *domina* $f(n)$.

Il concetto intuitivo nella notazione o è che la funzione $f(n)$ diventa trascurabile rispetto a $g(n)$ per n che tende all'infinito.

- $f(n) = \omega(g(n))$ analogo stretto di Ω grande.

algoritmi e strutture dati



Notazione Asintotica

- Analogie tra le proprietà relazionali dei numeri reali

- $f(n) = O(g(n))$ $f \leq g$
- $f(n) = \Omega(g(n))$ $f \geq g$
- $f(n) = \Theta(g(n))$ $f = g$
- $f(n) = o(g(n))$ $f < g$
- $f(n) = \omega(g(n))$ $f > g$

- Abuso di notazione: $f(n) = O(g(n))$
- Versione corretta: $f(n)$ appartiene a $O(g(n))$

algoritmi e strutture dati



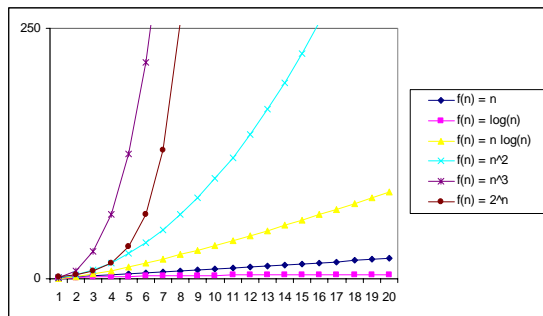
Limiti e notazione asintotica

- $f(n)/g(n) \rightarrow c$ allora $f(n) = \Theta(g(n))$
- $f(n)/g(n) \rightarrow 0$ allora $f(n) = o(g(n))$
- $f(n)/g(n) \rightarrow \infty$ allora $f(n) = \omega(g(n))$

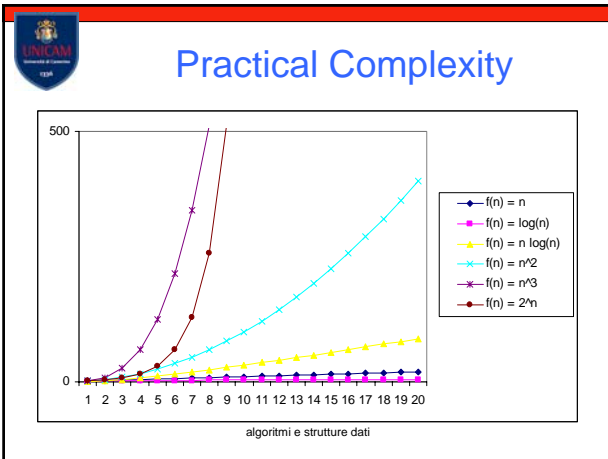
algoritmi e strutture dati

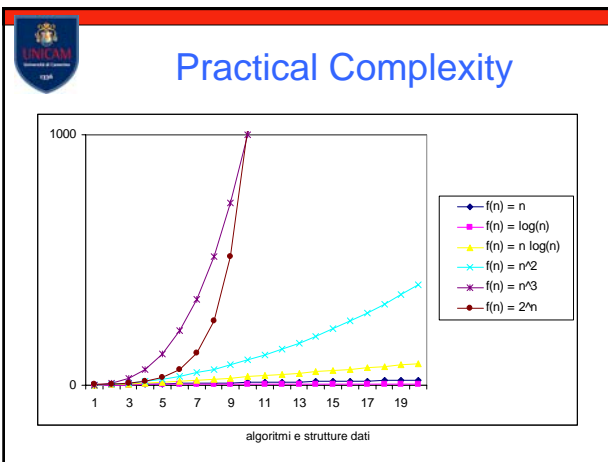


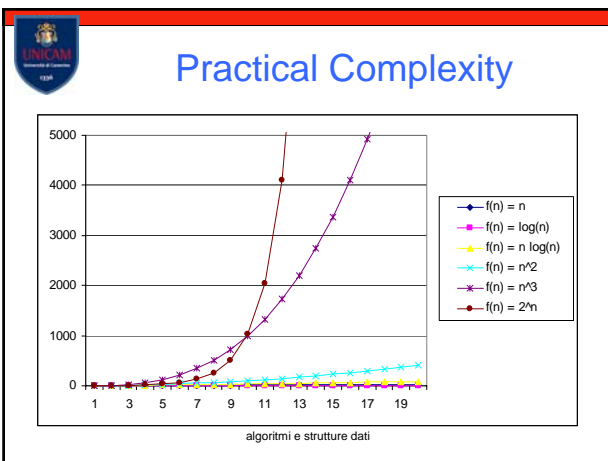
Practical Complexity



algoritmi e strutture dati









Practical Complexity

