Relazioni di ricorrenza e algoritmi di ordinamento di complessità $n \log n$

March 26, 2020

Obiettivi: studiare la complessità degli algoritmi ricorsivi e analizzare gli algoritmi di ordinamento di complessità $n \log n$.
Argomenti: relazioni di ricorrenza (a partizione costante e a partizione bilanciata) e relative teoremi, metodo iterativo, merge-sort, quick-sort.

Il numero di operazioni eseguite (il tempo di esecuzione) di un **algoritmo ricorsivo** è spesso semplice da esprimere con una **relazione di ricorrenza**.

Si possono distinguere due casi:

- la chiamata ricorsiva **diminuisce la dimensione del input** (oppure l'input stesso) **di un costante**: in questo caso si ottiene una **relazione di ricorrenza a partizione costante**;
- la chiamata ricorsiva **diminuisce la dimensione del input** (oppure l'input stesso) **di un certo fattore** (per esempio, dimezza l'input): in questo caso si ottiene una **relazione di ricorrenza a partizione bilanciata**.

1 Relazione di ricorrenza a partizione costante

Analizziamo l'algoritmo ricorsivo che calcola il fattoriale:

1: FATTORIALE(n)
2: **if** $n < 2$ **then return** 1
3: **return** n·FATTORIALE($n-1$)

Considerando ogni riga come singola operazione, si può contare il numero di operazioni $T(n)$ col seguente ragionamento:

- se $n < 2$ allora viene eseguita solo la riga 2;
- se $n \geq 2$ allora viene eseguita la riga 2 e la chiamata FATTORIALE($n-1$) della riga 3 dà luogo a $T(n-1)$ operazioni in più.

Se ne ottiene la seguente relazione di ricorrenza:

$$T(n) = \begin{cases} 1 & n < 2 \\ 1 + T(n-1) & n \geq 2 \end{cases}$$

Metodo dell'iterazione: un approccio per trovare la soluzione di una relazione di ricorrenza consiste nello sviluppare la relazione di ricorrenza, fino ad intuirne la soluzione.

Risolviamo col metodo dell'iterazione una relazione di ricorrenza leggermente diversa (e più generale) di quella precedente:

$$T(n) = \begin{cases} c & n = 0 \\ T(n-1) + d & n \geq 1 \end{cases}$$

Applicando la ricorrenza più volte:

$$\begin{aligned} T(n) &= T(n-1) + d = \\ &= T(n-2) + d + d = \\ &= T(n-3) + d + d + d = \\ \text{dopo } k \leq n \text{ iterazioni:} & \\ &= T(n-k) + kd = \\ \text{dopo } n \text{ iterazioni:} & \\ &= T(0) + nd = \\ &= c + nd \end{aligned}$$

Di conseguenza la soluzione è

$$T(n) = c + nd \quad \text{e quindi} \quad T(n) \in \Theta(n).$$

Metodo della sostituzione: consiste nell'ipotizzare una soluzione e applicare il principio di induzione per verificare la soluzione ipotizzata.

Verifichiamo la soluzione ($T(n) = c + nd$) ottenuta precedentemente col principio di induzione.
Caso base. Con $n = 0$ la soluzione funziona: $c + 0 \cdot d = c = T(0)$.
Passo induttivo. Dobbiamo dimostrare che $T(n) = c+nd$ (l'ipotesi induttiva) implica $T(n+1) = c+(n+1)d$:

$$\begin{aligned} \text{secondo la relazione di ricorrenza:} & \\ T(n+1) &= T(n) + d \\ \text{utilizzando l'ipotesi induttiva:} & \\ &= c + nd + d \\ &= c + (n+1)d \end{aligned}$$

Quindi la soluzione è corretta.

Consideriamo ora l'algoritmo che risolve il problema delle torre di Hanoi (riportato in forma leggermente diversa rispetto a com'era prima).

```
1: MoveTower(n, A, B, C)
2: if n ≥ 1 then
3:     MoveTower(n − 1, A, C, B)
4:     move 1 disk from A to C
5:     MoveTower(n − 1, B, A, C)
```

La seguente relazione di ricorrenza descrive il numero di operazioni (con $b = 1, c = 2$ e $d = 2$):

$$T(n) = \begin{cases} b & n = 0 \\ cT(n-1) + d & n \geq 1 \end{cases}$$

dove assumiamo $c > 1$ (altrimenti la ricorrenza ha la stessa forma di quella precedente).

Analizziamo la relazione di ricorrenza col metodo dell'iterazione:

$$\begin{aligned} T(n) &= cT(n-1)+d \\ &= c(cT(n-2)+d)+d = c^2T(n-2)+cd+d = \\ &= c^2(cT(n-3)+d)+cd+d = c^3T(n-3)+c^2d+cd+d = \end{aligned}$$

dopo $k \leq n$ iterazioni:

$$= c^kT(n-k)+(c^{k-1}+c^{k-2}+\cdots+c+1)d$$

dopo n iterazioni:

$$= c^nT(0)+(c^{n-1}+c^{n-2}+\cdots+c+1)d$$

con $c > 1$:

$$= c^nb+\frac{c^n-1}{c-1}d$$

Di conseguenza la soluzione è

$$T(n) = c^nb+\frac{c^n-1}{c-1}d$$

Verifichiamo la soluzione col principio di induzione.
Caso base. Con $n = 0$ la soluzione funziona: $c^0b+\frac{c^0-1}{c-1}d = b = T(0)$.
Passo induttivo. Dobbiamo dimostrare che $T(n) = c^nb+\frac{c^n-1}{c-1}d$ (l'ipotesi induttiva) implica $T(n+1) = c^{n+1}b+\frac{c^{n+1}-1}{c-1}d$:

secondo la relazione di ricorrenza:

$$T(n+1) = cT(n)+d$$

utilizzando l'ipotesi induttiva:

$$\begin{aligned} &= c\left(c^nb+\frac{c^n-1}{c-1}d\right)+d \\ &= c^{n+1}b+\frac{c^{n+1}-c}{c-1}d+d \\ &= c^{n+1}b+\frac{c^{n+1}-c+c-1}{c-1}d \\ &= c^{n+1}b+\frac{c^{n+1}-1}{c-1}d \end{aligned}$$

Quindi la soluzione è corretta.

La funzione $T(n) \in \Theta(c^n)$ (per verificarlo è sufficiente considerare $\lim_{n\to\infty}\frac{T(n)}{c^n}$).

1.1 Quick-sort

Quick-sort. L'idea dell'algoritmo: si seleziona un elemento del vettore, detto perno, attorno al quale si riorganizzano gli altri elementi in modo tale che gli elementi più piccoli o uguali al perno si trovino in posizioni precedenti a quella del perno e gli elementi più grandi in posizioni successive; poi si ripete lo stesso procedimento a sinistra e a destra del perno.

```
QUICK-SORT(A[1..n])
if n > 1 then
    p ← PARTITION(A[1..n])        ▷ partizionamento porta il perno nella posizione p
    if p > 2 then        ▷ se prima del perno ci sono almeno 2 elementi
        QUICK-SORT(A[1..p − 1])
    end if
    if p < n − 1 then        ▷ se dopo il perno ci sono almeno 2 elementi
        QUICK-SORT(A[p + 1..n])
    end if
end if

PARTITION(A[1..n])
i ← 2, j ← n
while i ≤ j do
    if A[i] ≤ A[1] then
        i ← i + 1
    else
        if A[j] > A[1] then
            j ← j − 1
        else
            scambia A[i] con A[j]
            i ← i + 1, j ← j − 1
        end if
    end if
end while
scambia A[1] con A[j]
return j
```

Il partizionamento effettuato da PARTITION utilizza $A[1]$ come perno e considera una volta tutti gli elementi del sottovettore che deve riorganizzare. Di conseguenza il numero di operazioni effettuate da PARTITION può essere considerato come

$$T_P(n) = an$$

con una costante a.

Il caso peggiore è quello nel quale il perno capita essere sempre o il minimo o il massimo del sottovettore riorganizzato da PARTITION. In questo caso una sola chiamata ricorsiva viene effettuata. Nel caso peggiore la relazione di ricorrenza per QUICK-SORT ha la seguente forma:

$$T(n) = \begin{cases} c & n = 1 \\ T(n-1) + T_P(n) + b & n > 1 \end{cases} = \begin{cases} c & n = 1 \\ T(n-1) + an + b & n > 1 \end{cases}$$

con tre costanti a, b e c.

Analizziamo la relazione di ricorrenza col metodo dell'iterazione:

$$\begin{aligned} T(n) &= T(n-1) + an + b \\ &= T(n-2) + a(n-1) + b + an + b = \\ &= T(n-3) + a(n-2) + b + a(n-1) + b + an + b = \end{aligned}$$

dopo $k \leq n$ iterazioni:

$$= T(n-k) + a\sum_{i=0}^{k-1}(n-i) + kb$$

dopo $n-1$ iterazioni:

$$\begin{aligned} &= T(1) + a\sum_{i=0}^{n-2}(n-i) + (n-1)b \\ &= c + a\sum_{i=0}^{n-2}(n-i) + (n-1)b = c + a\sum_{i=2}^{n} i + (n-1)b \end{aligned}$$

da dove si evince che $T(n) \in \Theta(n^2)$ e quindi **Quick-Sort è quadratico nel caso peggiore**.

Il seguente teorema fornisce una "ricetta" per trovare un limite superiore in senso O data una relazione di ricorrenza di ordine costante.

Teorema: Siano $a_1, ..., a_h$ costanti intere non negative, c, d e β costanti reali positive tale che

$$T(n) = \begin{cases} d & \text{se } n \leq m \leq h \\ \sum_{i=1}^{h} a_i T(n-i) + cn^\beta & \text{se } n > m \end{cases}$$

Posto $a = \sum_{i=1}^{h} a_i$

$$T(n) \in O(n^{\beta+1}) \text{ se } a = 1$$
$$T(n) \in O(n^\beta a^n) \text{ se } a \geq 2$$

(Il teorema precedente opera con O perché in certi casi la relazione di ricorrenza permette un confine superiore più stretto di quello indicato dal teorema.)

Il teorema si può applicare ai esempi discussi precedentemente (fattoriale, Hanoi, caso peggiore di Quick-Sort).

2 Relazione di ricorrenza a partizioni bilanciate

Un approccio generale per la soluzione di problemi computazionali è il *il divide et impera*. L'approccio consiste nel dividere ricorsivamente un problema in due o più sotto-problemi sino a che i sotto-problemi diventino di semplice risoluzione, quindi, si combinano le soluzioni al fine di ottenere la soluzione del problema dato.

L'approccio generale può essere scritto così in pseudo-codice:

```
DeI(P, n)        ▷ n è la dimensione del problema P
if n ≤ k then
    r ← soluzione diretta del problema
    return r
else
    dividi il problema in sotto-problemi P_i, ..., P_h di dimensione n_1, ..., n_h
    for i ← 1 to h do
        r_i ← DeI(P_i, n_i)
    end for
    return combinazione di r_1, ..., r_h
```

end if

Per esempio, la ricerca binaria segue lo schema precedente:

```
BinSearch-Ric(x, A, i, j)
    ▷ Pre: A[i..j] ordinato
    ▷ Post: true se x ∈ A[i..j]
if i > j then
    return false
else
    m ← ⌊(i + j)/2⌋
    if x = A[m] then
        return true
    else
        if x < A[m] then
            return BinSearch-Ric(x, A, i, m − 1)
        else      ▷ A[m] < x
            return BinSearch-Ric(x, A, m + 1, j)
        end if
    end if
end if
```

Nel caso peggiore (quando l'elemento cercato non è presente nel vettore), il tempo di esecuzione può essere espresso con la relazione

$$T(n) = \begin{cases} c & \text{se } n = 1 \\ T\left(\frac{n}{2}\right) + d & \text{se } n > 1 \end{cases}$$

dove n è il numero di elementi del vettore ($n = j - i + 1$).

Con il metodo dell'iterazione, assumendo che n è potenza di 2:

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + d \\ &= T\left(\frac{n}{4}\right) + d + d \\ \text{dopo } k \leq \log_2 n \text{ iterazioni:} & \\ &= T\left(\frac{n}{2^k}\right) + kd \\ \text{dopo } k = \log_2 n \text{ iterazioni:} & \\ &= T(1) + d\log_2 n \\ &= c + d\log_2 n \end{aligned}$$

e quindi

$$T(n) \in \Theta(\log n)$$

Merge-sort. Ordinamento per fusione. L'idea:

- un vettore che contiene un elemento è ordinato;
- se il vettore contiene più di un elemento allora viene diviso in due parti bilanciate;
- ognuna di queste due parti viene ordinata applicando ricorsivamente l'algoritmo;
- le due parti vengono fuse.

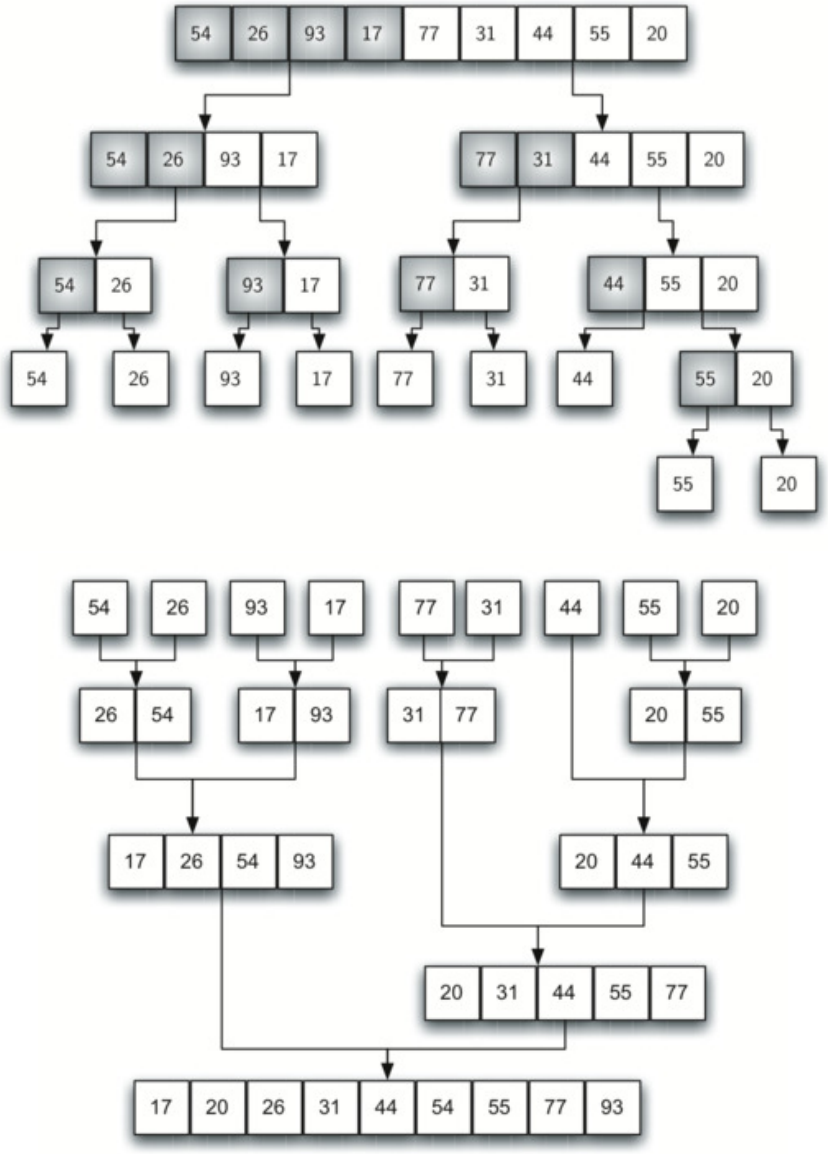
In pseudo-codice:

```
Merge-Sort(A[1..n])
if n > 1 then
    m ← ⌊n/2⌋
```

MERGE-SORT($A[1..m]$)
MERGE-SORT($A[m+1..n]$)
MERGE($A, 1, m, n$)
end if

dove MERGE$(1, m, n)$ effettua la fusione di $A[1..m]$ e $A[m+1..n]$ posto che essi sono ordinati.

L'algoritmo divide prima il vettore in sottovettori che contengono un solo elemento e poi tramite una sequenza di fusioni ordina tutto il vettore:



La fusione si può effettuare in maniera iterativa:

```
MERGE(A, f, m, l)
i, j, k ← f, m + 1, 1
while i ≤ m ∧ j ≤ l do
    if A[i] ≤ A[j] then
        B[k] ← A[i]
        i ← i + 1
    else
        B[k] ← A[j]
        j ← j + 1
    end if
    k ← k + 1
end while
if i ≤ m then
    B[k..k + m − i] ← A[i..m]
else
    B[k..k + l − j] ← A[j..l]
end if
A[f, l] ← B[1..l − f + 1]
```

dove f sta per *first*, m per *middle* e l per *last*. Come precondizione si richiede che $1 \leq f \leq m \leq l \leq n$ (dove n è il numero di elementi in A) e che i sottovettori $A[f, m]$ e $A[m + 1, l]$ siano ordinati.

(La "fusione" può essere effettuata anche in maniera ricorsiva (l'algoritmo prende in input due vettori ordinati da fondere):

```
MERGE(B[1..n_B], C[1..n_C])
if n_B = 0 then
    return C
else
    if n_C = 0 then
        return B
    else
        if B[1] ≤ C[1] then
            return [B[1], MERGE(B[2..n_B], C)]
        else
            return [C[1], MERGE(B, C[2..n_C])]
        end if
    end if
end if
```

)

Analisi della complessità del MERGE-SORT. La fusione ha complessità temporale $\Theta(n)$. Di conseguenza, la complessità di MERGE-SORT può essere descritta con la relazione di ricorrenza

$$T(n) = \begin{cases} a & \text{se } n = 1 \\ 2T\left(\frac{n}{2}\right) + bn + c & \text{altrimenti} \end{cases}$$

Con il metodo dell'iterazione:

$$\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + bn + c \\
&= 2\left(2T\left(\frac{n}{4}\right) + b\frac{n}{2} + c\right) + bn + c \\
&= 4T\left(\frac{n}{4}\right) + 2bn + 2c + c \\
&= 4\left(2T\left(\frac{n}{8}\right) + b\frac{n}{4} + c\right) + 2bn + 2c + c \\
&= 8T\left(\frac{n}{8}\right) + 3bn + 4c + 2c + c \\
\text{dopo } k \leq \log_2 n \text{ iterazioni :} & \\
&= 2^k T\left(\frac{n}{2^k}\right) + kbn + 2^{k-1}c + 2^{k-2}c + \cdots + c \\
\text{dopo } k = \log_2 n \text{ iterazioni :} & \\
&= nT(1) + \log_2 n \cdot bn + \frac{n}{2}c + \frac{n}{4}c + \cdots + c \\
&= na + \log_2 n \cdot bn + (n-1)c
\end{aligned}$$

dove il termina dominante è $bn \log_2 n$ e quindi **la complessità temporale del MERGE-SORT è** $T(n) \in \Theta(n \log n)$.

Per quanto riguarda QUICK-SORT, nel caso migliore, cioè quando il partizionamento divide in due parti uguali il vettore, la complessità è uguale a quella di MERGE-SORT, quindi $\Theta(n \log n)$.

Per analizzare il caso medio del QUICK-SORT, assumiamo che il partizionamento risulta in due parti che possono essere sbilanciate in qualunque modo con la stessa probabilità. I casi per quanto riguarda la dimensione delle due parti sono $(0, n-1), (1, n-2), (2, n-3), \ldots, (n-2, 1), (n-1, 0)$ e ognuno ha probabilità $1/n$.

Considerando tutti i casi con probabilità $1/n$ la relazione di ricorrenza diventa:

$$T(n) = \begin{cases} a & \text{se } n \leq 1 \\ \frac{1}{n}\sum_{k=1}^{n}(T(k-1) + T(n-k)) + bn + c & \text{altrimenti} \end{cases}$$

La ricorrenza può essere riscritta è manipolata

$$\begin{aligned}
T(n) &= \frac{2}{n}\sum_{i=0}^{n-1} T(i) + bn + c \\
nT(n) &= 2\sum_{i=0}^{n-1} T(i) + bn^2 + cn
\end{aligned}$$

Sottraendo

$$(n-1)T(n-1) = 2\sum_{i=0}^{n-2} T(i) + b(n-1)^2 + c(n-1)$$

si ottiene

$$\begin{aligned}
nT(n) - (n-1)T(n-1) &= \left(2\sum_{i=0}^{n-1} T(i) + bn^2 + cn\right) - \left(2\sum_{i=0}^{n-2} T(i) + b(n-1)^2 + c(n-1)\right) \\
nT(n) - (n-1)T(n-1) &= 2T(n-1) + c + b(2n-1) \\
nT(n) &= (n+1)T(n-1) + c + b(2n-1)
\end{aligned}$$

dividendo a $n(n+1)$

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{c+b(2n-1)}{n(n+1)}$$

Introducendo $D(n) = T(n)/(n+1)$ abbiamo

$$D(n) = D(n-1) + \frac{c+b(2n-1)}{n(n+1)}$$

dove $\frac{c+b(2n-1)}{n(n+1)} \in O(1/n)$. Di conseguenza

$$D(n) \in O\left(\sum_{i=1}^{n} \frac{1}{i}\right) \in O(\log n)$$

e quindi

$$T(n) \in O(n \log n)$$

e quindi anche **nel caso medio QUICK-SORT è ottimale**.

Il seguente teorema, chiamato **Teorema Master**, fornisce una "ricetta" per trovare un limite superiore in senso O data una relazione di ricorrenza a partizioni bilanciate.

Teorema: Siano $a \geq 1, b \geq 2, c > 0, d, \beta \geq 0$ costanti intere non negative tale che

$$T(n) = \begin{cases} d & \text{se } n = 1 \\ aT(n/b) + cn^{\beta} & \text{se } n > 1 \end{cases}$$

Posto $\alpha = \log a / \log b$

$$T(n) \in O(n^{\alpha}) \text{ se } \alpha > \beta$$
$$T(n) \in O(n^{\alpha} \log n) \text{ se } \alpha = \beta$$
$$T(n) \in O(n^{\beta}) \text{ se } \alpha < \beta$$