

# Analisi – Ordinamento funzioni

Ordinare le seguenti funzioni in accordo alla loro complessità asintotica. Si scriva  $f(n) < g(n)$  se  $O(f(n)) \subset O(g(n))$ . Si scriva  $f(n) = g(n)$  se  $O(f(n)) = O(g(n))$ , ovvero se  $f(n) = \Theta(g(n))$ .

$$f_1(n) = 2^{n+2}$$

$$f_2(n) = \log^2 n$$

$$f_3(n) = (\log_n(\sqrt{n})^2 n) + 1/n^2$$

$$f_4(n) = 3n^{0.5}$$

$$f_5(n) = 16^{n/4}$$

$$f_6(n) = 2\sqrt{n} + 4n^{1/4} + 8n^{1/8} + 16n^{1/16}$$

$$f_7(n) = \sqrt{(\log n)(\log n)}$$

$$f_8(n) = \frac{n^3}{(n+1)(n+3)}$$

$$f_9(n) = 2^n$$

## Ricorrenza $2T(n/8) + 2T(n/4) + n$

Trovare un limite asintotico superiore e un limite asintotico inferiore alla seguente ricorrenza, facendo uso del metodo di sostituzione:

$$T(n) = \begin{cases} 1 & n \leq 1 \\ 2T(n/8) + 2T(n/4) + n & n > 1 \end{cases}$$

## Analisi – MergeSortK

Si supponga di scrivere una variante di MergeSort, chiamata MergeSortK che, invece di suddividere l'array da ordinare in 2 parti (e ordinarle separatamente), lo suddivide in K parti, le ordina ognuna riapplicando MergeSortK, e le riunifica usando un'opportuna variante MergeK di Merge (la quale, naturalmente, fa la fusione su K sottoarray invece di 2). Come cambia, se cambia, la complessità temporale di MergeSortK rispetto a quella di MergeSort?

# Esercizi di programmazione

## Cerca la coppia - Versione 1

Dato un vettore  $A[1 \dots n]$  di interi e un intero  $v$ , scrivere un algoritmo che determini se esistono due elementi in  $A$  la cui somma sia esattamente  $v$ .

## Cerca la coppia - Versione 2

Dato un vettore  $A[1 \dots n]$  di interi, scrivere un algoritmo che determini se esistono due elementi in  $A$  la cui somma sia esattamente 17.

## Cerca la coppia - Versione 3

Dato un vettore  $A[1 \dots n]$  di interi positivi, scrivere un algoritmo che determini se esistono due elementi in  $A$  la cui somma sia esattamente 17.

Spoiler alert!

# Analisi – Ordinamento funzioni

Le funzioni da ordinare:

$$f_1(n) = 2^{n+2} = 4 \cdot 2^n = \Theta(2^n)$$

$$f_2(n) = \log^2 n = \Theta(\log^2 n)$$

$$f_3(n) = (\log_n(\sqrt{n})^2 n) + 1/n^2 = (\log_n n^2) + 1/n^2 = 2 + 1/n^2 = \Theta(1)$$

$$f_4(n) = 3n^{0.5} = \Theta(n^{1/2})$$

$$f_5(n) = 16^{n/4} = (2^4)^{n/4} = 2^{4n/4} = \Theta(2^n)$$

$$f_6(n) = 2\sqrt{n} + 4n^{1/4} + 8n^{1/8} + 16n^{1/16} = \Theta(n^{1/2})$$

$$f_7(n) = \sqrt{(\log n)(\log n)} = \Theta(\log n)$$

$$f_8(n) = \frac{n^3}{(n+1)(n+3)} = \Theta(n)$$

$$f_9(n) = 2^n = \Theta(2^n)$$

Una volta stabilito l'ordine  $\Theta$  delle funzioni, è abbastanza semplice stabilire l'ordine corretto:

$$f_3 < f_7 < f_2 < f_4 = f_6 < f_8 < f_1 = f_5 = f_9$$

## Ricorrenza $2T(n/8) + 2T(n/4) + n$

$$T(n) = \begin{cases} 1 & n \leq 1 \\ 2T(n/8) + 2T(n/4) + n & n > 1 \end{cases}$$

È facile dimostrare che ovviamente, la funzione  $T(n)$  è  $\Omega(n)$ ; infatti,

$$T(n) = 2T(n/8) + 2T(n/4) + n \geq n \geq c_1 n$$

per  $c_1 \leq 1$ . Non è necessario dimostrare il caso base, perché abbiamo rimosso gli elementi ricorsivi e quindi non abbiamo bisogno di induzione.

## Ricorrenza $2T(n/8) + 2T(n/4) + n$

$$T(n) = \begin{cases} 1 & n \leq 1 \\ 2T(n/8) + 2T(n/4) + n & n > 1 \end{cases}$$

Una prima osservazione che si potrebbe fare per “indovinare” un limite superiore è la seguente:

$$\begin{aligned} T(N) &= 2T(n/8) + 2T(n/4) + n \\ &\leq 2T(n/4) + 2T(n/4) + n \\ &\leq 4T(n/4) + n \end{aligned}$$

In base a questo risultato, il master theorem dice che  $T(n) = O(n \log n)$ .



## Ricorrenza $2T(n/8) + 2T(n/4) + n$

$$T(n) = \begin{cases} 1 & n \leq 1 \\ 2T(n/8) + 2T(n/4) + n & n > 1 \end{cases}$$

A questo punto, non ci resta che vedere quale dei due risultati ( $\Omega(n)$  e  $O(n \log n)$ ) è stretto. Proviamo con  $O(n)$ .

Ipotesi induttiva:  $\forall n < n' : T(n) \leq cn$ . Proviamo che il risultato è valido anche per  $n$ .

$$\begin{aligned} T(n) &= 2T(n/8) + 2T(n/4) + n \\ &\leq 2cn/8 + 2cn/4 + n \\ &= 3/4cn + n \\ &\leq cn \end{aligned}$$

L'ultima disequazione è vera se  $3/4c + 1 \leq c$ , ovvero se  $c \geq 4$ .

## Analisi – MergeSortK

La funzione di ricorrenza per MergeSortK è la seguente:

$$T(n) = \begin{cases} k(T(n/k)) + O(kn) & n > 1 \\ 1 & n = 1 \end{cases}$$

Un modo per implementare MergeK consiste nel trovare il minimo dei  $k$  valori, presenti, operazione che ha complessità  $O(k)$ . Ripetendo l'operazione  $n$  volte, la complessità di MergeK è pari a  $O(kn)$ .

Calcolando  $\alpha = \log_k k = 1$  e confrontandolo con  $n^1$ , è possibile vedere che il costo di MergeSortK è  $O(kn \log n)$ . Per valori costanti di  $k$ , questo corrisponde a  $O(n \log n)$ .