Algoritmi: retrospettiva

Algoritmo (termine derivante dal nome di un matematico usbeco del IX secolo d.C.) = procedimento di calcolo, sequenza di azioni che devono essere eseguite per giungere alla risoluzione di un problema computazionale; nozione antecedente (già registrata in papiri del XVII secolo a.C. con riferimenti a documenti di dodici secoli prima) e diversa rispetto a quella di dimostrazione matematica, risalente a Euclide (III secolo a.C.)

Teoria degli algoritmi: assestata nella prima metà del XX secolo

Tecniche di progettazione e analisi (di correttezza ed efficienza) di algoritmi: evolutesi nella seconda metà del XX secolo, grazie alla diffusione dei calcolatori elettronici

Esempi di algoritmi

- controllo del volo di aerei
- regolazione di reazioni nucleari
- reperimento di info in archivi di dati
- smistamento di comunicazioni telefoniche
- gioco degli scacchi
- controllo della produzione di catene di montaggio
- traduzione di programmi da un linguaggio di programmazione a un altro
- progetto di nuovi calcolatori
- riconoscimento di immagini
- elaborazione di testi letterari
- composizione di brani musicali

Algoritmi e strutture dati: una disciplina

È dedicata allo studio della rappresentazione e manipolazione dell'info al fine di trovare buone soluzioni algoritmiche a specifici problemi ben formalizzati

Attività trasversale, che trova applicazione in tutte le aree disciplinari dell'informatica, ma che nel contempo è dotata di metodi e autonomia propri, come riconosciuto da ACM (Association for Computing Machinery) che suddivide la Computer Science in 9 branche:

- Algoritmi e strutture dati
- Linguaggi di programmazione
- Architetture dei calcolatori
- Sistemi operativi
- Ingegneria del software
- Calcolo numerico e simbolico
- Basi di dati e sistemi per il reperimento dell'informazione
- Intelligenza artificiale
- Visione e robotica

Una prospettiva informale

La nozione di dato è inscindibilmente legata a quella di algoritmo, perché, per risolvere un problema computazionale, è necessario organizzare ed elaborare dati

Algoritmo = manipolatore di dati che, a fronte di dati di ingresso che descrivono il problema da risolvere, produce altri dati di uscita come risultato del problema

Una prospettiva più formale

Algoritmo = procedura computazionale ben definita che assume un insieme di valori come input e produce un insieme di valori come output; sequenza di passi computazionali che trasformano l'input nell'output

Problema = relazione che deve valere fra input ed output

Esempio: problema dell'ordinamento (in ordine non decrescente)

Input: una sequenza di n numeri $<a_1, a_2, \ldots, a_n>$

Output: una permutazione (riordinamento) $<a'_1, a'_2, \ldots, a'_n>$ della sequenza di input tale che $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Una istanza del problema si ottiene in corrispondenza di una precisa sequenza di input, ad es. <31, 41, 59, 26, 41, 58>

Correttezza di un algoritmo

Algoritmo corretto = algoritmo che risolve il problema = algoritmo che, per ogni istanza di input, termina e produce l'output corretto

N.B. Se per una o più istanze del problema l'algoritmo non termina e/o non produce un output corretto, esso si dice scorretto
Algoritmi scorretti possono essere utili se il loro tasso di errore può essere controllato

In seguito si esamineranno solo algoritmi corretti

Descrizione di un algoritmo

Si può usare il linguaggio corrente (definito però senza ambiguità), un linguaggio di programmazione, un progetto hw

In seguito adotteremo uno pseudocodice, che mescola linguaggio naturale e linguaggio di programmazione senza gestire aspetti di ingegneria del sw (quali astrazione dei dati, modularità, gestione degli errori)

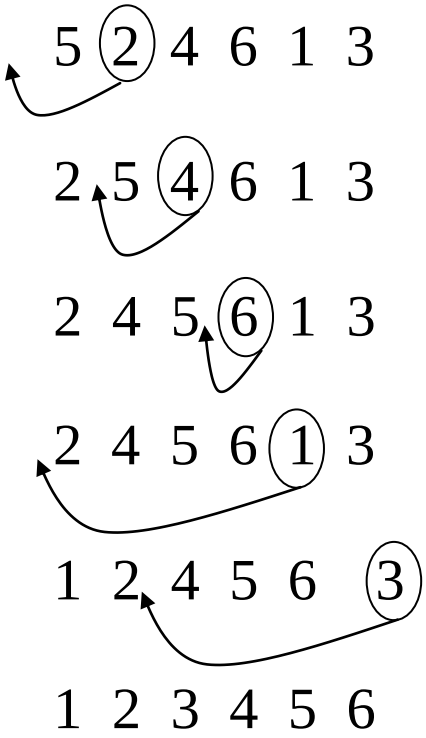
INSERTION-SORT: descrizione in pseudocodice

È il tipico procedimento con cui si ordina una mano di carte

```
INSERTION-SORT(A)
1 for j ← 2 to length[A]
2    do  key ← A[j]
3        ▷ Si inserisce A[j] nella
         sequenza ordinata A[1 .. j - 1]
4        i ← j - 1
5        while i > 0 e A[i] > key
6            do  A[i + 1] ← A[i]
7                i ← i - 1
8        A[i + 1] ← key
```

INSERTION-SORT: funzionamento

A = <5, 2, 4, 6, 1, 3 >



Pseudocodice: sintassi e semantica

- Le rientranze indicano la struttura dei blocchi (al posto di parentesi o **begin-end**, così da ridurre la lunghezza della rappresentazione)
- Strutture di controllo
  Iterative **while-do**, **for-do**, **repeat-until**
  Condizionali **if-then-else**
- Il simbolo $\triangleright$ introduce una linea di commento
- Il simbolo $\leftarrow$ è l'operatore di assegnamento, sia per variabili di tipo semplice, sia per variabili di tipo aggregato; l'assegnamento multiplo

  $i \leftarrow j \leftarrow e$

  equivale a

  $j \leftarrow e$
  $i \leftarrow j$
- Tutte le variabili, se non diversamente specificato, sono locali

Pseudocodice: sintassi e semantica (cont.)

- Array. A[i] indica l'i-esimo elemento dell'array A, A[1..j] denota il sottoarray contenente A[1], A[2], …, A[j]
- Dati composti. Sono organizzati come oggetti; un attributo di un oggetto si indica col suo nome seguito dal nome dell'oggetto fra parentesi quadre, es. length[A] denota l'attributo length dell'oggetto (array) A.
  Una variabile che rappresenta un array o, in generale, un oggetto è trattata come un puntatore ai dati contenuti nell'array o nell'oggetto. Così, dopo l'assegnamento y ← x fra dati composti, x e y sono lo stesso oggetto
- Passaggio dei parametri. Avviene solo per valore; quando si passano oggetti, viene passata una copia del puntatore ai dati
- $\lceil x \rceil$, detto *tetto* o *parte intera superiore*, denota il più piccolo intero $\geq x$
- $\lfloor x \rfloor$, detto *base* o *parte intera inferiore*, denota il più grande intero $\leq x$

Studio di algoritmi

Tre ambiti principali:

- Analisi: dato un algoritmo e un problema, provare la correttezza dell'algoritmo e valutare la quantità di risorse da esso utilizzate (complessità concreta) mediante tecniche matematiche (fra cui le relazioni di ricorrenza)
- Sintesi (o progetto): dato un problema, costruire un algoritmo per risolverlo (fra i metodi di sintesi di un algoritmo si citano ricorsione, "divide et impera", programmazione dinamica e tecniche "greedy")
- Classificazione (o complessità strutturale): data una quantità di risorse, individuare la classi di problemi risolubili da algoritmi che usano al più tale quantità (es. classi P e NP)

Analisi della complessità concreta

Consiste nel prevedere le richieste dell'algoritmo in termini di risorse, cioè memoria, porte di comunicazione, tempo di computazione, così da potere confrontare diversi algoritmi candidati per la soluzione di un problema; è necessario stabilire un modello univoco di computazione dell'esecutore usato, cosicché siano noti sia i modelli delle risorse, sia i loro costi

Usualmente viene adottato un modello computazionale RAM mono-processore, secondo cui le istruzioni sono eseguite una dopo l'altra senza operazioni concorrenti ($\rightarrow$ in tal modo si tratta unicamente la teoria degli algoritmi sequenziali)

Tempo di esecuzione di un algoritmo

Tempo di esecuzione (per un particolare input) = numero di operazioni elementari o "passi" eseguiti per il calcolo dell'output corrispondente

Passo: nozione da definire in modo il più possibile indipendente da ogni macchina; ad es. si può fare la seguente ipotesi (compatibile col modello RAM):

ogni riga *i* richiede un tempo costante c_i
che può essere diverso da quello delle altre linee

Tempo di esecuzione di un algoritmo

Nell'esempio proposto e in generale si osserva però che, a parità di algoritmo A, due input x e x'

$x \in \mathfrak{I}$, $x' \in \mathfrak{I}$, $\mathfrak{I}$ = insieme delle istanze,

della stessa dimensione

$|x| = |x'|$

possono dare luogo a tempi di esecuzione diversi

$T_A(x) \neq T_A(x')$

Domanda: come definire il tempo di calcolo di A in funzione della sola dimensione dell'input?

Risposta: considerare solo il tempo peggiore su tutti gli input di dimensione *n* fissata, oppure considerare il tempo medio

Analisi della complessità concreta (cont.)

Il comportamento di un algoritmo può essere diverso per input diversi (cioè per istanze diverse del problema) → è necessario generalizzare

Ad es. il tempo di calcolo di INSERTION-SORT dipende da

- dimensione dell'input (l'ordinamento di un migliaio di numeri richiede più tempo di quello di tre numeri)
- ordine parziale dell'elenco di numeri da ordinare

Dimensione dell'input = è una misura scelta di volta in volta, a seconda del problema affrontato. Ad es. si può scegliere

- il numero degli elementi dell'input per un problema di ordinamento
- il numero totale di bit necessari per rappresentare ciascun fattore in un problema di calcolo di moltiplicazioni
- la coppia (numero di nodi, numero di archi) per un problema di visita di un grafo

Tempo di esecuzione di INSERTION-SORT

INSERTION-SORT(A)	*costo*	*n° di volte*
1 **for** j ← 2 **to** length[A]	c_1	n (=length[A])
2 **do** key ← A[j]	c_2	$n - 1$
3 ▷ Si inserisce A[j] nella sequenza ordinata A[1 .. j – 1]		
4 i ← j – 1	c_4	$n - 1$
5 **while** i > 0 e A[i] > key	c_5	$\sum_{j=2}^{n} t_j$
6 **do** A[i + 1] ← A[i]	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 i ← i – 1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 A[i + 1] ← key	c_8	$n - 1$

t_j = numero di volte che il test del ciclo **while** alla linea 5 viene eseguito per quel valore di j

Tempo di esecuzione di INSERTION-SORT

Nel caso migliore
(che si ha quando l'array è già ordinato in ordine non decrescente $\rightarrow t_j = 1$)

$T(n) = c_1 n + (c_2 + c_4 + c_5 + c_8)(n - 1) =$
$= (c_1 + c_2 + c_4 + c_5 + c_8) n - (c_2 + c_4 + c_5 + c_8) \rightarrow T(n)$ è una funzione lineare di n

Nel caso peggiore
(che si ha quando l'array è ordinato in ordine decrescente $\rightarrow t_j = j$)

$$T(n) = c_1 n + c_2 (n-1) + c_4 (n-1) + c_5 (\frac{n(n+1)}{2} - 1) + c_6 (\frac{n(n-1)}{2}) + c_7 (\frac{n(n-1)}{2}) + c_8 (n-1) =$$

$$= (\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}) n^2 + (c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8) n - (c_2 + c_4 + c_5 + c_8)$$

$\rightarrow T(n)$ è una funzione quadratica di n

Analisi del caso peggiore e del caso medio

Solitamente si analizza solo il tempo di esecuzione nel caso peggiore perché

- È il limite superiore del tempo esecuzione
- Per alcuni algoritmi il caso peggiore si verifica abbastanza di frequente
- Il "caso medio" è, in genere, cattivo quanto il peggiore (ad es. per INSERTION-SORT, nel caso medio $t_j = j/2$, il che porta a un $T(n)$ quadratico)

Tuttavia il caso peggiore spesso fornisce una valutazione troppo pessimistica

Per definire il caso medio, è necessario stabilire cos'è un input medio (spesso si assume che tutti gli input della stessa dimensione siano ugualmente probabili, ipotesi discutibile in molte applicazioni)

Caso peggiore e caso medio

Dato un algoritmo A, si definisce

- complessità nel caso peggiore la funzione $T_A^p: N \to N$ tale che, $\forall n \in N$

$$T_A^p(n) = \max\{T_A(x) \mid |x| = n\}$$

- complessità nel caso medio, assunto che gli input della stessa dimensione siano equiprobabili, la funzione $T_A^m: N \to R$ tale che, $\forall n \in N$

$$T_A^m(n) = \frac{\sum_{|x|=n} T_A(x)}{I_n}$$

dove I_n è il n° di istanze di dimensione n

Astrazioni nel calcolo della complessità

Astrazione compiuta nel calcolo di *T(n)* di INSERTION-SORT: ignorato il costo effettivo di ciascuna istruzione, che si è assunto pari a una costante simbolica, dal momento che è difficile/impossibile da determinare, dipendendo da

- compilatore usato
- elaboratore
- …

Il livello di dettaglio resta comunque eccessivo se si intende solo stabilire il tipo (la forma) della funzione *T(n)*

Analisi asintotica

Ulteriore astrazione: considerare solo il tasso di crescita (o ordine di grandezza) di $T(n)$ al tendere di n a $+\infty$, che fornisce una valutazione della rapidità di incremento del tempo di calcolo al crescere delle dimensioni del problema $\rightarrow$

- considerare solo il termine principale (ovvero quello che rappresenta la maggiore potenza di n) dato che i termini di ordine inferiore sono relativamente non significativi per valori di n molto grandi
- ignorare il coefficiente del termine principale dato che è meno significativo del valore della potenza per input grandi

$\downarrow$

INSERTION-SORT nel caso peggiore ha un $T(n)$ pari a $\Theta(n^2)$
(pronunciato “theta di n al quadrato”)

Analisi asintotica (cont.)

Un algoritmo si considera più efficiente di un altro se il suo *T(n)* nel caso peggiore ha un ordine di grandezza inferiore, ad es. un algoritmo $\Theta(n^2)$ si reputa più efficiente di un algoritmo $\Theta(n^3)$

Questa valutazione può essere errata per input piccoli

Prestazioni degli algoritmi

Tabella formulata assumendo una velocità di calcolo di 10^6 istruzioni /sec
(è sottointeso che le istruzioni considerate sono elementari)

Complessità	n=10	n=20	n=50	n=100	$n=10^3$	$n=10^4$	$n=10^5$	$n=10^6$
n	10 μs	20 μs	50 μs	0,1 ms	1 ms	10 ms	0,1 s	1 s
n*lg*n	33,2 μs	86,4 μs	0,28 ms	0,6 ms	9,9 ms	0,1 s	1,6 s	19,9 s
n^2	0,1 ms	0,4 ms	2,5 ms	10 ms	1 s	100 s	2,7 h	11,5 g
n^3	1 ms	8 ms	125 ms	1 s	16,6 mn	11,5 g	31,7 a	≈ 300 c
2^n	1 ms	1 s	35,7 a	≈ 10^{14} c	∞	∞	∞	∞
3^n	59 ms	58 mn	≈ 10^8 c	∞	∞	∞	∞	∞

lg = logaritmo in base 2
μs = microsecondi
ms =millisecondi
s = secondi
mn = minuti
h = ore
g = giorni
a = anni
c = secoli
∞ = periodo superiore al millenio

Prestazioni degli algoritmi (cont.)

Tabella formulata assumendo una velocità di calcolo di 10^6 istruzioni/sec

	Complessità	Massima dimensione degli ingressi che possono essere elaborati in un minuto
lineare	n	6×10^7
quadratico	n*lg*n	28×10^5
polinomiale	n^2	77×10^2
	n^3	390
cubico	2^n	25
esponenziale		

Prestazioni degli algoritmi (cont.)

Algoritmi con complessità

- lineare o di poco superiore (n*lg*n) sono utilizzabili in modo efficiente anche per elevate dimensioni dell'input $\rightarrow$ trovare procedure con tale complessità è uno dei primi obiettivi della progettazione di algoritmi
- n^k risultano inaccettabili per input di lunghezza elevata quando $k \geq 3$
- a^n sono proibitivi per qualunque $a > 1$ anche per dimensioni di input limitate

Tutte le precedenti considerazioni sulle prestazioni degli algoritmi sono indipendenti dal livello tecnologico dei calcolatori $\rightarrow$ non saranno superate con l'avvento di elaboratori sensibilmente più veloci

Effetti della velocità di calcolo sulle prestazioni degli algoritmi

Complessità	Massima dimensione degli ingressi che possono essere elaborati in un tempo t sul calcolatore C1	Massima dimensione degli ingressi che possono essere elaborati in un tempo t sul calcolatore C2 che è M volte più veloce di C1
n	d_1	$M \cdot d_1$
$n\lg n$	d_2	$\approx M \cdot d_2$ (per $d_2 >> 0$)
n^2	d_3	$\sqrt{M} \cdot d_3$
2^n	d_4	$d_4 + \lg M$

↓

Algoritmi	**Vantaggio derivabile dal passaggio a una tecnologia più potente**
Linerari o quasi lineari ($n\lg n$)	Sostanziale
Polinomiali	Evidente ma smorzato
Esponenziali	Ininfluente

Progetto di algoritmi

Esistono molte tecniche di progetto, fra cui

- Paradigma incrementale (è quello adottato da INSERTION-SORT, che opera un ordinamento progressivo di sottoarray)
- Paradigma divide-et-impera, che è molto usato e consente un semplice calcolo di *T(n)* (è quello adottato da MERGE-SORT)

Paradigma divide-et-impera

A ciascun livello di ricorsione si struttura in 3 passi:

- Divide: il problema viene suddiviso in sottoproblemi simili a quello originale ma di dimensioni inferiori
- Impera: ogni sottoproblema viene risolto direttamente, se è sufficientemente piccolo, o ricorsivamente, invocando lo stesso algoritmo
- Combina: le soluzioni dei sottoproblemi sono combinate per ottenere la soluzione del problema di partenza

MERGE-SORT

- Divide: divide gli n elementi da ordinare in due sottosequenze di $n/2$ elementi ciascuna
- Impera: le due sottosequenze vengono ordinate; ogni sottosequenza di lunghezza unitaria è una soluzione, ogni sottosequenza di lunghezza ≥ 2 viene ordinata invocando ricorsivamente MERGE-SORT
- Combina: fonde le due sottosequenze ordinate per produrre un'unica sequenza ordinata degli n elementi di partenza

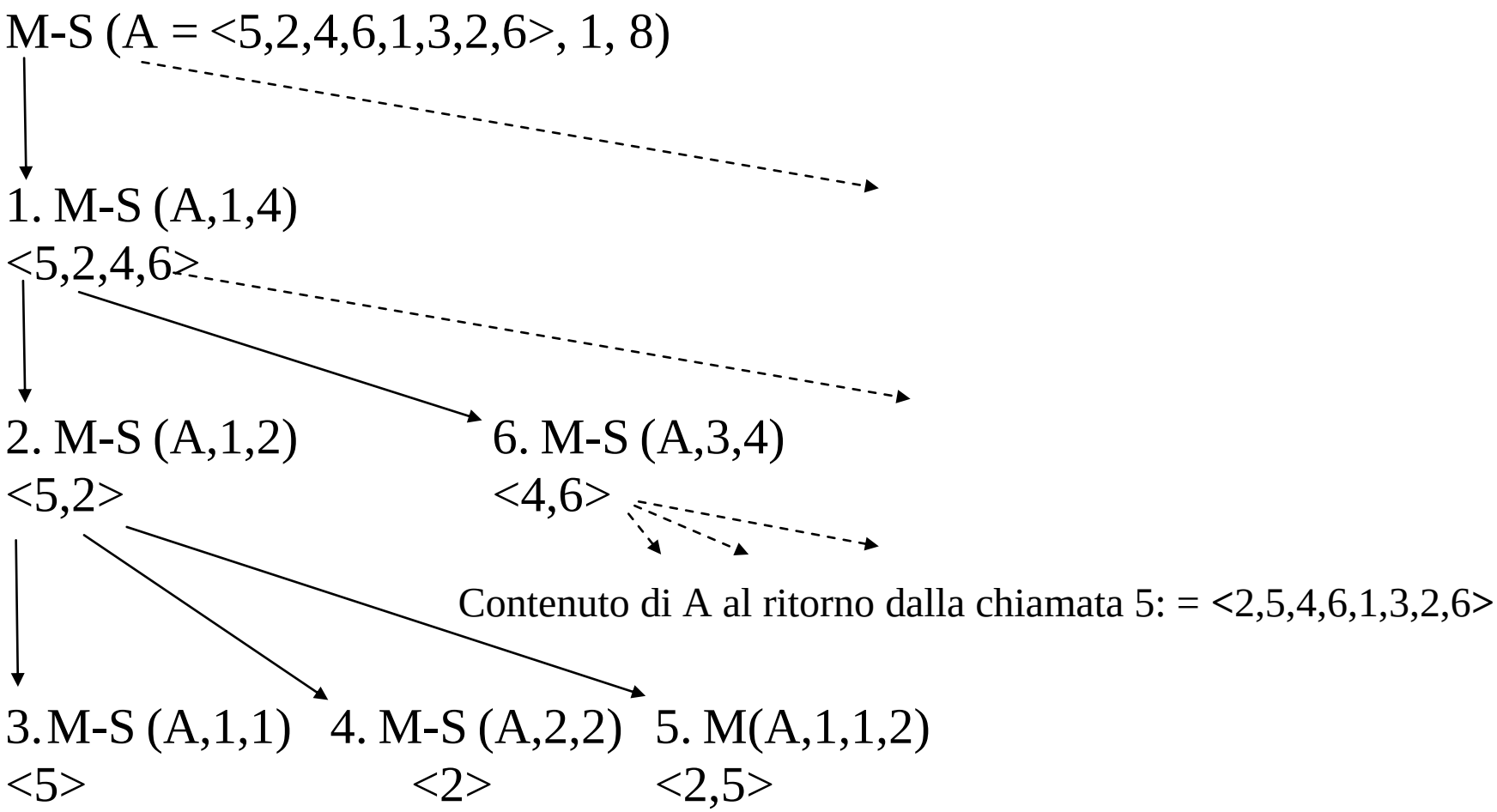
MERGE-SORT (cont.)

```
MERGE-SORT(A,p,r)
1 if p < r
2  ▷A[p .. r] è il sottoarray da ordinare
   se p ≥ r il sottoarry ha al più un elemento ed è perciò già ordinato
3    then   q ← ⌊(p+r)/2⌋
4           MERGE-SORT(A,p,q)
5           MERGE-SORT(A,q+1,r)
6           MERGE(A,p,q,r)
7           ▷ assumendo A[p .. q] e A[q + 1 .. r] ordinati, MERGE genera,
            al di fuori di A, un singolo sottoarray ordinato che
            sostituisce al corrente A[p .. r]
```

Si può facilmente immaginare una procedura MERGE lineare

Per ordinare l'intera sequenza A si invoca MERGE-SORT(A, 1, length[A])

MERGE-SORT: un esempio



Divide-et-impera: analisi

T(n) di un algoritmo che contiene una chiamata ricorsiva di se stesso: equazione di ricorrenza (o ricorrenza)

Per MERGE-SORT

- ♦ Se n = 1, T(n) è costante (cioè $\Theta(1)$)
- ♦ Se n > 1, T(n) è la somma di tre contributi:
- Divide: calcola l'indice di mezzo dell'array, impiegando un tempo costante (cioè $\Theta(1)$)
- Impera: risolve due sottoproblemi, ciascuno di dimensione n/2, per un tempo complessivo pari a 2T(n/2)
- Combina: se si adotta una procedura lineare il tempo impiegato è $\Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1, \\ 2T(n/2) + \Theta(n) + \cancel{\Theta(1)} & \text{se } n > 1 \end{cases}$$

→ risolvendo questa ricorrenza, $T(n) = \Theta(n\lg n)$

dove *lg* è il logaritmo in base 2

Algoritmi come tecnologia

Es. length[A] = 10^6

Calcolatore 1	Calcolatore 2
Velocità = 10^8 istruzioni/sec	Velocità = 10^6 istruzioni/sec
INSERTION-SORT, codificato in linguaggio macchina ($T(n) = 2n^2$)	MERGE-SORT, codificato in linguaggio di alto livello ($T(n) = 50n\lg n$)
Tempo di calcolo $= \frac{2(10^6)^2 \text{ istruzioni}}{10^8 \text{ istruzioni/sec}}$ = 20000 sec = 5,56 ore	Tempo di calcolo $= \frac{50 \cdot 10^6 \cdot \lg 10^6 \text{ istruzioni}}{10^6 \text{ istruzioni/sec}}$ = 1000 sec = 16,67 minuti