Merge-Sort con “coprocessore speciale”

Moreno Marzolla

13 marzo 2024

In questo documento risolviamo il problema seguente: supponiamo che esista un “coprocessore speciale” in grado di realizzare l'operazione Merge di Merge-Sort in tempo $\Theta(\sqrt{n})$ anziché $\Theta(n)$ come la procedura Merge vista a lezione. Determinare il costo asintotico di Merge-Sort se si usa tale coprocessore. (questo esercizio è tratto da Camil Demetrescu, Irene Finocchi, Giuseppe F. Italiano, *Algoritmi e strutture dati 2/ed*, McGraw-Hill, 2008, ISBN: 978 88 386 64687).

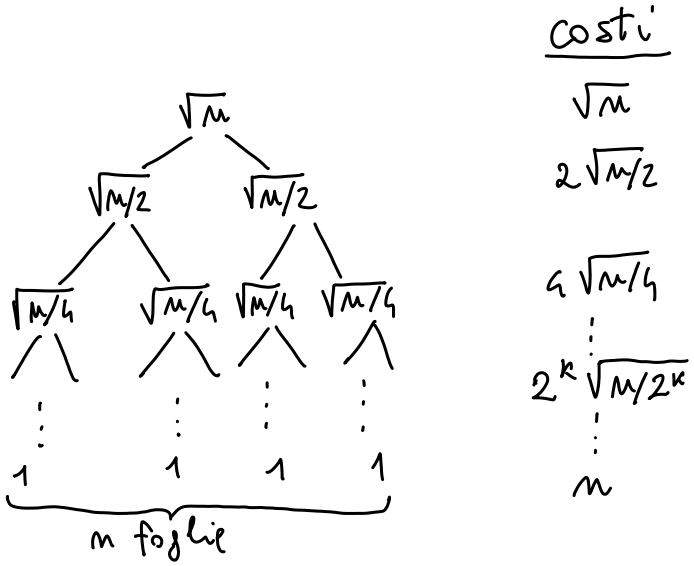
Detto $T(n)$ il numero di operazioni elementari svolte da Merge-Sort per ordinare un array di lunghezza n, possiamo definire $T(n)$ ricorsivamente come segue:

$$T(n) = \begin{cases} c_1 & \text{se } n \leq 1 \\ 2T(n/2) + c_2\sqrt{n} & \text{se } n > 1 \end{cases}$$

L'applicazione del Master Theorem consentirebbe di derivare immediatamente la soluzione. Supponiamo però di non conoscere (o di non volere usare) il Master Theorem, e procediamo con una dimostrazione diretta.

Disegniamo l'albero della ricorsione, segnando in ciascun nodo il costo dell'operazione Merge effettuata in quel nodo. Ricordiamo infatti che la tecnica dell'albero della ricorsione richiede di indicare in ciascun nodo il costo delle operazioni eseguite dall'algoritmo *escludendo* le chiamate ricorsive. Il costo complessivo dell'intero algoritmo (cioè il valore di $T(n)$) è la somma dei costi scritti nei singoli nodi.

Nel nostro caso, l'albero della ricorsione è un albero binario completo, perché ad ogni passo si effettuano due chiamate ricorsive che operano su due sottovettori la cui dimensione è metà di quella corrente:



Nella colonna di destra ("costi") indichiamo la somma dei costi di tutti i nodi di quel livello. Per semplicità omettiamo le costanti c_1, c_2 perché non influiscono sull'ordine di grandezza del risultato. Osservando che l'albero ha circa $\log_2 n$ livelli, si ha che $T(n)$ può essere espresso come:

$$T(n) = \sum_{k=0}^{\log_2 n} 2^k \sqrt{\frac{n}{2^k}}$$

Dato che il fattore $\sqrt{n}$ nella sommatoria non dipende dall'indice k, lo possiamo trattare come se fosse una costante e portare fuori, ottenendo:

$$T(n) = \sqrt{n} \sum_{k=0}^{\log_2 n} \frac{2^k}{\sqrt{2^k}}$$

Ricordando alcune proprietà degli esponenziali:

$$\frac{a^x}{a^y} = a^{x-y}$$
$$(a^x)^y = (a^y)^x = a^{xy}$$
$$\sqrt{2^k} = 2^{k/2}$$

possiamo concludere che $\frac{2^k}{\sqrt{2^k}} = 2^{k/2} = (\sqrt{2})^k$. La sommatoria può essere riscritta come

$$T(n) = \sqrt{n} \sum_{k=0}^{\log_2 n} (\sqrt{2})^k$$

che assume la forma di una serie geometrica di ragione $\sqrt{2}$, la quale ammette una forma chiusa che nel caso generale si esprime come:

$$\sum_{k=0}^{N} a^k = \frac{a^{N+1} - 1}{a - 1}$$

Nel nostro caso si ha $a = \sqrt{2}$, $N = \log_2 n$ per cui

$$T(n) = \sqrt{n}\ \frac{(\sqrt{2})^{\log_2 n+1} - 1}{\sqrt{2} - 1}$$

Osservando la frazione, si ha che il denominatore è una costante e il numeratore può essere riscritto come $(\sqrt{2})^{\log_2 n+1} - 1 = \sqrt{2} \times (\sqrt{2})^{\log_2 n} - 1$. Dato che siamo interessati all'ordine di grandezza di $T(n)$, possiamo trascurare le costanti additive e moltiplicative, ottenendo che l'ordine di grandezza della frazione è dominato da $(\sqrt{2})^{\log_2 n}$. Possiamo quindi scrivere

$$\begin{aligned} T(n) &\propto \sqrt{n} \times (\sqrt{2})^{\log_2 n} \\ &= \sqrt{n} \times \sqrt{2^{\log_2 n}} \\ &= \sqrt{n} \times \sqrt{n} = n \end{aligned}$$

($\propto$ si legge "è proporzionale a"). Possiamo quindi concludere che

$$T(n) = \Theta(n)$$

che è il risultato che avremmo ottenuto applicando il Master Theorem. $\square$