10.5 Ordinamento con *quicksort*

Nel Capitolo 5 abbiamo esaminato due metodi di ordinamento: quello ingenuo e *bubblesort*; quest'ultimo è stato successivamente utilizzato in più occasioni. Come ulteriore e più complesso esempio di procedura ricorsiva consideriamo ora il principe degli algoritmi di ordinamento, *quicksort* (letteralmente: "ordinamento veloce"). Con metodi matematici è possibile dimostrare che quicksort è in media il più veloce metodo di ordinamento a uso generale.

Uno degli aspetti più interessanti di quicksort è che esso ordina gli elementi di un vettore seguendo una procedura analoga a quella seguita da una persona per ordinare un insieme di oggetti. Immaginiamo per esempio di dover mettere in ordine su uno scaffale 300 numeri di una rivista di informatica. In genere, magari in maniera non sempre cosciente, si procede in questo modo: preso un primo fascicolo si portano alla sua sinistra tutti i numeri più piccoli di questo e alla sua destra quelli più grandi. Sui due gruppi così ottenuti si procede poi allo stesso modo ottenendo via via insiemi sempre più piccoli e facilmente ordinabili. Alla fine del processo la raccolta risulta ordinata.

Quicksort agisce essenzialmente allo stesso modo: prima crea due grossi blocchi che poi inizia a ordinare costruendone altri sempre più piccoli, per ritrovarsi alla fine una sequenza interamente ordinata.

L'algoritmo di quicksort inizia determinando un ipotetico valore medio del vettore, detto *pivot* ("pernio"), quindi suddivide gli elementi in due parti: quella degli elementi più piccoli e quella degli elementi più grandi del pivot. Non è indispensabile che la suddivisione sia esattamente in parti uguali: l'algoritmo funziona con qualunque approssimazione. Tuttavia, quanto più la suddivisione è esatta, tanto più l'ordinamento risulta veloce.

```
/* Ordinamento quicksort  di un array di int */

#include <stdio.h>

#define N 10 /* numero elementi dell'array */
int v[N];    /* array contenente gli interi immessi */

void quick(int, int);
void scambia(int *, int *);

main()
{
int i;
for(i=0; i<N; i++) {
  printf("\nImmettere un intero n.%d: ",i);
  scanf("%d", &v[i]);
}

quick(0,N-1);       /* Chiamata della procedura quick */

for(i=0; i<N; i++)      /* Sequenza ordinata */
  printf("\n%d", v[i]);
putchar('\n');
}

/* Procedura ricorsiva "quick" */
void quick(int sin, int des)
{
  int i, j, media;
  media= (v[sin]+v[des]) / 2;
  i = sin;
  j = des;

  do {
    while(v[i]<media) i = i+1;
    while(media<v[j]) j = j-1;
    if(i<=j) {
      scambia(&v[i], &v[j]);
      i = i+1;
      j = j-1;
    }
  }
  while (j>=i);

if(sin<j) quick(sin, j);  /* Invocazione ricorsiva */
if(i<des) quick(i, des);  /* Invocazione ricorsiva */
}

void scambia(int *a, int *b)
{
  int temp;
```

```
  temp = *a;
  *a = *b;
  *b = temp;
}
```

Listato 10.5 Ordinamento di una sequenza con il metodo quicksort

Il programma del Listato 10.5 realizza l'ordinamento quicksort su un vettore di 10 elementi interi; nella Figura 10.3 viene illustrata un'applicazione della procedura `quick`, dove la prima colonna contiene la sequenza iniziale. La stima del pivot avviene, in modo non molto raffinato, facendo semplicemente la media tra il primo e l'ultimo elemento della parte del vettore su cui l'algoritmo sta lavorando. Una volta stimato il pivot, la procedura sposta tutti gli elementi di valore minore di questo nella parte bassa del vettore, tutti quelli con valore maggiore nella parte alta. A ogni passo del processo quicksort ordina quindi attorno al pivot gli elementi del blocco del vettore esaminato. Man mano che i blocchi diventano sempre più piccoli il vettore tende a essere completamente ordinato.

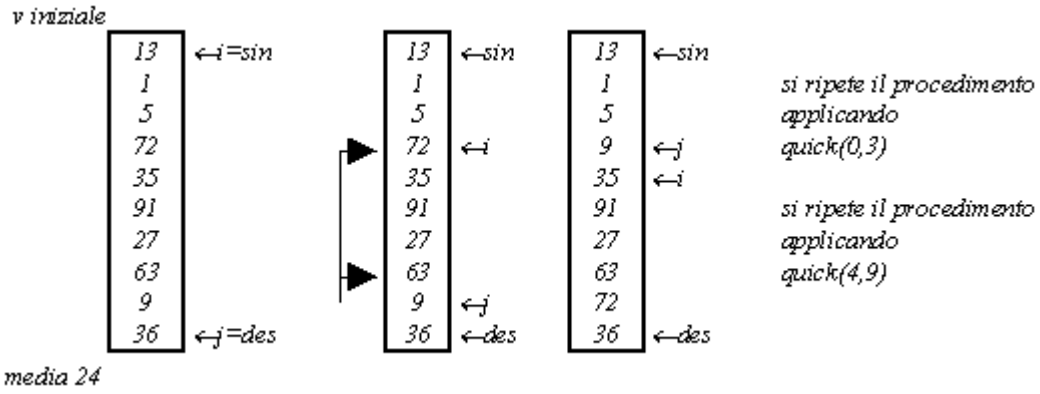

Figura 10.3 Fasi di lavoro di quicksort applicato a un vettore di 10 interi

La procedura `quick` del Listato 10.5 viene inizialmente applicata sull'intero vettore (dall'elemento di indice 0 a quello di indice *N*–1); è calcolato il valore medio `media` tra `vet[sin]` e `vet[des]` che alla prima chiamata è 24, la media appunto tra i due estremi del vettore: 13 (`vet[0]`) e 36 (`vet[9]`) (Figura 10.3). Successivamente `quick` scorre gli elementi del blocco a partire dal basso, fintantoché si mantengono minori della media:

```
while(v[i]<media) i = i+1;
```

Analogamente, la procedura scorre gli elementi del blocco a partire dall'alto, fintantoché si mantengono maggiori della media:

```
while(media<v[j]) j = j-1;
```

A questo punto, se l'indice dell'elemento della parte inferiore del blocco è minore o uguale all'indice dell'elemento della parte superiore, si effettua lo scambio tra i rispettivi valori:

```
if(i<=j) { scambia(&v[i], &v[j]); ...
```

Nel caso in esame viene scambiato il quarto elemento (`i=3`) con il nono (`j=8`), come mostrato nella seconda colonna della Figura 10.3. Successivamente la procedura incrementa gli indici degli elementi considerati e prosegue le scansioni dal basso e dall'alto finché `i` non risulti maggiore di `j`, condizione controllata dal `do-while`. Nell'esempio, dopo la seconda iterazione `i` assume valore 4 e `j` valore 5, così che il ciclo `do-while` ha termine. Quindi la procedura richiama se stessa passando nuovi valori superiori e inferiori e analizzando blocchi sempre più piccoli del vettore:

```
if(sin<j) quick(sin, j);
if(i<des) quick(i, des);
```

Nel caso in esame `sin` vale 0 e `j` vale 3, per cui è chiamata `quick(0,3)` e il blocco che viene ordinato è quello tra `vet[0]`, che vale 13, e `vet[3]`, che vale 9.

Quando raggiunge il livello più basso, ovvero quando i sottoinsiemi trattati sono costituiti da un solo elemento, la ricorsione termina e la procedura restituisce al programma il vettore ordinato.

È importante notare come l'uso di procedure ricorsive non sia, nella scrittura di quicksort come in quella di ogni altro algoritmo, indispensabile. Ciò nonostante, una versione non ricorsiva di quicksort risulta laboriosa e di difficile lettura, così che anche in questo caso, come nel calcolo del fattoriale, si preferisce in genere scrivere il programma nella sua versione ricorsiva.

Nei capitoli precedenti avevamo esaminato il metodo di ordinamento *bubblesort*, adesso abbiamo visto quicksort; il parametro rispetto al quale sono confrontati gli algoritmi di ordinamento è dato dal numero di confronti necessari per ordinare un vettore, poiché questo risulta essere direttamente proporzionale al tempo impiegato dall'algoritmo. Tale numero è espresso come funzione del numero di elementi del vettore. Per esempio, nell'ordinamento di un vettore di *n* numeri casuali, se *n* è pari a 100, 500 o 1000, il numero stimato di confronti in media corrisponderà rispettivamente per bubblesort a 4950, 124.750 e 499.500; con quicksort si avranno 232, 1437 e 3254 confronti. Sottolineiamo che si tratta di *stime*, perché il numero effettivo dei confronti dipende comunque dai particolari valori del vettore da ordinare.

Questa valutazione è stata ottenuta in base a considerazioni matematiche di tipo teorico ma molti programmatori trovano più semplice confrontare i metodi di ordinamento misurando direttamente il tempo impiegato da ognuno di loro per ordinare il medesimo vettore. Anche in questo caso si mantiene la stessa classifica nelle prestazioni; per esempio, con 1000 elementi bubblesort è in media 70 volte più lento di quicksort. Notiamo ovviamente che il tempo assoluto occorrente dipende dall'elaboratore su cui si sta lavorando.

Il vantaggio della ricorsione risiede nella possibilità di realizzare algoritmi sintetici ed eleganti. Inoltre, la scrittura di funzioni ricorsive permette al programmatore di verificare a fondo le proprie conoscenze e possibilmente di migliorarle. Nei capitoli successivi vedremo l'uso della ricorsività nell'ambito delle strutture dati, come liste e alberi, dove a volte la soluzione migliore, per la natura stessa dei problemi che si affronteranno, sarà appunto quella ricorsiva.