Titolo unità didattica: Efficienza degli algoritmi [10]

Titolo modulo : Definizione di complessità di tempo e di spazio di un algoritmo [01-T]

Le funzioni Complessità di tempo e Complessità di spazio

Argomenti trattati:

- ✓ dimensione computazionale di un problema
- ✓ complessità di tempo di un algoritmo
- ✓ complessità di spazio di un algoritmo
- ✓ benchmark e analisi sperimentale di complessità

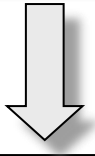
Prerequisiti richiesti: P1-07-*-T

efficienza di un algoritmo

tempo e **memoria**
di un calcolatore sono
risorse di calcolo

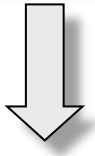
la valutazione delle **prestazioni** di un programma è la quantificazione dell'uso delle **risorse di calcolo** necessarie per la sua esecuzione

il **numero totale** delle **operazioni** e dei **dati** di un **algoritmo** **è proporzionale** al **tempo** e alla **memoria** richiesti per l'esecuzione del **programma** che implementa l'algoritmo su uno specifico calcolatore

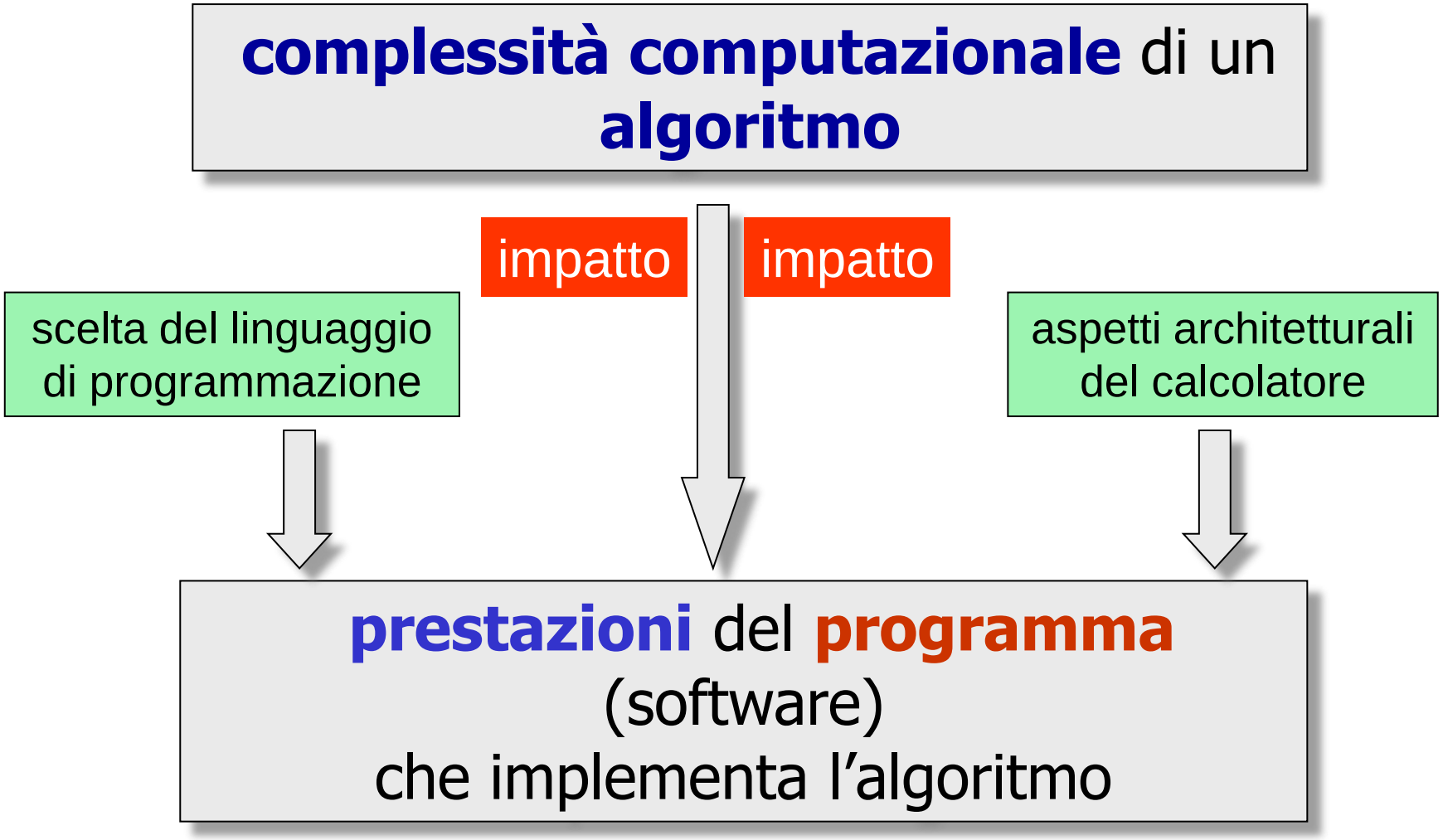


sono sinonimi

- ✓ **analisi (costo) di un algoritmo**
- ✓ **efficienza di un algoritmo**
- ✓ **complessità computazionale di un algoritmo**



determinare la **quantità di risorse necessarie** per l'esecuzione di un algoritmo

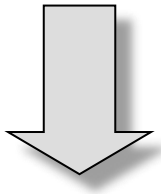

complessità computazionale di un **algoritmo**
impatto
impatto
scelta del linguaggio di programmazione
aspetti architetturali del calcolatore
prestazioni del **programma** (software) che implementa l'algoritmo

in un algoritmo, il **numero complessivo** delle operazioni da eseguire dipende dal **numero dei dati di input**

numero dei dati di input
=
dimensione computazionale
del (istanza del) **problema**

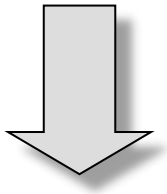
quando la **dimensione computazionale** di un problema **cresce**, come cambiano il numero delle operazioni (tempo di esecuzione) e il numero dei dati (memoria) dell'algoritmo ?

la **Teoria della Complessità Computazionale** investiga i problemi legati alla **quantità di risorse** richieste per l'esecuzione di algoritmi e la **difficoltà intrinseca** di progettare algoritmi efficienti per risolvere specifici problemi



- **complessità di tempo** di un algoritmo
- **complessità di spazio** di un algoritmo
- **difficoltà intrinseca di un problema**

la **Teoria della Complessità Computazionale** investiga i problemi legati alla **quantità di risorse** richieste per l'esecuzione di algoritmi e la **difficoltà intrinseca** di progettare algoritmi efficienti per risolvere specifici problemi



classificare
gli **algoritmi** e i **problemi**
in **classi di complessità**

complessità di tempo di un algoritmo

- individuare la **dimensione computazionale** del problema
- individuare l'**operazione dominante** (o le operazioni dominanti) dell'algoritmo

la **complessità di tempo** $T(n)$
di un algoritmo è la funzione che esprime il
numero di operazioni dominanti
in dipendenza della
dimensione computazionale n
del problema

complessità di tempo di un algoritmo

algoritmo	operazione dominante	T(n) , T(n,m)
somma_array	somma	n
somma_quad_rec	somma, prodotto	n somme, 2n prodotti
ricerca_sequenziale	confronto	n (al più)
massimo_array	confronto	n-1
massimo_a2D	confronto	nm
somma_a2Dtriang	somma	n(n+1)/2
unione	confronto	nm (al più)
intersezione	confronto	nm (al più)
uguaglianza_insiemi	confronto	n^2 (al più)
fusione	confronto	n+m (al più)

n e m individuano la dimensione computazionale, secondo il significato assunto nei vari problemi

complessità di spazio di un algoritmo

- individuare la **dimensione computazionale** del problema

la **complessità di spazio** $S(n)$ di un algoritmo è la funzione che esprime il **size** totale delle strutture dati utilizzate per memorizzare **dati di input, locali e di output**, in dipendenza della **dimensione computazionale** n del problema

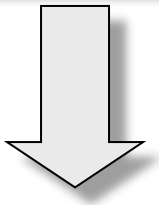
complessità di spazio di un algoritmo

algoritmo	$S(n)$, $S(n,m)$
somma_array	n
somma_quad_rec	0
ricerca_sequenziale (dati esterni)	0
massimo_array	n
massimo_a2D	nm
somma_a2Dtriang	nm
unione	2(n+m) al più
intersezione	n+m+min(n,m) al più
uguaglianza_array	2n
fusione	2(n+m)

n e m individuano la dimensione computazionale, secondo il significato assunto nei vari problemi

la **complessità di tempo** e la **complessità di spazio** possono **dipendere anche dai valori dei dati di input**

(per esempio: algoritmi di ricerca $(T(n))$**, di unione** $(T(n), S(n))$**, di intersezione** $(T(n), S(n))$**, uguaglianza di array** $(T(n))$**,….)**



complessità di tempo e complessità di spazio del caso peggiore

```
for (i=0; i<n; i++) {
    q  operazioni
       dominanti
}
```

complessità di tempo

$T(n) = qn$

cioè $T(n)$ è proporzionale a n

```
for (i=0; i<n; i++) {
    q  operazioni
       dominanti
}
for (j=0; j<n; j++) {
    q  operazioni
       dominanti
}
```

complessità di tempo

$T(n) = 2qn$

cioè $T(n)$ è proporzionale a $2n$

(proporzionale a n)

complessità di tempo

LINEARE

```
for (i=0; i<n;i++) {
  for (j=0; j<n; j++) {
    q  operazioni
       dominanti
  }
}
```

complessità di tempo

$T(n) = q\, n^2$

cioè $T(n)$ è proporzionale a n^2

```
for (i=0; i<n; i++) {
  for (j=i; j<n; j++) {
    q  operazioni
       dominanti
  }
}
```

complessità di tempo

$T(n) = q(1+2+3+\ldots+n)$

$= qn(n+1)/2$

cioè $T(n)$ è proporzionale a n^2

complessità di tempo **QUADRATICA**

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
    for (k=0; k<n; k++) {
      q  operazioni
         dominanti
    }
  }
}
```

complessità di tempo

$$T(n) = q\, n^3$$

cioè $T(n)$ è proporzionale a n^3

complessità di tempo
CUBICA

```
i = 0 ;
do {
    i = i+1 ;
        q  operazioni
            dominanti
} while (2^i < n);
```

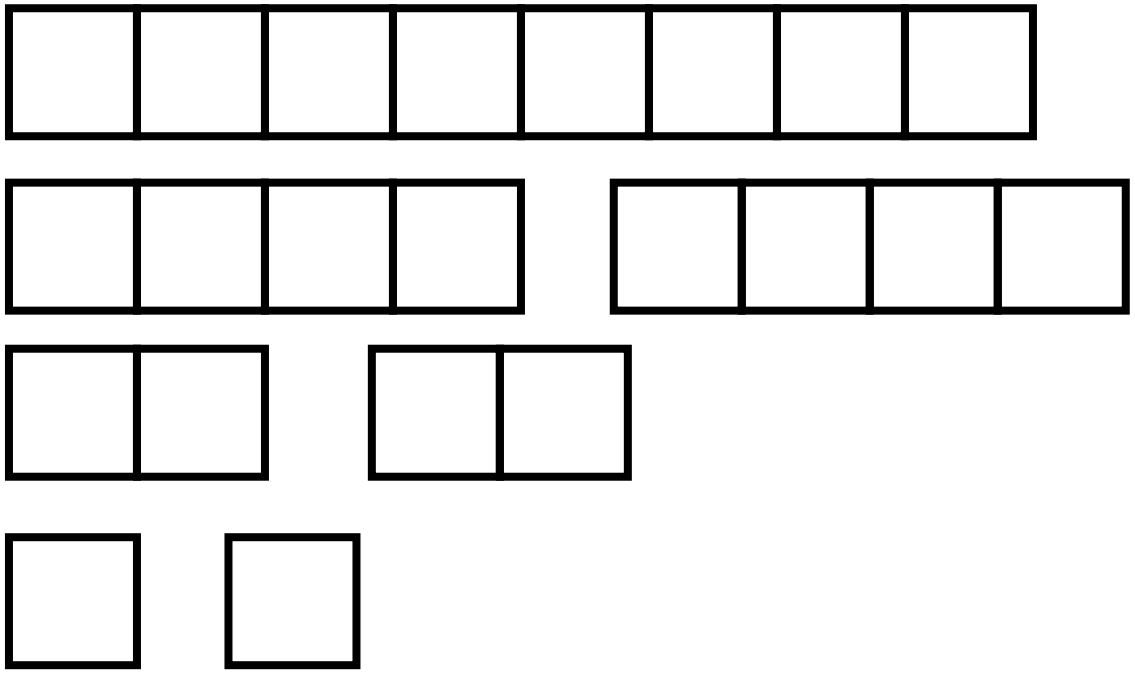
complessità di tempo

$$T(n) = q \log_2 n$$

cioè proporzionale a $\log_2 n$

complessità di tempo LOGARITMICA

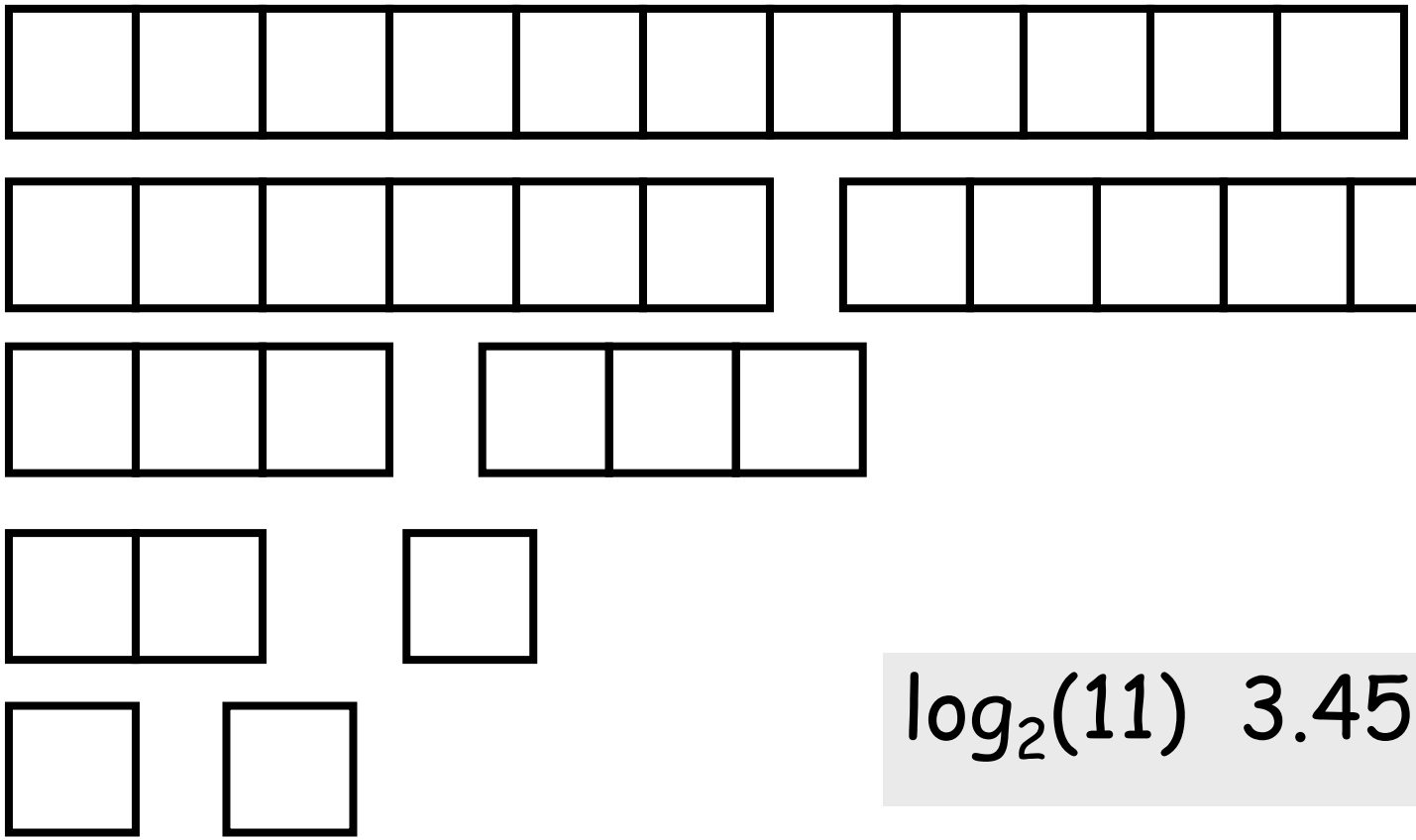
quante volte si può dividere a metà un array di size n?



n=8

n potenza di 2: $\log_2(n)$ volte

quante volte si può dividere a metà un array di size n?



n=11

$\log_2(11)$ 3.459… ⟶ 4

in generale:
più piccolo intero maggiore di $\log_2(n)$ volte
(detto ceiling e denotato con $\lceil log_2\, n \rceil$)

complessità computazionale di un algoritmo

Esempio: data una sequenza di **n** numeri interi ordinata in senso **crescente** e memorizzata in un array 1D, costruire la sequenza ordinata in senso **decrescente**

2	3	5	6	8	9	12	15	18	22	25	32

la soluzione del problema è costituita dagli stessi elementi della sequenza di input, ma ordinati nel senso opposto

32	25	22	18	15	12	9	8	6	5	3	2

complessità computazionale di un algoritmo

```
void ordine_inverso_1(in: int n; inout: int a[]) {
 int i;
 for (i=0; i < n/2; i++) {        /* n pari  */
      scambiare_i (&a[i], &a[n-1-i]) ;
 }
}
```

```
void ordine_inverso_2(in: int a[], int n; out: int b[]) {
int i;
 for (i=0; i < n; i++) {
      b[i] = a[n-1-i];
  }
}
```

complessità computazionale di un algoritmo

S(n)

ordine_inverso_1 : S(n) = n

ordine_inverso_2 : S(n) = 2n

T(n) : operazione dominante: **assegnazione**

ordine_inverso_1 : T(n) = 3/2 n

ordine_inverso_2 : T(n) = n

ordine_inverso_1 ha una complessità di tempo maggiore, ma una complessità di spazio minore
(opera *in place*)

ordine_inverso_2 ha una complessità di tempo minore, ma una complessità di spazio maggiore

informazioni sulla complessità di tempo di un algoritmo possono essere ricavate sperimentalmente, mediante esecuzioni di un programma che implementa l'algoritmo

eseguire il programma per risolvere il problema di dimensione **n** e poi **2n**, **4n**, **8n**, **16n** e infine analizzare i tempi di esecuzione

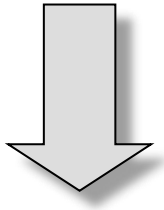
n	**tempo di esecuzione (sec)**	**fattore di incremento dei tempi di esecuzione (attuale/precedente)**

Esempio: programma che implementa l'algoritmo media_array

n	tempo di esecuzione (sec)	fattore di incremento dei tempi di esecuzione (attuale/precedente)
1000	0.0201	
2000	0.0408	2.004
4000	0.0793	1.967
8000	0.1611	2.031
16000	0.3182	1.975

fattore di incremento *quasi* costante e circa uguale a **2**

se si **raddoppia** la dimensione
il tempo di esecuzione **raddoppia**



la complessità di tempo $T(n)$ dell'algoritmo è proporzionale a n

se la complessità di tempo T(n) dell'algoritmo è proporzionale a n

$$T(n) = \alpha \cdot n$$

allora:

$$T(n) = \alpha \cdot n$$
$$T(2n) = \alpha \cdot 2n$$

e deve accadere che:

$$\frac{T(2n)}{T(n)} = \frac{\alpha \cdot 2n}{\alpha \cdot n} = 2$$

se la complessità di tempo T(n) dell'algoritmo è proporzionale a n^2

$$T(n) = \alpha \cdot n^2$$

allora:

$$T(n) = \alpha \cdot n^2$$

$$T(2n) = \alpha \cdot (2n)^2 = \alpha \cdot 4n^2$$

e deve accadere che:

$$\frac{T(2n)}{T(n)} = \frac{\alpha \cdot 4n^2}{\alpha \cdot n^2} = 4$$

se la complessità di tempo T(n) dell'algoritmo è proporzionale a n^3

$$T(n) = \alpha \cdot n^3$$

allora:

$$T(n) = \alpha \cdot n^3$$

$$T(2n) = \alpha \cdot (2n)^3 = \alpha \cdot 8n^3$$

e deve accadere che:

$$\frac{T(2n)}{T(n)} = \frac{\alpha \cdot 8n^3}{\alpha \cdot n^3} = 8$$

esperimenti (**benchmark**)

fattore di incremento dei tempi di esecuzione, raddoppiando la dimensione computazionale	forma presumibile della complessità di tempo
2	proporzionale a n
4	proporzionale a n^2
8	proporzionale a n^3
16	proporzionale a n^4

complessità di tempo **proporzionale a una POTENZA di** n

complessità di tipo **polinomiale**