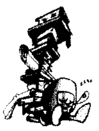
la ricorsione, situazioni che danno luogo a una distribuzione sbilanciata con situazioni che conducono a distribuzioni bilanciate. Tuttavia, tale costo medio dipende dall'ordine iniziale con cui sono presentati gli elementi nell'array da ordinare.

Mostriamo ora un'analisi al caso medio più robusta che risulta essere *indipendente* dall'ordine iniziale degli elementi nell'array e si basa sull'uso della **casualità** per far sì che la distribuzione sbilanciata occorra con una probabilità trascurabile: semplicemente scegliamo `pivot` in modo aleatorio, equiprobabile e uniforme, nella riga 4 del Codice 2.13. Il risultato di tale scelta casuale è che il valore di `perno` restituito nella riga 5 è uniformemente distribuito tra le (equiprobabili) posizioni del segmento a[`sinistra`, `destra`]. Supponiamo pertanto di dividere tale segmento in quattro parti uguali, chiamate **zone**. In base a quale zona contiene la posizione `perno` restituita nella riga 5, otteniamo i seguenti *due* eventi *equiprobabili*:



- la posizione `perno` ricade nella prima o nell'ultima zona: in tal caso, `perno` è detto essere *esterno*;
- la posizione `perno` ricade nella seconda o nella terza zona: in tal caso, `perno` è detto essere *interno*.

Indichiamo con $T(n)$ il *costo medio* dell'algoritmo `QuickSort` eseguito su n dati in ingresso. Osserviamo che la media $\frac{x+y}{2}$ di due valori x e y può essere vista come la loro somma pesata con la rispettiva probabilità $\frac{1}{2}$, ovvero $\frac{1}{2}x + \frac{1}{2}y$, considerando i due valori come equiprobabili. Nella nostra analisi, x e y sono sostituiti da opportuni valori di $T(n)$ corrispondenti ai due eventi equiprobabili sopra introdotti. Più precisamente, quando `perno` è esterno (con probabilità $\frac{1}{2}$), la distribuzione può essere estremamente sbilanciata nella ricorsione e, come abbiamo visto, quest'ultima può richiedere fino a $x = T(n-1) + c'n \leqslant T(n) + O(n)$ tempo, dove il termine $O(n)$ si riferisce al costo della distribuzione effettuata nel Codice 2.14. Quando invece `perno` è interno (con probabilità $\frac{1}{2}$), la distribuzione più sbilanciata possibile nella ricorsione avviene se `perno` corrisponde al minimo della seconda zona oppure al massimo della terza. Ne deriva una distribuzione dei dati che porta alla ricorsione su circa $\frac{n}{4}$ elementi in una chiamata di `QuickSort` e $\frac{3}{4}n$ elementi nell'altra (le altre distribuzioni in questo caso non possono andare peggio perché sono meno sbilanciate). In tal caso, la ricorsione richiede al più $y = T(\frac{n}{4}) + T(\frac{3}{4}n) + O(n)$ tempo. Facendo la media pesata di x e y, otteniamo

$$T(n) \leqslant \frac{1}{2}T(n) + \frac{1}{2}\left[T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right)\right] + c'n \tag{2.6}$$

per un'opportuna costante $c' > 0$. Moltiplicando entrambi i termini nella (2.6) per 2 e risolvendo rispetto a $T(n)$, otteniamo

$$T(n) \leqslant T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + 2c'n \tag{2.7}$$

```
QuickSelect( a
  IF (sinistra
    RETURN a[s
  } ELSE {
    scegli piv
    perno = Di
    IF (r-1 
      QuickSel
    } ELSE {
      QuickSel
    }
  }
```

Codice 2.15 Selezione d

che è simile a un'equazi
appare una disuguaglia

T'(

la cui soluzione dimos
$T(n) \leqslant T'(n)$, ne deri
dalle scelte casuali di pi
algoritmo si chiama cas
sfavorevoli, risultando
array già in ordine cre
celle di memoria aggiu

Come il suo nome
ne usato diffusamente
per fusione MergeSor
del linguaggio C usa un
quando $n \leqslant n_0$ per un
al più n_0 elementi va
namento per inserzion
(risparmiando la maggi
quella del linguaggio C
maniera sbilanciata, vie

```
QuickSelect( a, sinistra, r, destra ):
                                   ⟨pre: 0 ⩽ sinistra ⩽ r − 1 ⩽ destra ⩽ n − 1⟩
  IF (sinistra == destra) {
    RETURN a[sinistra];
  } ELSE {
    scegli pivot nell'intervallo [sinistra...destra];
    perno = Distribuzione( a, sinistra, pivot, destra );
    IF (r-1 == perno) { return a[perno]; } else if (r-1 < perno)
      QuickSelect( a, sinistra, r, perno-1 );
    } ELSE {
      QuickSelect( a, perno+1, r, destra );
    }
  }
```

Codice 2.15 Selezione dell'elemento di rango r per distribuzione in un array a.

che è simile a un'equazione di ricorrenza se non per il fatto che al posto dell'uguaglianza appare una disuguaglianza. Consideriamo allora la seguente equazione di ricorrenza

$$T'(n) = \begin{cases} c & \cancel{\text{se } n = 1} \quad n \leqslant n_0 \\ T'(\frac{n}{4}) + T'(\frac{3}{4}n) + 2c'n & \text{altrimenti} \end{cases} \tag{2.8}$$

la cui soluzione dimostriamo essere $T'(n) = O(n \log n)$ nel Paragrafo 2.5.5. Poiché $T(n) \leqslant T'(n)$, ne deriva che $T(n) = O(n \log n)$ al caso medio e che questo dipende dalle scelte casuali di pivot piuttosto che dalla configurazione dei dati in ingresso: un tale algoritmo si chiama **casuale** o **random** perché impiega la casualità per sfuggire a situazioni sfavorevoli, risultando più robusto rispetto a tali eventi (per esempio, in presenza di un array già in ordine crescente). Inoltre, può essere implementato usando solo $O(\log n)$ celle di memoria aggiuntive in media.

Come il suo nome suggerisce, l'algoritmo di quicksort è molto veloce in pratica e viene usato diffusamente per ordinare i dati in memoria principale (laddove l'ordinamento per fusione `MergeSort` è utile soprattutto in memoria secondaria). La libreria standard del linguaggio C usa un algoritmo di quicksort in cui il caso base della ricorsione si ferma quando $n \leqslant n_0$ per una certa costante $n_0 > 1$: terminata la ricorsione, ogni segmento di al più n_0 elementi va ordinato individualmente, ma basta una singola passata dell'ordinamento per inserzione per ordinare tutti questi segmenti in $O(n \times n_0) = O(n)$ tempo (risparmiando la maggior parte delle chiamate ricorsive). In altre librerie standard, come quella del linguaggio C++, quando l'algoritmo di quicksort inizia a distribuire i dati in maniera sbilanciata, viene sostituito dall'algoritmo di ordinamento per fusione.

Possiamo modificare lo schema ricorsivo del Codice 2.13 per risolvere il problema della **selezione** dell'elemento con rango r in un array a di n elementi distinti, *senza* bisogno di ordinarli (ricordiamo che a contiene r elementi minori o uguali di tale elemento): notiamo che tale problema diventa quello di trovare il minimo in a quando $r = 1$ e il massimo quando $r = n$. Per risolvere il problema per un qualunque valore di r con $1 \leqslant r \leqslant n$, osserviamo che la funzione `Distribuzione` del Codice 2.14 restituisce il rango del pivot px e posiziona tutti gli elementi di rango inferiore alla sua sinistra e tutti quelli di rango superiore alla sua destra. In base a tale osservazione, possiamo modificare il codice di ordinamento per distribuzione considerando che, per risolvere il problema della selezione, è sufficiente proseguire ricorsivamente nel *solo* segmento dell'array contenente l'elemento da selezionare: otteniamo così il Codice 2.15, che determina tale segmento sulla base del confronto tra $r - 1$ e `perno` (righe 8–12). La ricorsione ha termine quando il segmento è composto da un solo elemento, nel qual caso, il codice restituisce tale elemento (notiamo che alcuni elementi dell'array sono stati spostati durante l'esecuzione dell'algoritmo).

L'equazione di ricorrenza per il costo al caso medio è costruita in modo simile all'equazione (2.6), con la differenza che conteggiamo una sola chiamata ricorsiva (la più sbilanciata) ottenendo $T(n) \leqslant \frac{1}{2}T(n) + \frac{1}{2}T(\frac{3}{4}n) + c'n$. Moltiplicando entrambi i termini per 2 e risolvendo rispetto a $T(n)$, otteniamo $T(n) \leqslant T(\frac{3}{4}n) + 2c'n$, a cui associamo un'equazione di ricorrenza in cui $T'(n)$ appare al posto di $T(n)$ e la disuguaglianza diventa un'uguaglianza, come nell'equazione (2.8). Possiamo questa volta applicare il teorema fondamentale delle ricorrenze ponendo $\alpha = 1$, $\beta = \frac{4}{3}$ e $f(n) = 2c'n$ nell'equazione (2.2), per cui rientriamo nel primo caso ($\gamma = \frac{3}{4}$) ottenendo in media una complessità temporale $O(n)$ per l'algoritmo random di selezione per distribuzione (osserviamo che esiste un algoritmo lineare al caso pessimo, ma d'interesse più teorico).

2.5.5 Alternativa al teorema fondamentale delle ricorrenze

L'equazione di ricorrenza (2.8) non è risolvibile con il teorema fondamentale delle ricorrenze, in quanto non è un'istanza dell'equazione (2.2). In generale, quando un'equazione di ricorrenza non ricade nei casi del teorema fondamentale delle ricorrenze, occorre determinare tecniche di risoluzione alternative. Nello specifico dell'equazione (2.8), notiamo che il valore $T'(n)$ (livello 0 della ricorsione) è ottenuto sommando a $2c'n$ i valori restituiti dalle due chiamate ricorsive: quest'ultime, che costituiscono il livello 1 della ricorsione, sono invocate l'una con input $\frac{n}{4}$ e l'altra con input $\frac{3}{4}n$ e, in corrispondenza di tale livello, contribuiscono al valore $T'(n)$ per un totale di $2c'\frac{n}{4} + 2c'\frac{3}{4}n = 2c'n$.

Passando al livello 2 della ricorsione, ciascuna delle chiamate del livello 1 ne genera altre due, per un totale di quattro chiamate, rispettivamente con input $\frac{n}{4^2}$, $\frac{3}{4^2}n$, $\frac{3}{4^2}n$ e $\frac{3^2}{4^2}n$, che contribuiscono al valore $T'(n)$ per un totale di $2c'\frac{n}{4^2} + 2c'\frac{3}{4^2}n + 2c'\frac{3}{4^2}n +$