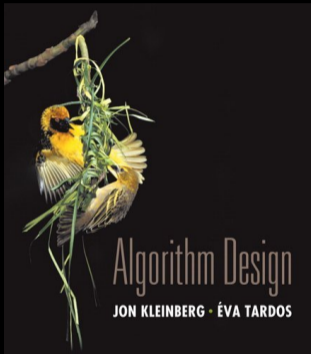


Chapter 2

Basics of Algorithm Analysis



PEARSON
Addison
Wesley

2.1 Computational Tractability

Efficienza degli algoritmi

Siamo interessati

- soprattutto running time
- ma anche spazio, ovvero memoria

Efficienza degli algoritmi

Siamo interessati

- soprattutto running time
- ma anche spazio, ovvero memoria

Come misuriamo l'efficienza di un algoritmo?

Una prima definizione. Un algoritmo è efficiente se, quando implementato, è veloce su istanze di input reali.

Efficienza degli algoritmi

Siamo interessati

- soprattutto running time
- ma anche spazio, ovvero memoria

Come misuriamo l'efficienza di un algoritmo?

Una prima definizione. Un algoritmo è efficiente se, quando implementato, è veloce su istanze di input reali.

Prima impressione. Come non essere d'accordo!

Efficienza degli algoritmi

Siamo interessati

- ❑ soprattutto running time
- ❑ ma anche spazio, ovvero memoria

Come misuriamo l'efficienza di un algoritmo?

Una prima definizione. Un algoritmo è efficiente se, quando implementato, è veloce su istanze di input reali.

Prima impressione. Come non essere d'accordo!

Riflettendo meglio manca qualcosa:

Implementato dove?

Implementato come?

Cos'è una istanza di input reale?

Se l'input è "piccolo"?

Come scala al crescere della grandezza dell'input?

Efficienza degli algoritmi

Siamo interessati

- ❑ soprattutto running time
- ❑ ma anche spazio, ovvero memoria

Come misuriamo l'efficienza di un algoritmo?

Una prima definizione. Un algoritmo è efficiente se, quando implementato, è veloce su istanze di input reali.

Prima impressione. Come non essere d'accordo!

Riflettendo meglio manca qualcosa:

Implementato dove?

Implementato come?

Cos'è una istanza di input reale?

Se l'input è "piccolo"?

Come scala al crescere della grandezza dell'input?

Requisiti concreta definizione.

Platform-independent

Instance-independent

Predictive value (grandezza crescente input)

Più "matematica"

Analisi del caso peggiore

Running time nel caso peggiore. Limitazione sul running time **più grande possibile** di un algoritmo su input di una grandezza fissata N .

- Generalmente è una buona misura in pratica.
- Troppo pessimisti? E' difficile trovare alternative.

Analisi del caso peggiore

Running time nel caso peggiore. Limitazione sul running time **più grande possibile** di un algoritmo su input di una grandezza fissata N.

- Generalmente è una buona misura in pratica.
- Troppo pessimisti? E' difficile trovare alternative.

Running time nel caso medio. Limitazione sul running time di un algoritmo su input **casuali** di una grandezza fissata N.

- Difficile (se non impossibile) modellare accuratamente istanze reali con distribuzioni casuali.
- Algoritmi messi a punto per una specifica distribuzione potrebbero andare male per altri input.

Algoritmi brute-force

Per molti problemi non banali c'è un naturale algoritmo che prova tutte le possibilità fino a trovare una soluzione

- Tipicamente richiede tempo 2^N o peggio per input di grandezza N .
- In pratica non utilizzabile.

Una seconda definizione. Un algoritmo è efficiente se ha prestazioni (ad un livello analitico) nel caso peggiore qualitativamente migliori di una ricerca brute-force.

Algoritmi brute-force

Provare tutte le possibilità fino a trovare una soluzione

Generalmente l'algoritmo brute-force ha running-time proibitivo

Una seconda definizione. Un algoritmo è efficiente se ha prestazioni (ad un livello analitico) nel caso peggiore qualitativamente migliori di una ricerca brute-force.

Definizione vaga. Che significa "qualitativamente migliori"?

Dobbiamo essere più precisi!

Caso peggiore polinomiale

Proprietà di scalabilità. Quando la grandezza dell'input raddoppia, il running time dell' algoritmo peggiora al massimo di un fattore costante C .

Una terza definizione. Un algoritmo è efficiente se ha un running-time polinomiale.

Esistono costanti assolute $c, d > 0$ tali che per ogni istanza di input di grandezza N , il suo running time è $< cN^d$ (misurando i passi computazionali primitivi).

Proprietà di scaling. Se input cresce da N a $2N$ allora il limite cresce da cN^d a $c2^d N^d$, cioè di un fattore costante 2^d

Caso peggiore polinomiale

Una terza definizione. Un algoritmo è efficiente se ha un running-time polinomiale.

Troppo matematica? E se ... $9.04 \times 10^{20} \times N^{100}$ $N^{1.01(\log N)}$

Giustificazione:

- Funziona bene in pratica!
- In pratica, gli algoritmi polinomiali che vengono progettati hanno quasi sempre costanti ed esponenti piccoli.

Eccezioni.

- Alcuni algoritmi polinomiali hanno costanti e/o esponenti grandi, e sono inutili in pratica.
- Alcuni algoritmi esponenziali sono utilizzati perchè le istanze cattive sono rare.

Grafico di alcuni running time

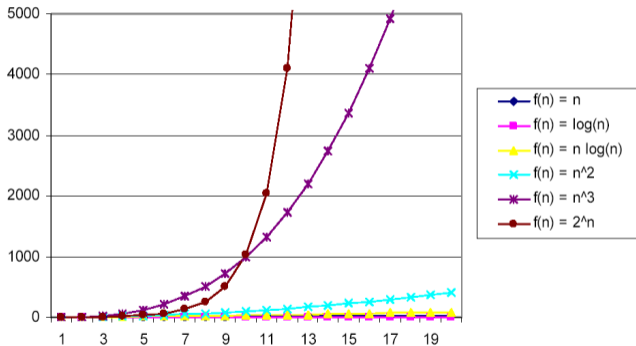


Tabella di alcuni running-time

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

2.2 Asymptotic Order of Growth

Asymptotic Order of Growth

Running time $1.62n^2 + 3.5n + 8$ istruzioni alto livello

Se occorrono 25 istruzioni basso livello per ogni istruzione alto livello

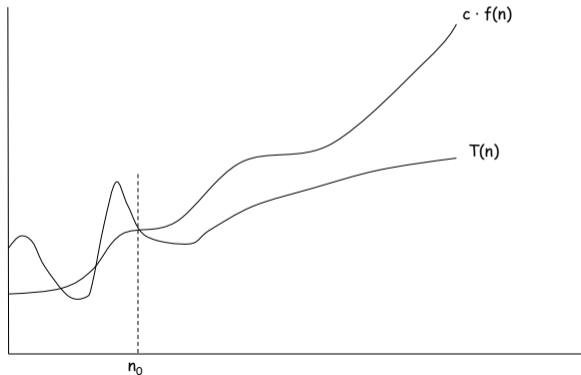
Allora Running time $40.5n^2 + 87.5n + 200$ istruzioni basso livello

Non sono importanti le costanti ed i termini che crescono più lentamente

Il running time è "essenzialmente" n^2

Notazione O

Limite superiore. $T(n)$ è $O(f(n))$ se esistono costanti $c > 0$ e $n_0 \geq 0$ tali che per tutti $n \geq n_0$ si ha $T(n) \leq c \cdot f(n)$.



Notazione O

Limite superiore. $T(n)$ è $O(f(n))$ se esistono costanti $c > 0$ e $n_0 \geq 0$ tali che per tutti $n \geq n_0$ si ha $T(n) \leq c \cdot f(n)$.

$$\begin{aligned} T(n) &= pn^2 + qn + r && \text{assumiamo costanti } p > 0, q, r \geq 0 \\ &\leq pn^2 + qn^2 + rn^2 \\ &\leq (p + q + r) n^2 \end{aligned}$$

Quindi, $T(n) = O(n^2)$, con $c = (p + q + r)$ ed $n_0 = 1$

Inoltre, $T(n) = O(n^3)$, con $c = (p + q + r)$ ed $n_0 = 1$

$T(n)$ non è $O(n)$

Se lo fosse allora esisterebbe $c > 0$ tale che

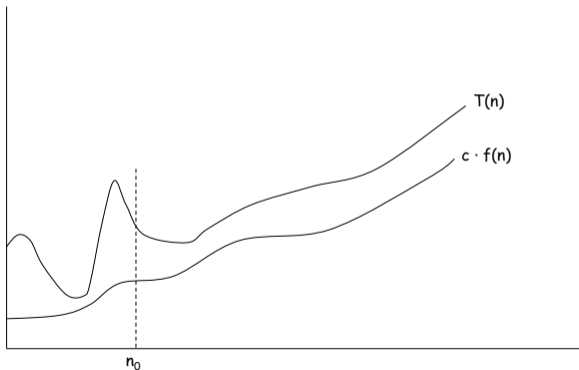
$$pn^2 + qn + r \leq c \cdot n$$

$$pn + q + r/n \leq c$$

impossibile!

Notazione Ω

Limite inferiore. $T(n)$ è $\Omega(f(n))$ se esistono costanti $c > 0$ e $n_0 \geq 0$ tali che per tutti $n \geq n_0$ si ha $T(n) \geq c \cdot f(n)$.



Notazione Ω

Limite inferiore. $T(n)$ è $\Omega(f(n))$ se esistono costanti $c > 0$ e $n_0 \geq 0$ tali che per tutti $n \geq n_0$ si ha $T(n) \geq c \cdot f(n)$.

$$\begin{aligned} T(n) &= pn^2 + qn + r && \text{assumiamo costanti } p > 0, q, r \geq 0 \\ &\geq pn^2 \end{aligned}$$

Quindi, $T(n) = \Omega(n^2)$, con $c = p$ ed $n_0 = 1$

Inoltre, $T(n) = \Omega(n)$, con $c = p$ ed $n_0 = 1$

$T(n)$ non è $\Omega(n^3)$

Se lo fosse allora esisterebbe $c > 0$ tale che

$$\begin{aligned} pn^2 + qn + r &\geq c \cdot n^3 \\ p + q/n + r/n^2 &\geq c \cdot n \end{aligned}$$

impossibile!

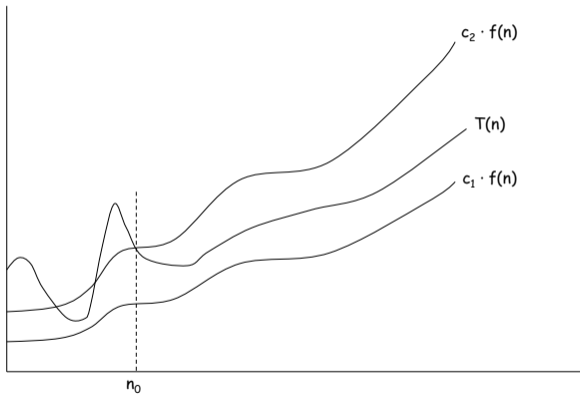
Ordine di crescita asintotica

Limite superiore. $T(n)$ è $O(f(n))$ se esistono costanti $c > 0$ e $n_0 \geq 0$ tali che per tutti $n \geq n_0$ si ha $T(n) \leq c \cdot f(n)$.

Limite inferiore. $T(n)$ è $\Omega(f(n))$ se esistono costanti $c > 0$ e $n_0 \geq 0$ tali che per tutti $n \geq n_0$ si ha $T(n) \geq c \cdot f(n)$.

Limiti stretti. $T(n)$ è $\Theta(f(n))$ se $T(n)$ è sia $O(f(n))$ che $\Omega(f(n))$.

Visualizzazione di $\Theta(f(n))$



Ordine di crescita asintotica

Limite superiore. $T(n)$ è $O(f(n))$ se esistono costanti $c > 0$ e $n_0 \geq 0$ tali che per tutti $n \geq n_0$ si ha $T(n) \leq c \cdot f(n)$.

Limite inferiore. $T(n)$ è $\Omega(f(n))$ se esistono costanti $c > 0$ e $n_0 \geq 0$ tali che per tutti $n \geq n_0$ si ha $T(n) \geq c \cdot f(n)$.

Limiti stretti. $T(n)$ è $\Theta(f(n))$ se $T(n)$ è sia $O(f(n))$ che $\Omega(f(n))$.

Esempio: $T(n) = pn^2 + qn + r$ è $\Theta(n^2)$

Esempio: $T(n) = 32n^2 + 17n + 32$.

- $T(n)$ è $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, e $\Theta(n^2)$.
- $T(n)$ non è $O(n)$, $\Omega(n^3)$, $\Theta(n)$, e nemmeno $\Theta(n^3)$.

Notazione Θ

Se $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = d > 0$ allora $f(n)$ è $\Theta(g(n))$.

Prova. Dalla definizione di limite esiste un $n_0 \geq 0$ tale che il rapporto è in $[d/2, 2d]$, ovvero per tutti $n \geq n_0$

$$\frac{d}{2} \leq \frac{f(n)}{g(n)} \leq 2d$$

Quindi, $f(n) \leq 2d \cdot g(n)$ per tutti $n \geq n_0$. Allora $f(n)$ è $O(g(n))$.

Inoltre, $f(n) \geq (d/2) \cdot g(n)$ per tutti $n \geq n_0$. Allora $f(n)$ è $\Omega(g(n))$.

Limite e notazioni Ω ed O

Se $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$ allora $f(n)$ è $\Omega(g(n))$ e $g(n)$ è $O(f(n))$.

Prova. Dalla definizione di limite esiste un $n_0 \geq 0$ tale che il rapporto è maggiore di $d > 0$, ovvero per tutti $n \geq n_0$

$$\frac{f(n)}{g(n)} \geq d$$

Quindi, $f(n) \geq d \cdot g(n)$ per tutti $n \geq n_0$. Allora $f(n)$ è $\Omega(g(n))$.

Inoltre, $g(n) \leq (1/d) \cdot f(n)$ per tutti $n \geq n_0$. Allora $g(n)$ è $O(f(n))$.

Leggero abuso di notazione

Scriviamo $T(n) = O(f(n))$.

- Asimmetrica:

- Sia $f(n) = 5n^3$; $g(n) = 3n^2$
- Allora $f(n) = O(n^3)$ $g(n) = O(n^3)$
- Ma $f(n) \neq g(n)$.

Notazione migliore $T(n) \in O(f(n))$.

$O(f(n))$ è un insieme

$\{g(n) : \text{Esistono costanti } c, n_0 > 0, \text{ tali che per tutti } n \geq n_0, \text{ si ha } 0 \leq g(n) \leq cf(n)\}$

Uso delle notazioni

Running time è $O(f(n))$

- Caso peggiore

Running time è $\Omega(f(n))$

- Caso migliore

Running time nel caso peggiore è $\Omega(f(n))$

- Caso peggiore è dato da una funzione $g(n) = \Omega(f(n))$

Running time nel caso peggiore è $\Theta(f(n))$

Affermazioni senza senso. Ogni algoritmo di ordinamento basato su confronti richiede almeno $O(n \log n)$ confronti.

- Bisogna utilizzare Ω per i limiti inferiori

Proprietà

Transitiva.

- If $f = O(g)$ and $g = O(h)$ then $f = O(h)$.
- If $f = \Omega(g)$ and $g = \Omega(h)$ then $f = \Omega(h)$.
- If $f = \Theta(g)$ and $g = \Theta(h)$ then $f = \Theta(h)$.

Additiva.

- If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$.
- If $f = \Omega(h)$ and $g = \Omega(h)$ then $f + g = \Omega(h)$.
- If $f = \Theta(h)$ and $g = O(h)$ then $f + g = \Theta(h)$.

Prova?

If $f = O(g)$ and $g = O(h)$ then $f = O(h)$

Ipotesi:

esistono costanti $c, n_0 > 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot g(n)$

esistono costanti $c', n'_0 > 0$ tali che per tutti $n \geq n'_0$ si ha $g(n) \leq c' \cdot h(n)$

Dobbiamo mostrare che:

esistono costanti $c'', n''_0 > 0$ tali che per tutti $n \geq n''_0$ si ha $f(n) \leq c'' \cdot h(n)$

Quanto valgono c'', n''_0 ?

$c'' = ?$

$n''_0 = ?$

If $f = O(g)$ and $g = O(h)$ then $f = O(h)$

Ipotesi:

esistono costanti $c, n_0 > 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot g(n)$

esistono costanti $c', n'_0 > 0$ tali che per tutti $n \geq n'_0$ si ha $g(n) \leq c' \cdot h(n)$

Dobbiamo mostrare che:

esistono costanti $c'', n''_0 > 0$ tali che per tutti $n \geq n''_0$ si ha $f(n) \leq c'' \cdot h(n)$

Quanto valgono c'', n''_0 ?

$$c'' = c \cdot c'$$

$$n''_0 = ?$$

If $f = O(g)$ and $g = O(h)$ then $f = O(h)$

Ipotesi:

esistono costanti $c, n_0 \geq 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot g(n)$

esistono costanti $c', n'_0 \geq 0$ tali che per tutti $n \geq n'_0$ si ha $g(n) \leq c' \cdot h(n)$

Dobbiamo mostrare che:

esistono costanti $c'', n''_0 \geq 0$ tali che per tutti $n \geq n''_0$ si ha $f(n) \leq c'' \cdot h(n)$

Quanto valgono c'', n''_0 ?

$$c'' = c \cdot c'$$

$$n''_0 = \max \{n_0, n'_0\}$$

Proprietà

Transitiva.

- If $f = O(g)$ and $g = O(h)$ then $f = O(h)$.
- If $f = \Omega(g)$ and $g = \Omega(h)$ then $f = \Omega(h)$.
- If $f = \Theta(g)$ and $g = \Theta(h)$ then $f = \Theta(h)$.

Additiva.

- If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$.
- If $f = \Omega(h)$ and $g = \Omega(h)$ then $f + g = \Omega(h)$.
- If $f = \Theta(h)$ and $g = O(h)$ then $f + g = \Theta(h)$.

Prova?

If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$

Ipotesi:

esistono costanti $c, n_0 > 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot h(n)$

esistono costanti $c', n'_0 > 0$ tali che per tutti $n \geq n'_0$ si ha $g(n) \leq c' \cdot h(n)$

Dobbiamo mostrare che:

esistono cost $c'', n''_0 > 0$ tali che per tutti $n \geq n''_0$ si ha $f(n)+g(n) \leq c'' \cdot h(n)$

Quanto valgono c'', n''_0 ?

$c'' = ?$

$n''_0 = ?$

If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$

Ipotesi:

esistono costanti $c, n_0 \geq 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot h(n)$

esistono costanti $c', n'_0 \geq 0$ tali che per tutti $n \geq n'_0$ si ha $g(n) \leq c' \cdot h(n)$

Dobbiamo mostrare che:

esistono cost $c'', n''_0 > 0$ tali che per tutti $n \geq n''_0$ si ha $f(n)+g(n) \leq c'' \cdot h(n)$

Quanto valgono c'', n''_0 ?

$$c'' = c + c'$$

$$n''_0 = ?$$

If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$

Ipotesi:

esistono costanti $c, n_0 \geq 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot h(n)$

esistono costanti $c', n'_0 \geq 0$ tali che per tutti $n \geq n'_0$ si ha $g(n) \leq c' \cdot h(n)$

Dobbiamo mostrare che:

esistono cost $c'', n''_0 > 0$ tali che per tutti $n \geq n''_0$ si ha $f(n)+g(n) \leq c'' \cdot h(n)$

Quanto valgono c'', n''_0 ?

$$c'' = c + c'$$

$$n''_0 = \max \{n_0, n'_0\}$$

Proprietà

Transitiva.

- If $f = O(g)$ and $g = O(h)$ then $f = O(h)$.
- If $f = \Omega(g)$ and $g = \Omega(h)$ then $f = \Omega(h)$.
- If $f = \Theta(g)$ and $g = \Theta(h)$ then $f = \Theta(h)$.

Additiva.

- If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$.
- If $f = \Omega(h)$ and $g = \Omega(h)$ then $f + g = \Omega(h)$.
- If $f = \Theta(h)$ and $g = O(h)$ then $f + g = \Theta(h)$.

Esercizio: Provare le altre proposizioni

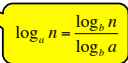
Limiti asintotici per alcune funzioni comuni

Polinomi. $a_0 + a_1n + \dots + a_d n^d$ è $\Theta(n^d)$ se $a_d > 0$.

Tempo polinomiale. Il running time è $O(n^d)$ per qualche costante d indipendente dalla grandezza dell'input n .

Logaritmi. $O(\log_a n) = O(\log_b n)$ per tutte le costanti $a, b > 0$.

↑
quindi possiamo evitare di indicare la base


$$\log_a n = \frac{\log_b n}{\log_b a}$$

Logaritmi. Per ogni $x > 0$, $\log n = O(n^x)$.

↑
log cresce più lentamente di un qualunque polinomio

Esponenziali. Per ogni $r > 1$ e ogni $d > 0$, $n^d = O(r^n)$.

↑
ogni esponenziale cresce più velocemente di un qualunque polinomio

Prova?

Limiti asintotici per alcune funzioni comuni

Esponenziali. Per ogni $r > 1$ e ogni $d > 0$, $n^d = O(r^n)$.

Tempo esponenziale. Il running time è $O(r^n)$ per qualche costante r indipendente dalla grandezza dell' input n .

Un po' ambigua, perchè non è chiaro il valore r

Se $r > s$ allora s^n è $O(r^n)$, ma r^n non è $O(s^n)$

Infatti:

- $s^n < r^n$ quindi s^n è $O(r^n)$
- Se esistessero costanti $c, n_0 > 0$ tali che per tutti $n \geq n_0$ si ha $r^n \leq c \cdot s^n$
Allora $(r/s)^n \leq c$ con $r/s > 1$, quindi r^n non è $O(s^n)$

Solved Exercise 1, pag. 65-66

Ordinare le seguenti funzioni

$$f_1(n) = 10^n$$

$$f_2(n) = n^{1/3}$$

$$f_3(n) = n^n$$

$$f_4(n) = \log_2 n$$

$$f_5(n) = 2^{\sqrt{\log_2 n}}$$

in senso crescente. Cioè, se $g(n)$ segue la funzione $f(n)$ allora $f(n) = O(g(n))$.

Soluzione.

Solved Exercise 1, pag. 65-66

Ordinare le seguenti funzioni

$$f_1(n)=10^n$$

$$f_2(n)=n^{1/3}$$

$$f_3(n)=n^n$$

$$f_4(n)=\log_2 n$$

$$f_5(n)=2^{\sqrt{\log_2 n}}$$

in senso crescente. Cioè, se $g(n)$ segue la funzione $f(n)$ allora $f(n) = O(g(n))$.

Soluzione.

Parte più facile: $f_4(n)=\log_2 n$ $f_2(n)=n^{1/3}$ $f_1(n)=10^n$

Solved Exercise 1, pag. 65-66

Ordinare le seguenti funzioni

$$f_1(n)=10^n$$

$$f_2(n)=n^{1/3}$$

$$f_3(n)=n^n$$

$$f_4(n)=\log_2 n$$

$$f_5(n)=2^{\sqrt{\log_2 n}}$$

in senso crescente. Cioè, se $g(n)$ segue la funzione $f(n)$ allora $f(n) = O(g(n))$.

Soluzione.

Parte più facile: $f_4(n)=\log_2 n$ $f_2(n)=n^{1/3}$ $f_1(n)=10^n$

E $f_3(n)=n^n$?

Solved Exercise 1, pag. 65-66

Ordinare le seguenti funzioni

$$f_1(n) = 10^n$$

$$f_2(n) = n^{1/3}$$

$$f_3(n) = n^n$$

$$f_4(n) = \log_2 n$$

$$f_5(n) = 2^{\sqrt{\log_2 n}}$$

in senso crescente. Cioè, se $g(n)$ segue la funzione $f(n)$ allora $f(n) = O(g(n))$.

Soluzione.

Parte più facile: $f_4(n) = \log_2 n$ $f_2(n) = n^{1/3}$ $f_1(n) = 10^n$ $f_3(n) = n^n$

E $f_5(n) = 2^{\sqrt{\log_2 n}}$?

Solved Exercise 1, pag. 65-66

Ordinare le seguenti funzioni

$$f_1(n) = 10^n$$

$$f_2(n) = n^{1/3}$$

$$f_3(n) = n^n$$

$$f_4(n) = \log_2 n$$

$$f_5(n) = 2^{\sqrt{\log_2 n}}$$

in senso crescente. Cioè, se $g(n)$ segue la funzione $f(n)$ allora $f(n) = O(g(n))$.

Soluzione.

Parte più facile: $f_4(n) = \log_2 n$ $f_2(n) = n^{1/3}$ $f_1(n) = 10^n$ $f_3(n) = n^n$

E $f_5(n) = 2^{\sqrt{\log_2 n}}$?

Consideriamo i logaritmi:

$$\log f_4(n) = \log (\log_2 n)$$
$$\log f_5(n) = \log 2^{\sqrt{\log_2 n}} = \sqrt{\log n}$$
$$\log f_2(n) = \log n^{1/3} = (1/3) \log n$$
$$\log f_1(n) = \log 10^n = (\log 10) n$$

Solved Exercise 1, pag. 65-66

Ordinare le seguenti funzioni

$$f_1(n) = 10^n$$

$$f_2(n) = n^{1/3}$$

$$f_3(n) = n^n$$

$$f_4(n) = \log_2 n$$

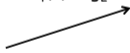
$$f_5(n) = 2^{\sqrt{\log_2 n}}$$

in senso crescente. Cioè, se $g(n)$ segue la funzione $f(n)$ allora $f(n) = O(g(n))$.

Soluzione.

Parte più facile: $f_4(n) = \log_2 n$ $f_2(n) = n^{1/3}$ $f_1(n) = 10^n$ $f_3(n) = n^n$

E $f_5(n) = 2^{\sqrt{\log_2 n}}$?



Consideriamo i logaritmi: $\log f_4(n) = \log (\log_2 n)$

$$\log f_5(n) = \log 2^{\sqrt{\log_2 n}} = \sqrt{\log n}$$

$$\log f_2(n) = \log n^{1/3} = (1/3)\log n$$

Continuare lo svolgimento ...

Solved Exercise 2, pag. 66

Se $f = O(g)$ allora $g = \Omega(f)$

Prova.

Ipotesi. esistono costanti $c, n_0 > 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot g(n)$.

Tesi. esistono costanti $c', n'_0 > 0$ tali che per tutti $n \geq n'_0$ si ha $g(n) \geq c' \cdot f(n)$.

Quanto valgono c', n'_0 ?

$c' = ?$

$n'_0 = ?$

Solved Exercise 2, pag. 66

Se $f = O(g)$ allora $g = \Omega(f)$

Prova.

Ipotesi. esistono costanti $c, n_0 > 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot g(n)$.

Tesi. esistono costanti $c', n'_0 > 0$ tali che per tutti $n \geq n'_0$ si ha $g(n) \geq c' \cdot f(n)$.

Quanto valgono c', n'_0 ?

$$c' = 1/c$$

$$n'_0 = n_0$$

Esercizio 5 pag 68

Assumiamo che le funzioni positive $f(n)$ e $g(n)$ soddisfano $f(n) = O(g(n))$. Allora per ognuna delle seguenti affermazioni dire se è vera oppure falsa e dare una dimostrazione oppure un controesempio

a) $\log_2 f(n)$ è $O(\log_2 g(n))$

b) $2^{f(n)}$ è $O(2^{g(n)})$

c) $f(n)^2$ è $O(g(n)^2)$

Svolgimento.

Esercizio 5 pag 68

Assumi che le funzioni positive $f(n)$ e $g(n)$ soddisfano $f(n) = O(g(n))$. Allora per ognuna delle seguenti affermazioni dire se è vera oppure falsa e dare una dimostrazione oppure un controesempio

a) $\log_2 f(n)$ è $O(\log_2 g(n))$

b) $2^{f(n)}$ è $O(2^{g(n)})$

c) $f(n)^2$ è $O(g(n)^2)$

Svolgimento b).

Ipotesi: esistono costanti $c, n_0 > 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot g(n)$

Tesi: esistono costanti $c', n'_0 > 0$ tali che per tutti $n \geq n'_0$ si ha $2^{f(n)} \leq c' \cdot 2^{g(n)}$

Vera o falsa?

Esercizio 5 pag 68

Assumi che le funzioni positive $f(n)$ e $g(n)$ soddisfano $f(n) = O(g(n))$. Allora per ognuna delle seguenti affermazioni dire se è vera oppure falsa e dare una dimostrazione oppure un controesempio

a) $\log_2 f(n)$ è $O(\log_2 g(n))$

b) $2^{f(n)}$ è $O(2^{g(n)})$

c) $f(n)^2$ è $O(g(n)^2)$

Svolgimento b).

Ipotesi: esistono costanti $c, n_0 > 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot g(n)$

Tesi: esistono costanti $c', n'_0 > 0$ tali che per tutti $n \geq n'_0$ si ha $2^{f(n)} \leq c' \cdot 2^{g(n)}$

Controesempio: $f(n) = 2n$ $g(n) = n$

Infatti $2^{2n} \leq c' \cdot 2^n$ cioè $2^n \leq c'$ che è impossibile

Esercizio 5 pag 68

Assumi che le funzioni positive $f(n)$ e $g(n)$ soddisfano $f(n) = O(g(n))$. Allora per ognuna delle seguenti affermazioni dire se è vera oppure falsa e dare una dimostrazione oppure un controesempio

a) $\log_2 f(n)$ è $O(\log_2 g(n))$

b) $2^{f(n)}$ è $O(2^{g(n)})$

c) $f(n)^2$ è $O(g(n)^2)$

Svolgimento c).

Ipotesi: esistono costanti $c, n_0 > 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot g(n)$

Tesi: esistono costanti $c', n'_0 > 0$ tali che per tutti $n \geq n'_0$ si ha $f(n)^2 \leq c' \cdot g(n)^2$

Vera o falsa?

Esercizio 5 pag 68

Assumi che le funzioni positive $f(n)$ e $g(n)$ soddisfano $f(n) = O(g(n))$. Allora per ognuna delle seguenti affermazioni dire se è vera oppure falsa e dare una dimostrazione oppure un controesempio

a) $\log_2 f(n)$ è $O(\log_2 g(n))$

b) $2^{f(n)}$ è $O(2^{g(n)})$

c) $f(n)^2$ è $O(g(n)^2)$

Svolgimento c).

Ipotesi: esistono costanti $c, n_0 > 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot g(n)$

Tesi: esistono costanti $c', n'_0 > 0$ tali che per tutti $n \geq n'_0$ si ha $f(n)^2 \leq c' \cdot g(n)^2$

$$c' = c^2 \quad n'_0 = n_0$$

Esercizio 5 pag 68

Assumi che le funzioni positive $f(n)$ e $g(n)$ soddisfano $f(n) = O(g(n))$. Allora per ognuna delle seguenti affermazioni dire se è vera oppure falsa e dare una dimostrazione oppure un controesempio

a) $\log_2 f(n)$ è $O(\log_2 g(n))$

b) $2^{f(n)}$ è $O(2^{g(n)})$

c) $f(n)^2$ è $O(g(n)^2)$

Svolgimento a).

Ipotesi: esistono costanti $c, n_0 > 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot g(n)$

Tesi: esist cost $c', n'_0 > 0$ tali che per tutti $n \geq n'_0$ si ha $\log_2 f(n) \leq c' \cdot \log_2 g(n)$

Controesempio: $f(n) = 2^{1+1/n}$ $g(n) = 2^{1/n}$

Quindi $\log_2 f(n) = 1+1/n$ $\log_2 g(n) = 1/n$

Infatti, se esistesse c', n'_0 tale che $1+1/n \leq c' \cdot 1/n$

ovvero $n+1 \leq c'$ che è impossibile

Esercizio 5 pag 68

Assumi che le funzioni positive $f(n)$ e $g(n)$ soddisfano $f(n) = O(g(n))$. Allora per ognuna delle seguenti affermazioni dire se è vera oppure falsa e dare una dimostrazione oppure un controesempio

a) $\log_2 f(n)$ è $O(\log_2 g(n))$

b) $2^{f(n)}$ è $O(2^{g(n)})$

c) $f(n)^2$ è $O(g(n)^2)$

Svolgimento a). Assumiamo $g(n) > 2$

Ipotesi: esistono costanti $c, n_0 > 0$ tali che per tutti $n \geq n_0$ si ha $f(n) \leq c \cdot g(n)$

Tesi: esist cost $c', n'_0 > 0$ tali che per tutti $n \geq n'_0$ si ha $\log_2 f(n) \leq c' \cdot \log_2 g(n)$

$$\log_2 f(n) \leq \log_2 c + \log_2 g(n) \leq (1 + \log_2 c) \cdot \log_2 g(n) \quad \text{prendiamo } c > 2$$

$$c' = 1 + \log_2 c \quad n'_0 = n_0$$

Esercizi

Siano $f(n)$ e $g(n)$ funzioni positive. Allora dire se la seguente affermazione è vera oppure falsa e dare una dimostrazione oppure un controesempio

$$f(n)+g(n) \text{ è } O(\max(f(n),g(n)))$$

Svolgimento.

Se è vera dobbiamo mostrare che:

esistono cost $c, n_0 > 0$ tali che per $n \geq n_0$ si ha $f(n)+g(n) \leq c \cdot \max(f(n), g(n))$

Quanto valgono c, n_0 ?

$c = ?$

$n_0 = ?$

Se è falsa dobbiamo mostrare un controesempio: $f(n) = ?$ $g(n) = ?$

Esercizi

Siano $f(n)$ e $g(n)$ funzioni positive. Allora dire se la seguente affermazione è vera oppure falsa e dare una dimostrazione oppure un controesempio

$$f(n)+g(n) \text{ è } O(\max(f(n),g(n)))$$

Svolgimento.

Se è vera dobbiamo mostrare che:

esistono cost $c, n_0 > 0$ tali che per $n \geq n_0$ si ha $f(n)+g(n) \leq c \cdot \max(f(n), g(n))$

Quanto valgono c, n_0 ?

$$c = 2$$

$$n_0 = 1$$

vera

Esercizi

Siano $f(n)$ e $g(n)$ funzioni positive. Allora dire se la seguente affermazione è vera oppure falsa e dare una dimostrazione oppure un controesempio

$$f(n)+g(n) \text{ è } O(\min(f(n),g(n)))$$

Svolgimento.

Se è vera dobbiamo mostrare che:

esistono cost $c, n_0 > 0$ tali che per $n \geq n_0$ si ha $f(n)+g(n) \leq c \cdot \min(f(n),g(n))$

Quanto valgono c, n_0 ?

$c = ?$

$n_0 = ?$

Se è falsa dobbiamo mostrare un controesempio: $f(n) = ?$ $g(n) = ?$

Esercizi

Siano $f(n)$ e $g(n)$ funzioni positive. Allora dire se la seguente affermazione è vera oppure falsa e dare una dimostrazione oppure un controesempio

$$f(n)+g(n) \text{ è } O(\min(f(n), g(n)))$$

Svolgimento.

Se è vera dobbiamo mostrare che:

esistono cost $c, n_0 > 0$ tali che per $n \geq n_0$ si ha $f(n)+g(n) \leq c \cdot \min(f(n), g(n))$

Quanto valgono c, n_0 ?

$$c = ?$$

$$n_0 = ?$$

Se è falsa dobbiamo mostrare un controesempio: $f(n)=n$ $g(n)=n^2$

$$n+n^2 \leq c \cdot n$$

Dividendo per n si ha $1+n \leq c$

falsa

Esercizi

Siano $f(n)$ e $g(n)$ funzioni positive. Allora dire se la seguente affermazione è vera oppure falsa e dare una dimostrazione oppure un controesempio

$$2f(n)+4g(n) \text{ è } O(\max(f(n),g(n)))$$

Svolgimento.

Se è vera dobbiamo mostrare che:

esistono cost $c, n_0 > 0$ tali che per $n \geq n_0$ si ha $2f(n)+4g(n) \leq c \cdot \max(f(n), g(n))$

Quanto valgono c, n_0 ?

$$c = ?$$

$$n_0 = ?$$

Se è falsa dobbiamo mostrare un controesempio: $f(n) = ?$ $g(n) = ?$

Esercizi

Siano $f(n)$ e $g(n)$ funzioni positive. Allora dire se la seguente affermazione è vera oppure falsa e dare una dimostrazione oppure un controesempio

$$2f(n)+4g(n) \text{ è } O(\max(f(n),g(n)))$$

Svolgimento.

Se è vera dobbiamo mostrare che:

esistono cost $c, n_0 > 0$ tali che per $n \geq n_0$ si ha $2f(n)+4g(n) \leq c \cdot \max(f(n), g(n))$

Quanto valgono c, n_0 ?

$$c = 6$$

$$n_0 = 1$$

vera

Esercizi

Siano $f(n)$ e $g(n)$ funzioni positive. Allora dire se la seguente affermazione è vera oppure falsa e dare una dimostrazione oppure un controesempio

$$\max(f(n), g(n)) \text{ è } \Theta(f(n)+g(n))$$

Svolgimento.

Se è vera dobbiamo mostrare che:

esistono cost $c, c', n_0 > 0$ tali che per $n \geq n_0$ si ha

$$c' \cdot (f(n)+g(n)) \leq \max(f(n), g(n)) \leq c \cdot (f(n)+g(n))$$

Quanto valgono c, c', n_0 ?

$$c = ?$$

$$c' = ?$$

$$n_0 = ?$$

Se è falsa dobbiamo mostrare un controesempio: $f(n) = ?$ $g(n) = ?$

Esercizi

Siano $f(n)$ e $g(n)$ funzioni positive. Allora dire se la seguente affermazione è vera oppure falsa e dare una dimostrazione oppure un controesempio

$$\max(f(n), g(n)) \text{ è } \Theta(f(n)+g(n))$$

Svolgimento.

Se è vera dobbiamo mostrare che:

esistono cost $c, c', n_0 > 0$ tali che per $n \geq n_0$ si ha

$$c' \cdot (f(n)+g(n)) \leq \max(f(n), g(n)) \leq c \cdot (f(n)+g(n))$$

Quanto valgono c, c', n_0 ?

vera

$$c = 1$$

$$c' = 1/2$$

$$n_0 = 1$$

Esercizi

Provare:

- 1) Se $f = \Theta(g)$ allora $g = \Theta(f)$
- 2) $f = O(f)$
- 3) $f = \Omega(f)$
- 4) $f = \Theta(f)$
- 5) Se $f_1 = O(g_1)$ e $f_2 = O(g_2)$ allora $f_1 \cdot f_2 = O(g_1 \cdot g_2)$
- 6) $4f(n) + 9g(n) = \Theta(f(n) + g(n))$
- 7) $2f(n) + 3g(n) = O(\max(f(n), g(n)))$
- 8) $n^{1/\log(n)} = O(1)$
- 9) $2^{\log n} = \Theta(n)$

Esercizio

Per ognuna delle seguenti coppie di funzioni dire se $f(n) = O(g(n))$,
 $f(n) = \Omega(g(n))$, oppure $f(n) = \Theta(g(n))$:

- 1) $f(n) = \log n^2$; $g(n) = \log n + 5$
- 2) $f(n) = n$; $g(n) = \log n^2$
- 3) $f(n) = \log \log n$; $g(n) = \log n$
- 4) $f(n) = n$; $g(n) = \log^2 n$
- 5) $f(n) = n \log n + n$; $g(n) = \log n$
- 6) $f(n) = 10$; $g(n) = \log 10$
- 7) $f(n) = 2^n$; $g(n) = 10n^2$
- 8) $f(n) = 2^n$; $g(n) = 3^n$

Motivare le risposte.

Esercizio: Soluzione

Per ognuna delle seguenti coppie di funzioni dire se $f(n) = O(g(n))$,
 $f(n) = \Omega(g(n))$, oppure $f(n) = \Theta(g(n))$:

- 1) $f(n) = \log n^2$; $g(n) = \log n + 5$
- 2) $f(n) = n$; $g(n) = \log n^2$
- 3) $f(n) = \log \log n$; $g(n) = \log n$
- 4) $f(n) = n$; $g(n) = \log^2 n$
- 5) $f(n) = n \log n + n$; $g(n) = \log n$
- 6) $f(n) = 10$; $g(n) = \log 10$
- 7) $f(n) = 2^n$; $g(n) = 10n^2$
- 8) $f(n) = 2^n$; $g(n) = 3^n$

Soluzione

$$f(n) = \Theta(g(n))$$

$$f(n) = \Omega(g(n))$$

$$f(n) = O(g(n))$$

$$f(n) = \Omega(g(n))$$

$$f(n) = \Omega(g(n))$$

$$f(n) = \Theta(g(n))$$

$$f(n) = \Omega(g(n))$$

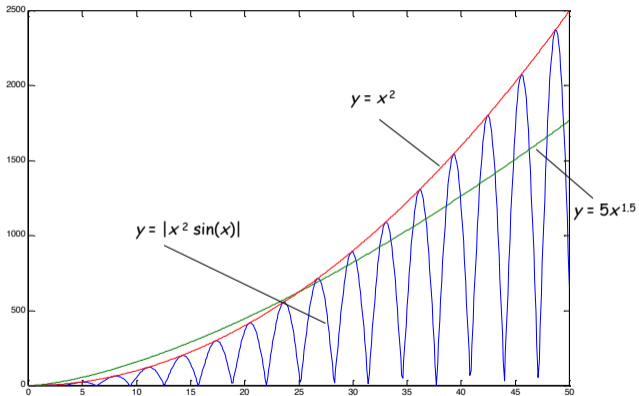
$$f(n) = O(g(n))$$

Motivare le risposte.

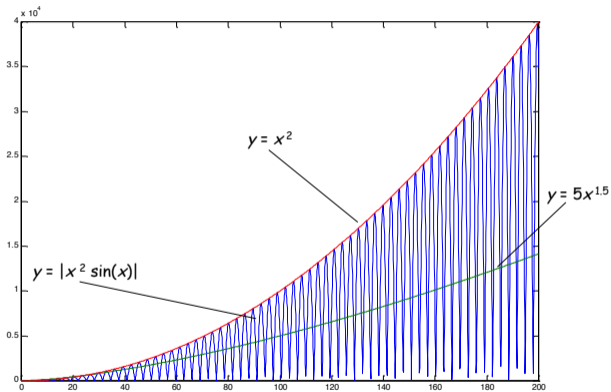
Funzioni incomparabili

Esistono funzioni $f(n)$ e $g(n)$ che non sono comparabili, cioè che non vale nessuna di $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, oppure $f(n) = \Theta(g(n))$?

Funzioni incomparabili



Funzioni incomparabili



Esercizio

Ordinare le seguenti funzioni

$$x + \sin x, \ln x, x + \sqrt{x}, \frac{1}{x}, 13 + \frac{1}{x}, 13 + x, e^x, x^e, x^x$$

$$(x + \sin x)(x^{20} - 102), x \ln x, x(\ln x)^2, \lg_2 x$$

in senso crescente. Cioè, se $g(n)$ segue la funzione $f(n)$ allora $f(n) = O(g(n))$.
Raggruppare insieme tutte le funzioni che sono Θ tra loro.

Esercizio: Soluzione

1. $1/x$

2. $13 + 1/x$

3. $\ln x, \lg_2 x$ (cambio base logaritmi)

4. $x + \sin x, x + \sqrt{x}, 13 + x$ (crescono come x)

5. $x \ln x$

6. $x(\ln x)^2$

7. x^e

8. $(x + \sin x)(x^{10} - 100)$

9. e^x

10. x^x

2.4 A Survey of Common Running Times

Running Time Comuni

- ❑ Linear Time: $O(n)$
- ❑ $O(n \log n)$ Time
- ❑ Quadratic Time: $O(n^2)$
- ❑ Cubic Time: $O(n^3)$
- ❑ Polynomial Time: $O(n^k)$ Time
- ❑ Exponential Time
- ❑ Tempo Sublineare

Linear Time: $O(n)$

Tempo lineare. Il running time è al massimo un fattore costante per la taglia dell'input.

Due esempi:

1. Calcolare il massimo.

2. Merge.

Linear Time: $O(n)$ - Calcolo massimo

Tempo lineare. Il running time è al massimo un fattore costante per la taglia dell'input.

Calcolare il massimo. Calcola il massimo di n numeri a_1, \dots, a_n .

```
max = a1
For i = 2 to n
  If ai > max then
    set max = ai
  Endif
Endfor
```

```
max ← a1
for i = 2 to n {
  if (ai > max)
    max ← ai
}
```

Linear Time: $O(n)$ - Merge

Merge. Combina due liste ordinate $A = a_1, a_2, \dots, a_n$ $B = b_1, b_2, \dots, b_n$ in una sola lista ordinata $C = c_1, c_2, \dots, c_{2n}$

Esempio

Lista A: 2, 3, 11, 19

Lista B: 4, 9, 16, 25

Lista output: 2, 3, 4, 9, 11, 16, 19, 25

Linear Time: $O(n)$ - Merge

Merge. Combina due liste ordinate $A = a_1, a_2, \dots, a_n$ $B = b_1, b_2, \dots, b_n$ in una sola lista ordinata $C = c_1, c_2, \dots, c_{2n}$

Esempio

Lista A: 2, 3, 11, 19

Lista B: 4, 9, 16, 25

Lista output: 2, 3, 4, 9, 11, 16, 19, 25

Append the smaller of a_i and b_j to the output.

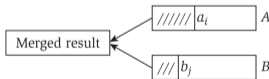


Figure 2.2 To merge sorted lists A and B , we repeatedly extract the smaller item from the front of the two lists and append it to the output.

Linear Time: $O(n)$ - Merge

Merge. Combina due liste ordinate $A = a_1, a_2, \dots, a_n$ $B = b_1, b_2, \dots, b_n$ in una sola lista ordinata $C = c_1, c_2, \dots, c_{2n}$

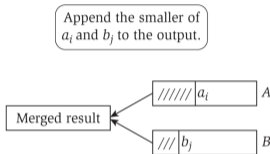
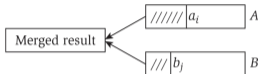


Figure 2.2 To merge sorted lists A and B , we repeatedly extract the smaller item from the front of the two lists and append it to the output.

```
i = 1, j = 1
while (both lists are nonempty) {
    if ( $a_i \leq b_j$ ) append  $a_i$  to output list and increment  $i$ 
    else append  $b_j$  to output list and increment  $j$ 
}
append remainder of nonempty list to output list
```

Linear Time: $O(n)$ - Merge

Merge. Combina due liste ordinate $A = a_1, a_2, \dots, a_n$ $B = b_1, b_2, \dots, b_n$ in una sola lista ordinata $C = c_1, c_2, \dots, c_{2n}$



```
i = 1, j = 1
while (both lists are nonempty) {
    if ( $a_i \leq b_j$ ) append  $a_i$  to output list and increment i
    else append  $b_j$  to output list and increment j
}
append remainder of nonempty list to output list
```

Claim. Il merge di due liste di grandezza n richiede tempo $O(n)$.

Prova. Dopo ogni confronto, la lunghezza della lista output si incrementa di 1.

$O(n \log n)$ Time

Molto comune perchè

- è il running time di algoritmi divide-and-conquer che dividono l'input in due parti, le risolvono ricorsivamente e poi combinano le soluzioni in tempo lineare.
- Running time algoritmi di ordinamento.
Mergesort and Heapsort usano $O(n \log n)$ confronti.
- Molti algoritmi usano l'ordinamento come passo più costoso.

Esempio

Largest empty interval. Dati n time-stamp x_1, \dots, x_n per i tempi di arrivo di copie di n file ad un server, qual'è il più lungo intervallo di tempo senza copie che arrivano al server?

$O(n \log n)$ Time

Molto comune perchè

- è il running time di algoritmi divide-and-conquer che dividono l'input in due parti, le risolvono ricorsivamente e poi combinano le soluzioni in tempo lineare.
- Running time algoritmi di ordinamento.
Mergesort and Heapsort usano $O(n \log n)$ confronti.
- Molti algoritmi usano l'ordinamento come passo più costoso.

Esempio

Largest empty interval. Dati n time-stamp x_1, \dots, x_n per i tempi di arrivo di copie di n file ad un server, qual'è il più lungo intervallo di tempo senza copie che arrivano al server?

Soluzione $O(n \log n)$.

Ordina i time-stamp.

Scorri la lista ordinata, cercando il gap massimo tra intervalli successivi.

$O(n \log n)$ Time

Molto comune perchè

- è il running time di algoritmi divide-and-conquer che dividono l'input in due parti, le risolvono ricorsivamente e poi combinano le soluzioni in tempo lineare.
- Running time algoritmi di ordinamento.
Mergesort and Heapsort usano $O(n \log n)$ confronti.
- Molti algoritmi usano l'ordinamento come passo più costoso.

Esempio

Elementi ripetuti. Dati n elementi x_1, \dots, x_n ci sono due elementi uguali ?

$O(n \log n)$ Time

Molto comune perchè

- è il running time di algoritmi divide-and-conquer che dividono l'input in due parti, le risolvono ricorsivamente e poi combinano le soluzioni in tempo lineare.
- Running time algoritmi di ordinamento.
Mergesort and Heapsort usano $O(n \log n)$ confronti.
- Molti algoritmi usano l'ordinamento come passo più costoso.

Esempio

Elementi ripetuti. Dati n elementi x_1, \dots, x_n ci sono due elementi uguali ?

Soluzione $O(n \log n)$.

Ordina gli elementi.

Scorri la lista ordinata, cercando due elementi successivi uguali.

Quadratic Time: $O(n^2)$

Quadratic time. Enumerare tutte le copie di elementi.

Coppia di punti più vicini. Data una lista di n punti nel piano $(x_1, y_1), \dots, (x_n, y_n)$, trova la coppia più vicina.

Soluzione $O(n^2)$. Minimo tra tutte le coppie.

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

Quadratic Time: $O(n^2)$

Quadratic time. Enumerare tutte le copie di elementi.

Coppia di punti più vicini. Data una lista di n punti nel piano $(x_1, y_1), \dots, (x_n, y_n)$, trova la coppia più vicina.

Soluzione $O(n^2)$. Minimo tra tutte le coppie.

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

```
min ← (x1 - x2)2 + (y1 - y2)2
for i = 1 to n {
  for j = i+1 to n {
    d ← (xi - xj)2 + (yi - yj)2
    if (d < min)
      min ← d
  }
}
```

← Non c'è bisogno della radice quadrata

Quadratic Time: $O(n^2)$

Quadratic time. Enumerare tutte le copie di elementi.

Coppia di punti più vicini. Data una lista di n punti nel piano $(x_1, y_1), \dots, (x_n, y_n)$, trova la coppia più vicina.

Soluzione $O(n^2)$. Minimo tra tutte le coppie.

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

```
min ← (x1 - x2)2 + (y1 - y2)2
for i = 1 to n {
  for j = i+1 to n {
    d ← (xi - xj)2 + (yi - yj)2
    if (d < min)
      min ← d
  }
}
```

← Non c'è bisogno della radice quadrata

Nota. Sembra che $\Omega(n^2)$ sia inevitabile, ma è solo un'illusione (lo vedremo nel Cap. 5)

Cubic Time: $O(n^3)$

Cubic time. Enumerare tutte le triple di elementi

Insiemi disgiunti. Dati n insiemi S_1, \dots, S_n ognuno sottoinsieme di $\{1, 2, \dots, n\}$, c'è tra questi una coppia di insiemi disgiunti?

Soluzione $O(n^3)$. Per ogni coppia di insiemi, verifica se sono disgiunti.

Cubic Time: $O(n^3)$

Cubic time. Enumerare tutte le triple di elementi

Insiemi disgiunti. Dati n insiemi S_1, \dots, S_n ognuno sottoinsieme di $\{1, 2, \dots, n\}$, c'è tra questi una coppia di insiemi disgiunti?

Soluzione $O(n^3)$. Per ogni coppia di insiemi, verifica se sono disgiunti.

```
foreach set  $S_i$  {  
  foreach other set  $S_j$  {  
    foreach element  $p$  of  $S_i$  {  
      determine whether  $p$  also belongs to  $S_j$   
    }  
    if (no element of  $S_i$  belongs to  $S_j$ )  
      report that  $S_i$  and  $S_j$  are disjoint  
  }  
}
```

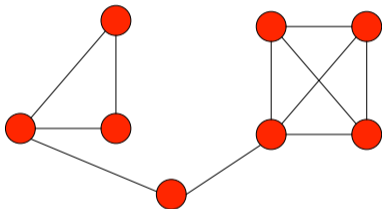
Supponiamo
rappresentazione
 S_i tale che
 $O(1)$



Polynomial Time: $O(n^k)$ Time

Insiemi indipendenti di taglia k . Dato un grafo, esistono k nodi tali che nessuna coppia è connessa da un arco?

k è una costante

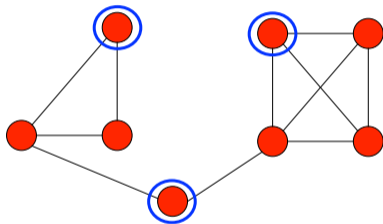


Insieme indipendente di taglia 3?

Polynomial Time: $O(n^k)$ Time

Insiemi indipendenti di taglia k . Dato un grafo, esistono k nodi tali che nessuna coppia è connessa da un arco?

k è una costante

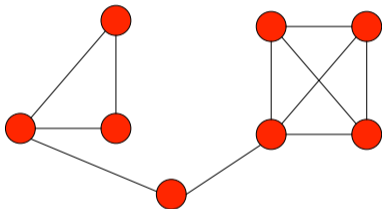


Insieme indipendente di taglia 3?

Polynomial Time: $O(n^k)$ Time

Insiemi indipendenti di taglia k . Dato un grafo, esistono k nodi tali che nessuna coppia è connessa da un arco?

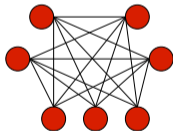
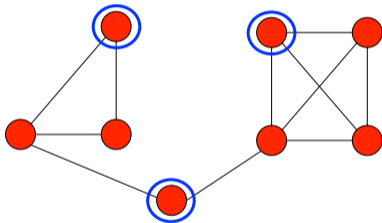
k è una costante



Polynomial Time: $O(n^k)$ Time

Insiemi indipendenti di taglia k . Dato un grafo, esistono k nodi tali che nessuna coppia è connessa da un arco?

k è una costante



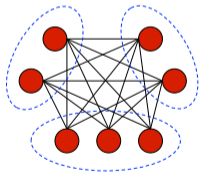
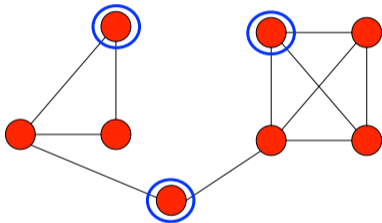
Insieme indipendente di taglia 3?

Insieme indipendente di taglia 4?

Polynomial Time: $O(n^k)$ Time

Insiemi indipendenti di taglia k . Dato un grafo, esistono k nodi tali che nessuna coppia è connessa da un arco?

k è una costante



Polynomial Time: $O(n^k)$ Time

Insiemi indipendenti di taglia k . Dato un grafo, esistono k nodi tali che nessuna coppia è connessa da un arco?

\swarrow
 k è una costante

Soluzione $O(n^k)$. Enumerazione di tutti i sottoinsiemi di k nodi.

```
foreach subset S of k nodes {  
  check whether S is an independent set  
  if (S is an independent set)  
    report S is an independent set  
}
```

- Verifica se S è un insieme indipendente = $O(k^2)$.
- Numero dei sottoinsiemi di k elementi = $\binom{n}{k} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{k(k-1)(k-2)\cdots(2)(1)} \leq \frac{n^k}{k!}$
- $O(k^2 n^k / k!) = O(n^k)$.

\swarrow
Polinomiale per $k=17$
anche se non pratico

Exponential Time

Insiemi indipendenti. Dato un grafo, calcolare la taglia massima di un insieme indipendente.

Soluzione $O(n^2 2^n)$. Enumera tutti i sottoinsiemi.

```
S* ← ∅  
foreach subset S of nodes {  
  check whether S is an independent set  
  if (S is largest independent set seen so far)  
    update S* ← S  
}  
}
```

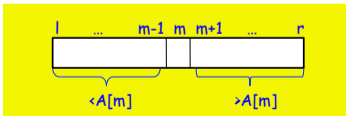
Tempo Sublineare

Non è necessario leggere tutto l'input

Ricerca binaria. Tempo $O(\log n)$

Input:

- array ordinato $A[0] \leq \dots \leq A[m] \leq \dots \leq A[n-1]$
- elemento K



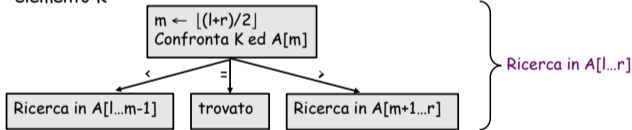
Tempo Sublineare

Non è necessario leggere tutto l'input

Ricerca binaria. Tempo $O(\log n)$

Input:

- array ordinato $A[0] \leq \dots \leq A[m] \leq \dots \leq A[n-1]$
- elemento K



```
BS(A, l, r, K)
if l > r "non trovato"
else m ← ⌊(l+r)/2⌋
  if K = A[m] "trovato"
  else if K < A[m] BS(A, l, m-1, K)
  else BS(A, m+1, r, K)
```

```
l ← 0; r ← n-1
while l ≤ r do
  m ← ⌊(l+r)/2⌋
  if K = A[m] "trovato"
  else if K < A[m] r ← m-1
  else l ← m+1
return "non trovato"
```


Ricerca Binaria: esempio

Invariante. L'algoritmo mantiene $A[l] \leq K \leq A[r]$.

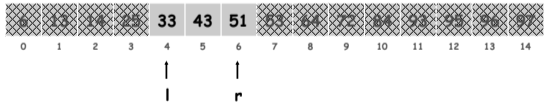
Esempio. Ricerca binaria per 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑							↑							↑
l							m							r

Ricerca Binaria: esempio

Invariante. L'algoritmo mantiene $A[l] \leq K \leq A[r]$.

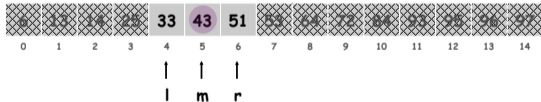
Esempio. Ricerca binaria per 33.



Ricerca Binaria: esempio

Invariante. L'algoritmo mantiene $A[l] \leq K \leq A[r]$.

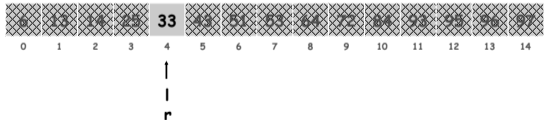
Esempio. Ricerca binaria per 33.



Ricerca Binaria: esempio

Invariante. L'algoritmo mantiene $A[l] \leq K \leq A[r]$.

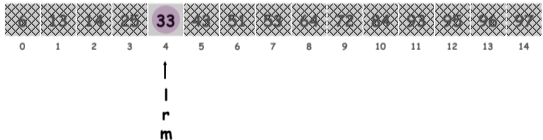
Esempio. Ricerca binaria per 33.



Ricerca Binaria: esempio

Invariante. L'algoritmo mantiene $A[l] \leq K \leq A[r]$.

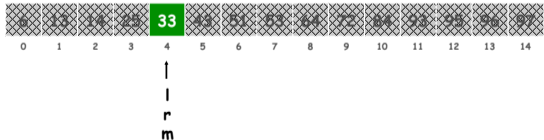
Esempio. Ricerca binaria per 33.



Ricerca Binaria: esempio

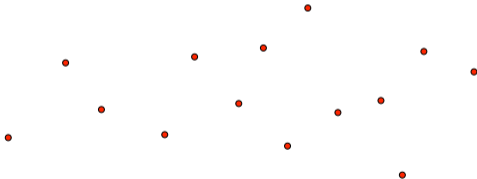
Invariante. L'algoritmo mantiene $A[l] \leq K \leq A[r]$.

Esempio. Ricerca binaria per 33.



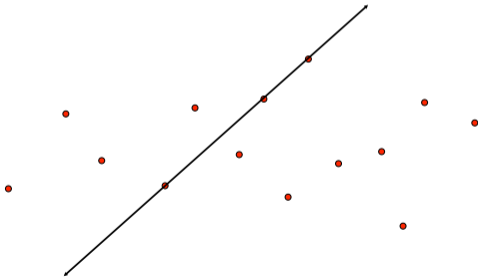
Esercizio

Dati n punti nel piano, determinare se ce ne sono 3 collineari (cioè giacciono su una stessa retta).



Esercizio

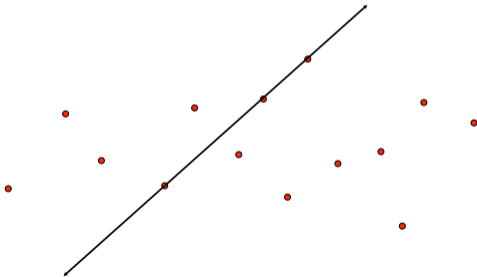
Dati n punti nel piano, determinare se ce ne sono 3 collineari (cioè giacciono su una stessa retta).



Esercizio

Dati n punti nel piano, determinare se ce ne sono 3 collineari (cioè giacciono su una stessa retta).

Soluzione $O(n^3)$. Per ogni tripla di elementi, verificare se sono collineari.

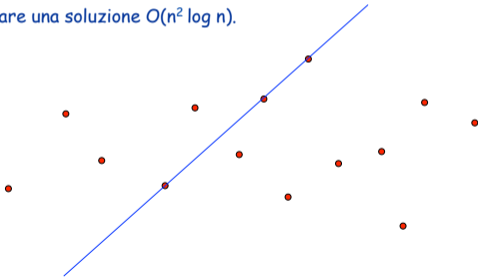


Esercizio

Dati n punti nel piano, determinare se ce ne sono 3 collineari (cioè giacciono su una stessa retta).

Soluzione $O(n^3)$. Per ogni tripla di elementi, verificare se sono collineari.

Cercare una soluzione $O(n^2 \log n)$.

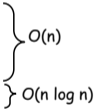


Esercizio

Dati n punti nel piano, determinare se ce ne sono 3 collineari (cioè giacciono su una stessa retta).

Soluzione $O(n^3)$. Per ogni tripla di elementi, verificare se sono collineari.

```
foreach punto P dell'insieme
  L ← ∅
  foreach punto Q dell'insieme (diverso da P) {
    α ← coefficiente angolare retta che connette P e Q
    L ← L ∪ {α}
  }
  if (ci sono duplicati in L) return "trovati"
}
return "non trovati"
```



Complessità $O(n^2 \log n)$