Le Notazioni Asintotiche e l'Analisi della Complessità degli Algoritmi

Prof. Simone Faro
Dispensa didattica per il corso di Algoritmi e Laboratorio
Università di Catania

Anno Accademico 2025–2026

Indice

1 Introduzione (Pinocchio e le due banche)

Per introdurre l'importanza della crescita di una funzione, raccontiamo una piccola storia. Pinocchio ha una sola moneta, del valore di 1 euro. Geppetto, prudente e saggio, gli consiglia di depositarla in banca, che offre un rendimento dello 3,5% mensile. «È poco, ma crescerà nel tempo» gli dice Geppetto.

Poco dopo, Pinocchio incontra il Gatto e la Volpe, che gli fanno una proposta apparentemente irresistibile: «Se lasci la tua moneta nella nostra banca — gli dicono — ti daremo 50 euro ogni mese!»

Pinocchio, attratto dal guadagno immediato, sceglie la seconda opzione. Ma quale delle due strategie è davvero la migliore?

La strategia di Geppetto (interesse composto). Ogni mese la somma cresce del 3,5%. Dopo n mesi, il capitale è:

$$C_G(n) = 1 \times (1{,}035)^n.$$

Si tratta di una funzione *esponenziale*, che cresce lentamente all'inizio ma accelera nel tempo.

La strategia del Gatto e della Volpe (rendita fissa). Ogni mese Pinocchio riceve 50 euro, che si sommano in modo costante:

$$C_{GV}(n) = 50n.$$

Questa è una funzione *lineare*, che aumenta sempre allo stesso ritmo.

All'inizio, il guadagno dei furbi sembra imbattibile, e dopo 12 mesi abbiamo:

$$C_{GV}(12) = 600, \qquad C_G(12) \approx 1{,}56.$$

Dopo un anno, il piccolo investimento di Geppetto non sembra dare frutti, mentre la "banca del campo dei miracoli" ha già fruttato centinaia di euro.

Ma osserviamo cosa accade nel tempo:

Mesi	$C_G(n)$	$C_{GV}(n)$
60	9,3	3,000
120	6,150	6,000
240	3,857,000	12,000

Dopo circa 10 anni (120 mesi), le due curve si incontrano: l'interesse composto raggiunge e supera il guadagno lineare. Da quel momento in poi, la crescita esponenziale prende il sopravvento: dopo 20 anni, Pinocchio — seguendo il consiglio di Geppetto — possiede più di 3 milioni di euro, mentre il Gatto e la Volpe sono fermi a 12 mila.

Questo esempio mostra un principio sorprendente: le crescite esponenziali possono sembrare lente all'inizio, ma sul lungo periodo superano qualunque crescita lineare. Saper riconoscere il tipo di crescita di una funzione significa quindi saper prevedere l'evoluzione di un processo: un'abilità essenziale tanto negli investimenti quanto nell'analisi degli algoritmi.

Come nella storia di Pinocchio, anche in informatica l'apparente "guadagno facile" di un algoritmo semplice può rivelarsi ingannevole: nel lungo periodo, ciò che conta davvero è il ritmo con cui la complessità cresce.

2 Concetti preliminari di analisi asintotica

Quando si analizza l'efficienza di un algoritmo, non ci si limita a misurare il tempo effettivo di esecuzione sul computer, poiché tale misura dipende da fattori contingenti — come la velocità del processore, il linguaggio di programmazione o il sistema operativo. L'analisi algoritmica mira invece a descrivere in modo *astratto e generale* il comportamento dell'algoritmo, indipendentemente dal contesto in cui viene eseguito.

In particolare, si studia come il **tempo di esecuzione** cresce al crescere della dimensione dell'input, indicata con n. Se chiamiamo $T(n)$ il numero di *passi elementari* eseguiti da un algoritmo su un input di dimensione n, allora $T(n)$ rappresenta la *funzione di costo* dell'algoritmo. Un **passo** può essere considerato come un'operazione di base che richiede un tempo costante, ad esempio un confronto, un'assegnazione o un accesso a un elemento di un array. In questo modo, l'analisi non dipende dall'hardware o dal linguaggio, ma solo dal numero relativo di operazioni compiute.

Diremo quindi che due algoritmi sono più o meno efficienti a seconda di come cresce la loro funzione $T(n)$: un algoritmo la cui funzione di costo cresce più lentamente sarà, per input sufficientemente grandi, più efficiente di uno con crescita più rapida. Ma cosa significa che una funzione *cresce più rapidamente* di un'altra? Intuitivamente, significa che per valori crescenti di n, i valori di una funzione diventano molto più grandi rispetto a quelli dell'altra. Ad esempio, n^3 cresce più rapidamente di n^2, perché per n grandi il rapporto $\frac{n^3}{n^2} = n$ tende all'infinito. Allo stesso modo, n^2 cresce più rapidamente di $n \log n$, e quest'ultimo più rapidamente di $\log n$.

L'obiettivo delle **notazioni asintotiche** è proprio quello di rappresentare formalmente questa idea di crescita, trascurando i dettagli che non influiscono sul comportamento generale dell'algoritmo, come le costanti moltiplicative o i termini di ordine inferiore. Esse consentono di classificare gli algoritmi in base al loro *ordine di grandezza*, ovvero in base al termine che domina la crescita della funzione per n molto grandi.

Immaginiamo, ad esempio, di avere due algoritmi che risolvono lo stesso problema: uno con tempo di esecuzione $T_1(n) = 5n^2 + 3n + 10$ e un altro con tempo $T_2(n) = 0.1n^3$. Per piccoli valori di n, il primo può risultare più lento a causa del fattore costante 5; tuttavia, per n grandi, il termine cubico di $T_2(n)$ diventa dominante, e il secondo algoritmo crescerà molto più rapidamente. Le notazioni asintotiche permettono di formalizzare questa osservazione, scrivendo che $T_1(n) = \Theta(n^2)$ e $T_2(n) = \Theta(n^3)$: in altre parole, il primo algoritmo ha una complessità quadratica, il secondo cubica.

In sintesi, le notazioni asintotiche ci offrono un linguaggio rigoroso e sintetico per confrontare algoritmi, esprimendo non tanto *quanto tempo* impiegheranno, ma *come* il loro tempo cresce con la dimensione dell'input.

2.1 Confronto tra le principali funzioni di crescita

Prima di introdurre formalmente le notazioni asintotiche, è utile esaminare alcune tra le funzioni matematiche più comuni che descrivono la crescita del tempo di esecuzione di un algoritmo. Ogni algoritmo, infatti, può essere associato a una funzione $T(n)$ che esprime approssimativamente il numero di operazioni necessarie per risolvere un problema di dimensione n. Il modo in cui $T(n)$ cresce al crescere di n determina la sua *efficienza asintotica*.

Alcune funzioni crescono lentamente, altre molto rapidamente. Comprendere queste differenze è essenziale per valutare in modo intuitivo il comportamento di un algoritmo.

Immaginiamo di avere diversi algoritmi che risolvono lo stesso problema, ma con funzioni di costo differenti: uno impiega un numero di passi proporzionale a $\log n$, un altro a n, un altro ancora a n^2. Per valori piccoli di n, le differenze tra di essi possono sembrare trascurabili; ma per input molto grandi, le funzioni con crescita più rapida diventano rapidamente insostenibili in termini di tempo di esecuzione.

Ad esempio, se $n = 1000$:

$$\log_2 n \approx 10, \quad n = 1000, \quad n^2 = 10^6, \quad 2^n \approx 10^{301}.$$

Ciò dimostra quanto rapidamente una funzione esponenziale superi una quadratica o una lineare.

In generale, possiamo distinguere alcuni **tipi fondamentali di crescita**, ciascuno con un nome specifico, che riassumiamo nella tabella seguente.

Funzione di crescita	**Nome**
1	Costante
$\log n$	Logaritmica
n	Lineare
$n \log n$	Linearitmica (o quasi-lineare)
n^2	Quadratica
n^3	Cubica
n^k (con $k > 1$)	Polinomiale
2^n	Esponenziale
$n!$	Fattoriale

Tabella 1: Confronto tra funzioni di crescita e loro denominazioni comuni.

All'interno della tabella, le funzioni sono elencate in ordine di crescita crescente: una funzione situata più in basso cresce più rapidamente di tutte quelle sopra di essa. In altre parole, per n grandi:

$$1 \ < \ \log n \ < \ n \ < \ n \log n \ < \ n^2 \ < \ n^3 \ < \ n^k \ < \ 2^n \ < \ n!.$$

- Le **funzioni costanti** ($T(n) = 1$) rappresentano algoritmi che impiegano sempre lo stesso numero di operazioni, indipendentemente dalla dimensione dell'input. Esempio: accedere a un elemento in un array.
- Le **funzioni logaritmiche** ($T(n) = \log n$) descrivono algoritmi che riducono il problema di un fattore costante a ogni passo, come la ricerca binaria.
- Le **funzioni lineari** ($T(n) = n$) corrispondono ad algoritmi che analizzano tutti gli elementi dell'input una sola volta, come la scansione di un array.
- Le **funzioni linearitmiche** ($T(n) = n \log n$) caratterizzano algoritmi efficienti di ordinamento, come Merge Sort e Heap Sort.

- Le **funzioni polinomiali** ($T(n) = n^k$) includono molti algoritmi praticabili, ma diventano rapidamente onerose per $k > 2$.
- Le **funzioni esponenziali** e **fattoriali** descrivono problemi di natura combinatoria, per i quali il numero di soluzioni cresce esplosivamente con n (ad esempio, la ricerca esaustiva o il calcolo di tutte le permutazioni).

Questa panoramica consente di visualizzare in modo intuitivo il significato delle notazioni asintotiche: esse non servono solo a esprimere formule matematiche, ma a classificare gli algoritmi in base al loro comportamento di crescita, cioè alla loro *scalabilità* rispetto alla dimensione dell'input.

2.2 Termini e fattori dominanti e trascurabili

Quando si analizza il tempo di esecuzione di un algoritmo, la funzione di costo $T(n)$ può contenere più termini, ciascuno dei quali rappresenta un contributo parziale al numero totale di operazioni necessarie. Al crescere della dimensione dell'input n, tuttavia, alcuni termini crescono molto più rapidamente di altri: essi sono detti termini dominanti, mentre quelli che aumentano più lentamente vengono considerati trascurabili.

Si chiama termine dominante quello che determina l'andamento complessivo della funzione per valori grandi di n. I termini di ordine inferiore influenzano solo i risultati per input di piccola taglia, ma diventano sempre meno rilevanti man mano che n aumenta. Allo stesso modo, anche i fattori moltiplicativi costanti possono essere considerati trascurabili, poiché non modificano la forma della crescita ma solo la sua scala.

Consideriamo alcuni esempi.

Esempio 1.

$$T(n) = 3n^2 + 5n + 10.$$

In questa funzione, il termine n^2 domina gli altri perché cresce molto più rapidamente di n o di una costante. Quando n diventa grande, $5n$ e 10 diventano insignificanti rispetto a $3n^2$. Anche il coefficiente moltiplicativo 3 è trascurabile dal punto di vista asintotico, perché non cambia la velocità di crescita. Quindi la funzione può essere descritta (eliminando i fattori moltiplicativi) da n^2.

Esempio 2.

$$T(n) = n^3 + 50n^2 + 100n + 500.$$

Il termine dominante è n^3, poiché la sua crescita supera quella di tutti gli altri. Per $n = 100$, ad esempio, $n^3 = 10^6$, mentre $50n^2 = 500\,000$, e i termini successivi risultano via via più piccoli. Quindi la funzione può essere descritta (eliminando i fattori moltiplicativi) da n^3.

Esempio 3.

$$T(n) = n \log n + 100n.$$

Qui il termine $n \log n$ cresce più rapidamente del termine lineare, che risulta trascurabile per n grandi. La funzione può essere approssimata come $n \log n$.

In sintesi, per determinare la complessità asintotica di una funzione:

- si eliminano i termini di ordine inferiore, cioè quelli che crescono più lentamente;
- si ignorano i fattori costanti che moltiplicano i termini;
- si conserva solo il termine che domina la crescita complessiva.

Questo processo consente di isolare il comportamento che effettivamente caratterizza l'algoritmo per input di grandi dimensioni, evitando di farsi influenzare da dettagli numerici che hanno effetto solo nei casi piccoli o specifici.

3 Le notazioni asintotiche

Prima di introdurre le diverse notazioni nel dettaglio, è utile chiarire che cosa si intende, in generale, per **notazione asintotica**. Il termine "asintotico" deriva dal greco *asýmptotos*, che significa "che non si incontra mai": esso descrive il comportamento di una funzione quando la variabile indipendente cresce senza limiti, cioè tende all'infinito.

Nel contesto dell'analisi degli algoritmi, studiare il comportamento *asintotico* di una funzione significa analizzarne l'andamento per valori di n molto grandi, tralasciando i dettagli irrilevanti per piccoli input. Lo scopo è individuare quale termine o fattore domina la crescita della funzione di costo $T(n)$ quando n aumenta.

Una **notazione asintotica** è un modo sintetico per descrivere il *modo in cui una funzione cresce* rispetto a un'altra. Essa non ci dice quanto vale esattamente una funzione, ma *quanto rapidamente aumenta* rispetto ad altre funzioni al crescere di n.

Dire, ad esempio, che $T(n)$ è "dell'ordine di n^2" (cioè $T(n) = \Theta(n^2)$) significa che, per input molto grandi, il numero di operazioni richieste dall'algoritmo cresce in modo proporzionale a n^2, indipendentemente da costanti o termini minori come $3n$ o 10.

In altre parole, le notazioni asintotiche non misurano tempi o quantità precise, ma descrivono *l'andamento della crescita* di una funzione rispetto ad altre di riferimento.

Dal punto di vista matematico, una notazione asintotica rappresenta un insieme di funzioni che condividono lo stesso ordine di crescita. Più precisamente, se $g(n)$ è una funzione positiva, la notazione asintotica $O(g(n))$ (così come le altre notazioni che vedremo a breve) è un insieme di funzioni $f(n)$ che soddisfano una certa relazione di confronto per valori di n sufficientemente grandi.

In generale, le notazioni asintotiche forniscono un linguaggio per esprimere relazioni del tipo:

$f(n)$ cresce non più rapidamente / alla stessa velocità / più rapidamente di $g(n)$.

Esse si basano quindi sul confronto tra le due funzioni nel limite per $n \to \infty$, cioè osservando il loro comportamento quando la dimensione dell'input diventa molto grande.

Consideriamo due funzioni:

$$f(n) = 3n^2 + 5n + 7, \quad g(n) = n^2.$$

Per valori piccoli di n, la differenza tra le due può essere significativa; ma man mano che n cresce, il termine $3n^2$ diventa dominante, e gli altri termini hanno un impatto trascurabile. Possiamo allora dire che $f(n)$ e $g(n)$ hanno la stessa crescita asintotica: in simboli,

$$f(n) = \Theta(g(n)) = \Theta(n^2).$$

Questo tipo di ragionamento è alla base dell'analisi degli algoritmi: l'obiettivo non è conoscere il numero preciso di passi eseguiti, ma capire come il tempo di esecuzione varia al variare della dimensione dell'input, ossia come *cresce asintoticamente* la funzione che lo descrive.

3.1 La notazione Θ (theta)

La notazione Θ fornisce un **vincolo asintotico stretto**: rappresenta l'insieme di tutte le funzioni che crescono *alla stessa velocità* di una funzione di riferimento $g(n)$, a meno di costanti moltiplicative. Essa esprime dunque un'equivalenza asintotica tra funzioni: dire che $f(n) = \Theta(g(n))$ significa che $f(n)$ cresce proporzionalmente a $g(n)$ per valori di n sufficientemente grandi.

Formalmente, si definisce:

$$\Theta(g(n)) = \left\{ f(n) \;\middle|\; \exists\, c_1, c_2, n_0 > 0 \text{ tali che } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n),\ \forall n \geq n_0 \right\}.$$

In altre parole, a partire da una certa soglia n_0, la funzione $f(n)$ è sempre compresa tra due multipli costanti della funzione di riferimento $g(n)$.

In termini intuitivi, la notazione Θ descrive una *crescita bilanciata*: $f(n)$ non cresce né più rapidamente né più lentamente di $g(n)$, ma segue la stessa tendenza, differendo al più per un fattore costante. Ciò significa che, per input molto grandi, la loro proporzione rimane stabile:

$$\frac{f(n)}{g(n)} \to k, \qquad \text{con } 0 < k < \infty.$$

Si considerino i seguenti esempi:

- $3n^2 + 5n + 7 = \Theta(n^2)$, poiché per n grandi il termine quadratico domina, e gli altri hanno effetto trascurabile. In questo caso, possiamo scegliere ad esempio $c_1 = 2$, $c_2 = 4$ e $n_0 = 10$, e verificare che:

  $$2n^2 \leq 3n^2 + 5n + 7 \leq 4n^2 \quad \text{per ogni } n \geq 10.$$

- $5n + 20 = \Theta(n)$, perché la costante additiva 20 non influisce sul comportamento asintotico.

- $\frac{1}{2}n^3 + 100n = \Theta(n^3)$, poiché anche in questo caso il termine cubico determina la crescita dominante.

La notazione Θ è la più informativa tra le notazioni asintotiche, poiché fornisce sia un limite superiore sia uno inferiore: se $f(n) = \Theta(g(n))$, allora $f(n)$ è simultaneamente $O(g(n))$ e $\Omega(g(n))$.

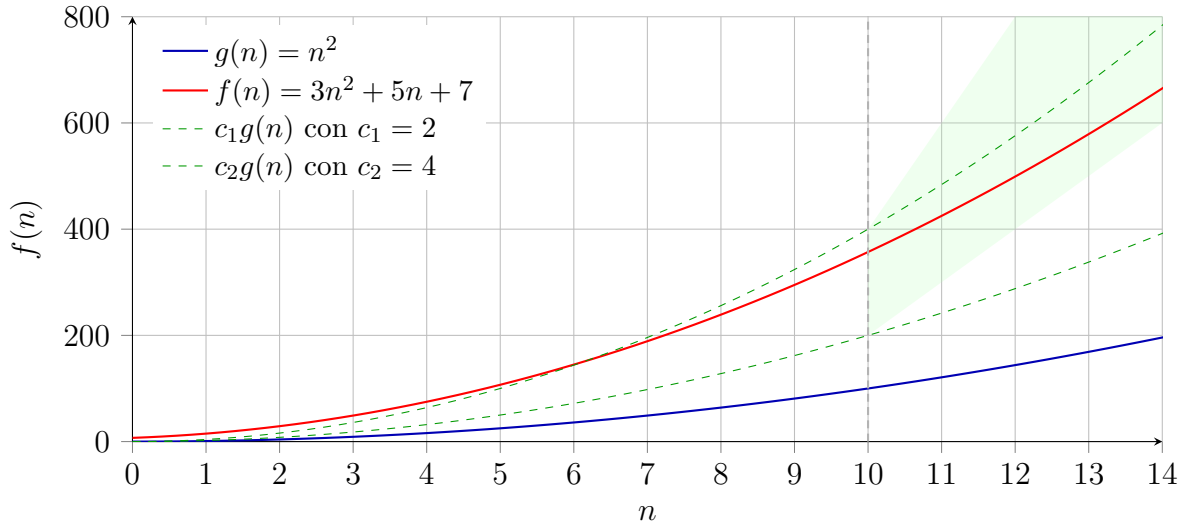

Figura 1: Definizione grafica di $\Theta(g(n))$ con $g(n) = n^2$: per $n \geq n_0 = 10$, la funzione $f(n)$ rimane compresa tra $c_1 g(n)$ e $c_2 g(n)$ (fascia ombreggiata).

La seguente figura mostra in modo intuitivo il significato della definizione formale di $\Theta(g(n))$. Per $n \geq n_0$, la funzione $f(n)$ rimane confinata tra due curve proporzionali a $g(n)$: una moltiplicata per c_1 (in basso) e una per c_2 (in alto).

Questa rappresentazione aiuta a visualizzare il concetto: a partire da un certo punto n_0, la funzione $f(n)$ rimane confinata all'interno di una "fascia" determinata da due multipli costanti della funzione di riferimento. È questo che intendiamo quando diciamo che $f(n)$ e $g(n)$ crescono "alla stessa velocità".

3.2 La notazione O (Big-Oh)

La notazione O (detta *Big-Oh*) rappresenta un **limite superiore asintotico** per una funzione. Essa descrive un insieme di funzioni che, per n sufficientemente grande, non crescono più rapidamente di una funzione di riferimento $g(n)$, a meno di un fattore costante. In altre parole, $O(g(n))$ raccoglie tutte le funzioni la cui crescita è al più proporzionale a quella di $g(n)$.

Formalmente si definisce:

$$O(g(n)) = \left\{ f(n) \;\middle|\; \exists\, c > 0,\ n_0 > 0 \text{ tali che } 0 \leq f(n) \leq c\, g(n),\ \forall n \geq n_0 \right\}.$$

Dire che $f(n) = O(g(n))$ significa che, a partire da una certa soglia n_0, il valore di $f(n)$ non supera mai quello di $c\, g(n)$.

Dal punto di vista intuitivo, la notazione O serve a stabilire una *stima superiore* del comportamento asintotico: essa garantisce che $f(n)$ non crescerà mai più velocemente di $g(n)$ per n grandi, anche se per valori piccoli di n le due funzioni possono intersecarsi o oscillare.
Si considerino i seguenti esempi:

- $3n^2 + 5n + 7 = O(n^2)$, poiché esistono costanti c e n_0 tali che $3n^2 + 5n + 7 \leq c\, n^2$ per ogni $n \geq n_0$. Ad esempio, scegliendo $c = 5$ e $n_0 = 10$, l'ineguaglianza risulta verificata.

- $7n + 200 = O(n)$, perché il termine costante non influisce sulla crescita asintotica.
- $\log n = O(n)$, poiché la funzione logaritmica cresce più lentamente di quella lineare.
- $n = O(n^2)$, perché la funzione lineare cresce meno rapidamente della quadratica.

È importante osservare che la notazione O indica solo un limite *superiore*: se diciamo che $T(n) = O(n^2)$, stiamo affermando che l'algoritmo non richiede mai più di un tempo proporzionale a n^2, ma potrebbe essere anche più efficiente. Per questo motivo, O viene spesso usata per descrivere il **caso peggiore** (*worst case*) di un algoritmo.

La figura seguente mostra graficamente la definizione di $O(g(n))$. Per $n \geq n_0$, la funzione $f(n)$ resta sempre al di sotto della curva $c\,g(n)$, che ne rappresenta il limite superiore asintotico.

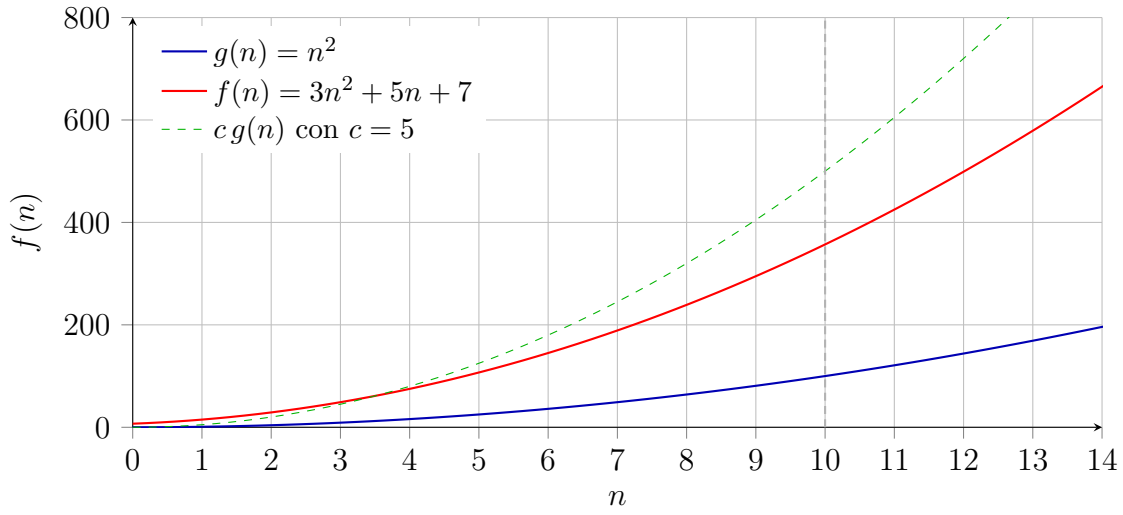

Figura 2: Rappresentazione grafica della definizione di $O(g(n))$: per $n \geq n_0$, $f(n)$ rimane sempre al di sotto del limite superiore $c\,g(n)$.

Questa rappresentazione aiuta a visualizzare la relazione: la curva verde $c\,g(n)$ delimita una soglia oltre la quale la funzione $f(n)$ non può più crescere più rapidamente di $g(n)$. In altre parole, $f(n)$ rimane sempre confinata al di sotto del suo limite asintotico.

3.3 La notazione Ω (Big-Omega)

La notazione Ω rappresenta un **limite inferiore asintotico** per una funzione. Essa raccoglie tutte le funzioni che, per n sufficientemente grande, crescono almeno tanto rapidamente quanto una funzione di riferimento $g(n)$, a meno di un fattore costante. In altre parole, $f(n) = \Omega(g(n))$ significa che, per input grandi, $f(n)$ non potrà mai crescere più lentamente di $g(n)$ (a parte costanti moltiplicative).

Formalmente, si definisce:

$$\Omega(g(n)) = \left\{ f(n) \;\middle|\; \exists\, c > 0,\ n_0 > 0 \text{ tali che } 0 \leq c\,g(n) \leq f(n),\ \forall n \geq n_0 \right\}.$$

Ciò significa che, a partire da un certo punto n_0, il valore di $f(n)$ è sempre maggiore o uguale a una costante moltiplicativa di $g(n)$.

In termini intuitivi, la notazione Ω descrive una *crescita minima garantita*: indica che, oltre una certa soglia, la funzione $f(n)$ rimane sempre al di sopra della funzione di riferimento, salvo differenze costanti. È dunque il complemento naturale della notazione O, che invece stabilisce un limite superiore.
Si considerino i seguenti esempi:

- $3n^2 + 5n + 7 = \Omega(n^2)$, poiché esistono costanti c e n_0 tali che $3n^2 + 5n + 7 \geq c\,n^2$ per ogni $n \geq n_0$. Ad esempio, con $c = 2$ e $n_0 = 10$, la condizione risulta soddisfatta.
- $5n + 200 = \Omega(n)$, perché il termine lineare domina sul termine costante.
- $n^3 = \Omega(n^2)$, poiché la funzione cubica cresce più rapidamente della quadratica.
- $n = \Omega(\log n)$, perché la funzione lineare cresce più rapidamente di quella logaritmica.

Dal punto di vista dell'analisi degli algoritmi, la notazione Ω viene spesso utilizzata per descrivere il **caso migliore** (*best case*) o per fornire una stima del comportamento minimo che un algoritmo deve necessariamente raggiungere.

La figura seguente illustra graficamente il significato di $\Omega(g(n))$. Per $n \geq n_0$, la funzione $f(n)$ rimane sempre al di sopra della curva $c\,g(n)$, che rappresenta il limite inferiore asintotico.

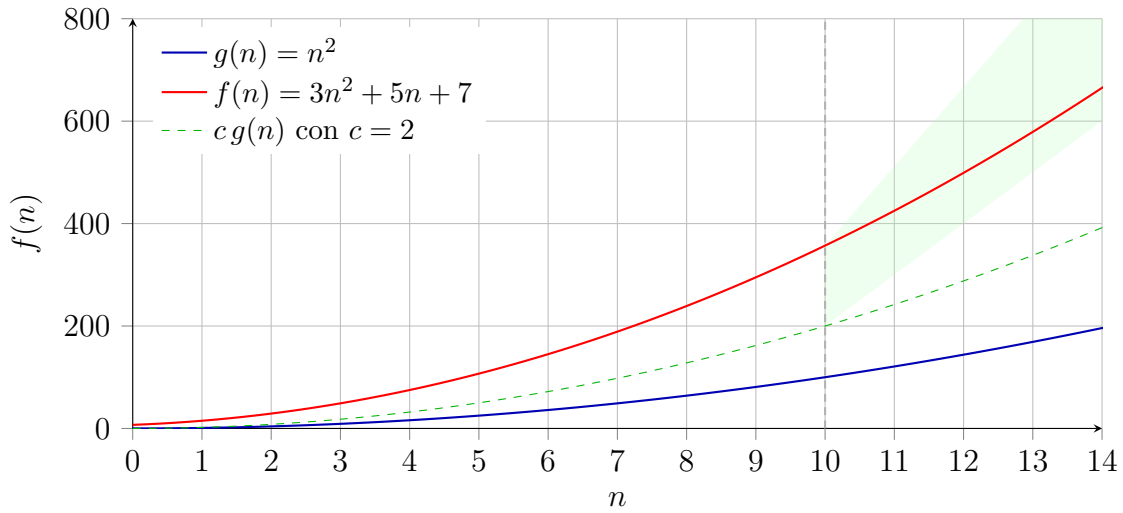

Figura 3: Definizione grafica di $\Omega(g(n))$: per $n \geq n_0$, la funzione $f(n)$ rimane sempre al di sopra del limite inferiore $c\,g(n)$ (area ombreggiata).

La figura evidenzia che, al crescere di n, la funzione $f(n)$ resta confinata nella regione superiore rispetto a $c\,g(n)$. Questo rappresenta il concetto fondamentale della notazione Ω: a partire da un certo punto, la crescita di $f(n)$ non potrà più essere più lenta di quella di $g(n)$, ma solo uguale o più rapida.

3.4 Le notazioni o e ω (piccole)

Accanto alle notazioni O e Ω, che rappresentano limiti asintotici *ampi*, esistono le loro controparti "minuscole" o e ω, che esprimono limiti *stretti*. Queste versioni "piccole"

descrivono situazioni in cui una funzione cresce *più lentamente* oppure *più velocemente* di un'altra, senza mai raggiungerne la stessa velocità di crescita, nemmeno a fattore costante.

In altre parole, mentre O e Ω permettono l'uguaglianza fino a un fattore moltiplicativo costante, le notazioni o e ω indicano una disuguaglianza asintotica *stretta*, cioè che si mantiene indefinitamente al crescere di n.

La prima, $o(g(n))$, esprime che $f(n)$ cresce **più lentamente** di $g(n)$:

$$f(n) = o(g(n)) \quad \iff \quad \forall\, c > 0,\ \exists\, n_0 > 0 \text{ tale che } f(n) < c\, g(n),\ \forall n \geq n_0,$$

oppure, in forma equivalente,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

In modo intuitivo, significa che il rapporto $\frac{f(n)}{g(n)}$ tende a zero: $f(n)$ cresce in modo così più lento di $g(n)$ che, per n grandi, il suo contributo diventa trascurabile. Esempi tipici sono:

$$\log n = o(n), \qquad n = o(n^2), \qquad n^2 = o(2^n).$$

Nel linguaggio dell'analisi degli algoritmi, dire che un tempo di esecuzione è $o(n^2)$ significa che esso cresce *più lentamente* di qualunque funzione proporzionale a n^2, anche moltiplicata per una costante.

La seconda, $\omega(g(n))$, ha significato opposto: descrive funzioni che crescono **più rapidamente** di $g(n)$, senza mai potersi considerare "dello stesso ordine":

$$f(n) = \omega(g(n)) \quad \iff \quad \forall\, c > 0,\ \exists\, n_0 > 0 \text{ tale che } f(n) > c\, g(n),\ \forall n \geq n_0,$$

oppure, in forma equivalente,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty.$$

Ciò significa che, per n sufficientemente grandi, $f(n)$ supera qualsiasi multiplo di $g(n)$, crescendo sempre più rapidamente. Ad esempio:

$$n^2 = \omega(n), \qquad n \log n = \omega(n), \qquad 2^n = \omega(n^3).$$

Possiamo dunque pensare alle notazioni o e ω come versioni *esclusive* di O e Ω: indicano una crescita più lenta o più rapida che non raggiunge mai quella della funzione di riferimento, neppure asintoticamente.

La seguente figura mostra in modo intuitivo la differenza tra O, Θ, e o: nel primo caso (O) la funzione rossa rimane al di sotto della verde ma può "tangere" la sua crescita asintotica; nel secondo (o), la funzione rossa cresce sempre più lentamente e tende a distaccarsi dalla verde.

Allo stesso modo, la distinzione tra Ω e ω riflette un rapporto analogo ma dal lato inferiore: $\Omega(g(n))$ indica una crescita almeno pari a $g(n)$, mentre $\omega(g(n))$ indica una crescita più rapida, che supera qualunque multiplo costante di $g(n)$.

Riassumendo, le relazioni tra tutte le principali notazioni asintotiche possono essere rappresentate in modo sintetico come:

$$\omega(g(n)) \ \supset\ \Omega(g(n)) \ \supset\ \Theta(g(n)) \ \subset\ O(g(n)) \ \subset\ o(g(n)).$$

Le notazioni "grandi" (O, Ω, Θ) indicano limiti che possono essere raggiunti, mentre le notazioni "piccole" (o, ω) esprimono limiti che non possono mai essere toccati asintoticamente. Esse ci permettono di distinguere in modo preciso non solo la direzione della crescita, ma anche la sua intensità relativa.

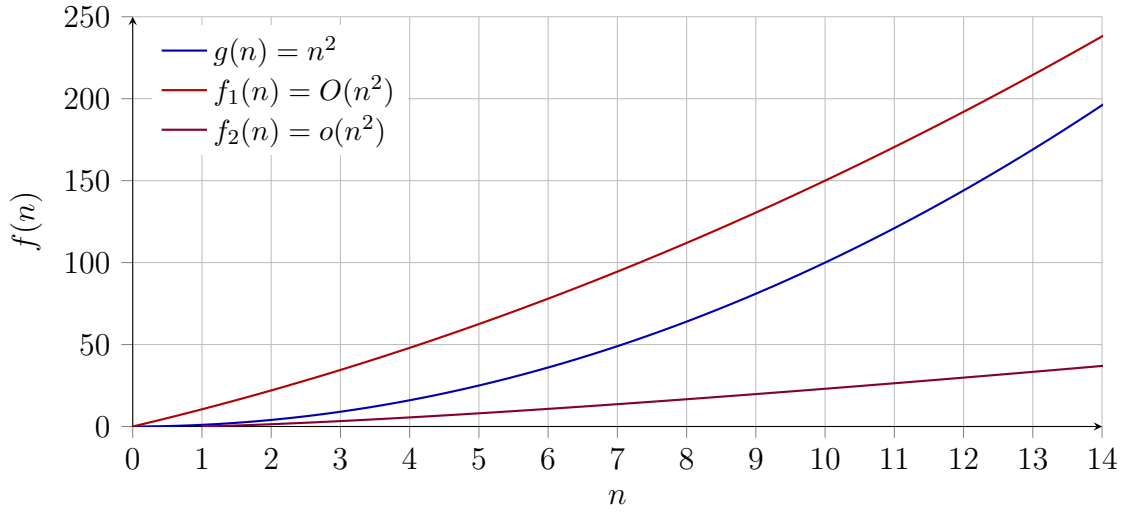

Figura 4: Confronto intuitivo tra $O(g(n))$ e $o(g(n))$: entrambe le funzioni restano sotto $g(n)$, ma $f_2(n)$ cresce molto più lentamente e il divario aumenta con n.

4 Oltre i casi limite

Le notazioni asintotiche descrivono il comportamento di un algoritmo per input molto grandi, ma nella realtà le cose possono essere più sfumate. Le costanti, i termini minori e la distribuzione degli input influenzano spesso le prestazioni più di quanto le formule suggeriscano. Un algoritmo $O(n)$ con una costante elevata può risultare più lento di uno $O(n \log n)$ per input di piccole dimensioni, e un algoritmo $O(n^2)$ può comportarsi sorprendentemente bene su dati particolarmente favorevoli.

Allo stesso modo, i casi limite descrivono solo scenari estremi: il caso peggiore (O) rappresenta un limite superiore, il migliore (Ω) un limite inferiore, ma nella pratica il comportamento medio (Θ) è spesso quello più rappresentativo.

In questa sezione analizzeremo due aspetti fondamentali per interpretare correttamente le notazioni asintotiche, ovvero l'importanza delle costanti e dei termini non dominanti, spesso determinanti nei casi reali e la differenza tra analisi teorica e comportamento pratico degli algoritmi.

4.1 Cosa accade davvero in pratica

Le notazioni asintotiche sono strumenti potenti per classificare gli algoritmi, ma è importante ricordare che esse rappresentano un modello *astratto* del comportamento di un programma. Dire che un algoritmo ha complessità $O(g(n))$ non significa che esegua esattamente $g(n)$ operazioni, ma solo che — per n sufficientemente grandi — il suo tempo di esecuzione non cresce più rapidamente di una funzione proporzionale a $g(n)$. Allo stesso modo, dire che un algoritmo è $\Omega(g(n))$ significa che non potrà mai essere più veloce di una funzione di quel tipo, almeno in media o nel caso migliore.

Nell'analisi degli algoritmi, è consuetudine distinguere tre situazioni principali:

- $O(g(n))$: descrive il **caso peggiore** (*worst case*), ossia il limite superiore del tempo di esecuzione. È la stima che garantisce che l'algoritmo non sarà mai più lento di quanto previsto.

- $\Omega(g(n))$: rappresenta il **caso migliore** (*best case*), cioè il limite inferiore del tempo necessario, raggiungibile solo in condizioni ottimali.
- $\Theta(g(n))$: indica che il tempo di esecuzione è compreso tra i due limiti precedenti, fornendo una descrizione **asintoticamente esatta** o, in molti casi, una stima del comportamento medio.

Tuttavia, nella pratica, gli algoritmi raramente operano nei loro casi estremi. Un algoritmo descritto come $O(n^2)$ potrebbe, in molti scenari reali, comportarsi quasi sempre come un algoritmo $O(n \log n)$, se le condizioni medie del problema lo favoriscono. Viceversa, un algoritmo con complessità $\Omega(n)$ nel caso migliore potrebbe raramente riuscire a raggiungere quel limite, perché le situazioni ottimali sono poco frequenti.

Per rendere più concreto questo concetto, consideriamo alcuni esempi:

Esempio 1: la ricerca lineare Nel caso peggiore, una ricerca lineare in un array di n elementi richiede di esaminare tutti gli elementi:

$$T_{\text{worst}}(n) = O(n).$$

Tuttavia, se l'elemento cercato si trova spesso nelle prime posizioni, il numero medio di confronti può essere molto più basso, ad esempio circa $n/2$. In tal caso, il comportamento medio si avvicina a:

$$T_{\text{avg}}(n) = \Theta(n/2) = \Theta(n),$$

ma il tempo effettivo percepito sarà spesso inferiore rispetto al limite teorico.

Esempio 2: l'ordinamento tramite QuickSort Il QuickSort ha una complessità nel caso peggiore di $O(n^2)$, che si verifica quando il pivot scelto in ogni passo è sempre il minimo o il massimo elemento (ad esempio, se l'array è già ordinato). Tuttavia, nella pratica, grazie alla scelta casuale del pivot o a tecniche di bilanciamento, l'algoritmo si comporta quasi sempre come un algoritmo $O(n \log n)$. Il suo caso medio è quindi molto più rappresentativo delle prestazioni reali:

$$T_{\text{avg}}(n) = \Theta(n \log n).$$

Esempio 3: la ricerca binaria La ricerca binaria ha caso migliore $\Omega(1)$ (quando l'elemento cercato è esattamente quello centrale) e caso peggiore $O(\log n)$. Nella pratica, tuttavia, ogni ricerca richiede un numero di passi che dipende in modo logaritmico da n, per cui il comportamento effettivo si avvicina costantemente al caso peggiore.

Questi esempi mostrano che le notazioni asintotiche non devono essere interpretate come previsioni puntuali, ma come **descrizioni qualitative del comportamento**. Esse indicano la tendenza di crescita, non il tempo effettivo di esecuzione.

Nella progettazione degli algoritmi è quindi fondamentale saper interpretare correttamente queste notazioni:

- il caso peggiore (O) ci garantisce una soglia di sicurezza, utile in contesti critici (ad esempio nei sistemi real-time);
- il caso migliore (Ω) indica il limite di ottimalità teorica, ma non sempre raggiungibile;

- il comportamento medio (Θ) è spesso quello più significativo nella valutazione pratica.

In conclusione, l'analisi asintotica fornisce un linguaggio essenziale per confrontare algoritmi in modo indipendente dall'hardware e dai dettagli implementativi, ma la valutazione empirica — misurando i tempi reali su input rappresentativi — resta indispensabile per capire davvero come un algoritmo si comporta "sul campo".

4.2 Quando le costanti contano

Nell'analisi asintotica degli algoritmi, è prassi comune trascurare i **fattori costanti** e i **termini di ordine inferiore**. Questo approccio consente di concentrarsi sulla crescita dominante della funzione di costo al crescere della dimensione dell'input n. Tuttavia, nella pratica quotidiana — soprattutto per input di piccole o medie dimensioni — queste costanti possono fare una differenza sostanziale.

Per comprendere questa distinzione, consideriamo due algoritmi che risolvono lo stesso problema:

$$T_1(n) = 100n, \qquad T_2(n) = 5n \log_2 n.$$

Dal punto di vista asintotico, $T_1(n) = O(n)$ e $T_2(n) = O(n \log n)$. Teoricamente, il primo algoritmo è più efficiente, poiché cresce linearmente, mentre il secondo include un fattore logaritmico. Tuttavia, il fattore costante 100 nel primo algoritmo può renderlo *più lento* del secondo per una vasta gamma di valori realistici di n.

Vediamolo con un esempio numerico. Per $n = 100$:

$$T_1(100) = 100 \times 100 = 10\,000, \qquad T_2(100) = 5 \times 100 \times \log_2 100 \approx 5 \times 100 \times 6.64 = 3\,320.$$

In questo caso, l'algoritmo con complessità asintoticamente peggiore ($O(n \log n)$) risulta in realtà più veloce, perché la costante moltiplicativa del primo algoritmo è molto più grande.

Solo quando n cresce oltre una certa soglia, la parte dominante della funzione (cioè la crescita asintotica) comincia a prevalere. Nel nostro esempio, se calcoliamo per quale valore di n i due tempi diventano uguali:

$$100n = 5n \log_2 n \quad \Rightarrow \quad \log_2 n = 20 \quad \Rightarrow \quad n \approx 10^6.$$

Ciò significa che soltanto per n superiori a un milione l'algoritmo lineare (T_1) diventa realmente più efficiente dell'altro.

Questo esempio mette in evidenza un punto importante: le notazioni asintotiche descrivono il *comportamento per n grandi*, ma non sempre riflettono le prestazioni reali su input di dimensioni moderate. In contesti applicativi (ad esempio nei sistemi embedded, nei giochi o nelle app mobili), il valore tipico di n può essere così piccolo che le costanti, l'implementazione o la cache del processore hanno un impatto molto più significativo della complessità teorica.

In sintesi, possiamo dire che:

- le notazioni asintotiche descrivono il comportamento *a lungo termine*;

- i fattori costanti e i termini minori influenzano le prestazioni *nel breve termine*;
- un algoritmo asintoticamente migliore può essere più lento di un altro per una vasta gamma di input reali.

La figura seguente illustra graficamente questo concetto: per valori piccoli di n, la curva di colore rosso (corrispondente a $100n$) è più alta di quella blu ($5n \log_2 n$); ma oltre un certo punto, l'andamento asintotico fa sì che quest'ultima la superi definitivamente.

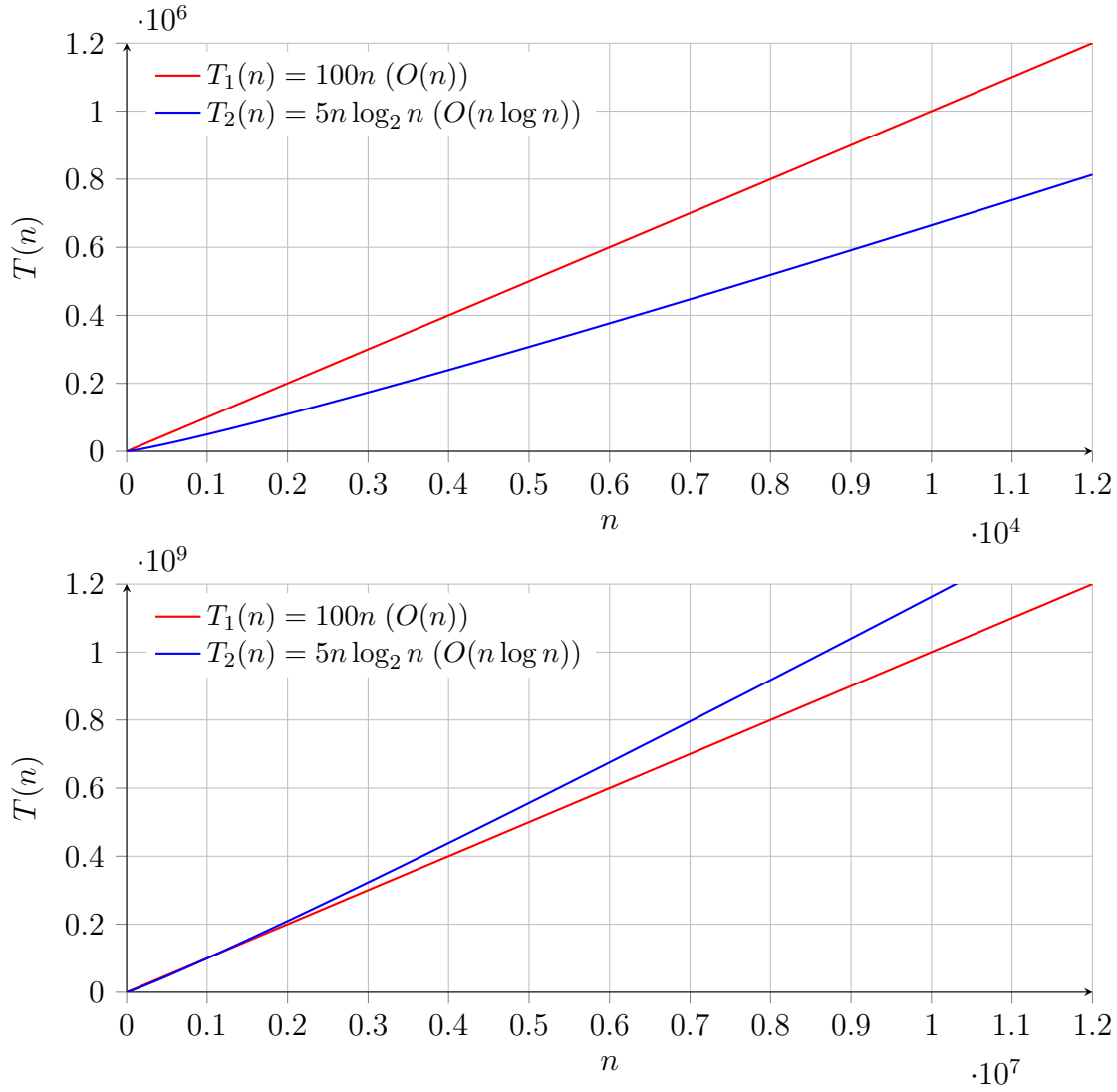

Figura 5: Esempio di confronto tra un algoritmo $O(n)$ con grande costante e uno $O(n \log n)$ con costante ridotta. Per n piccoli domina la costante, per n grandi domina la crescita asintotica.

Questo dimostra come l'analisi asintotica e quella empirica vadano sempre considerate in modo complementare: la prima fornisce una visione generale del comportamento dell'algoritmo per input molto grandi, la seconda rivela quale approccio risulta davvero più efficiente nelle condizioni operative reali.

5 Esercizi sul confronto e sull'analisi di funzioni

In questa sezione vengono proposti esercizi per consolidare la comprensione del comportamento asintotico delle funzioni di costo, del confronto tra ordini di crescita e dell'individuazione del termine dominante. Ogni esercizio richiede di analizzare la funzione o le funzioni proposte, determinando il loro andamento al crescere di n e la notazione asintotica appropriata.

1. Identificazione dell'ordine di crescita

Per ciascuna delle seguenti funzioni, individua il termine dominante e scrivi la notazione asintotica corretta (O, Ω o Θ).

1. $T(n) = 5n^2 + 7n + 2$
2. $T(n) = 3n^3 + 100n^2 + 7$
3. $T(n) = 20n + 50$
4. $T(n) = 10 \log n + 7$
5. $T(n) = n^4 + n^3 + n^2 + n$
6. $T(n) = 3n^2 \log n + 4n$
7. $T(n) = 2^n + n^5$
8. $T(n) = n! + 2^n$
9. $T(n) = 5n^2 + 2n \log n + 3$
10. $T(n) = n^{1/2} + \log n$

2. Confronto tra funzioni

Per ciascuna coppia di funzioni seguenti, stabilisci quale cresce più rapidamente al tendere di $n \to \infty$ e giustifica la risposta confrontando il loro rapporto o discutendone l'andamento.

1. $f(n) = n$ e $g(n) = \log n$
2. $f(n) = n^2$ e $g(n) = 2^n$
3. $f(n) = n \log n$ e $g(n) = n^{1.5}$
4. $f(n) = \log^2 n$ e $g(n) = n$
5. $f(n) = n!$ e $g(n) = 3^n$
6. $f(n) = n^3$ e $g(n) = n^3 + n^2$
7. $f(n) = n^{\log n}$ e $g(n) = 2^n$
8. $f(n) = 2^{\sqrt{n}}$ e $g(n) = n^{10}$
9. $f(n) = n^2$ e $g(n) = n^2 \log n$
10. $f(n) = \log(n!)$ e $g(n) = n \log n$

3. Classificazione in ordine di crescita

Ordina le funzioni seguenti dalla più lenta alla più rapida crescita asintotica, giustificando brevemente la tua scelta.

1. $\log n,\ n,\ n^2,\ 2^n,\ n!$
2. $\sqrt{n},\ n \log n,\ n^2,\ 3^n$
3. $n^2,\ n^2 \log n,\ n^3$
4. $2^{\log n},\ n,\ \log n$
5. $n^{1/2},\ n^{2/3},\ n,\ n^{1.5}$
6. $n,\ n^{\log n},\ 2^n$
7. $\log n,\ \log(\log n),\ n^{1/10}$
8. $n,\ 10n,\ n + 100$
9. $2^n,\ 3^n,\ n!$
10. $n^{1.01},\ n \log n,\ n^{0.9}$

4. Effetto delle costanti e dei termini minori

Mostra come i coefficienti moltiplicativi e i termini di ordine inferiore non modifichino la crescita asintotica della funzione.

1. Confronta $5n^2 + 10n + 3$ e n^2.
2. Confronta $100n$ e $0.5n$.
3. Confronta $3n^3 + 2n$ e $10n^3$.
4. Confronta $n^3 + n^2 + n$ e n^3.
5. Confronta $50n \log n + 10n$ e $n \log n$.
6. Confronta $10^6 n + 7$ e n.
7. Confronta $n + \log n$ e n.
8. Confronta $3 \log n + 100$ e $\log n$.
9. Confronta $5n^2 + 3n \log n + 7$ e n^2.
10. Confronta $2^n + 100n^5$ e 2^n.

5. Confronti non immediati

Determina quale tra le funzioni seguenti cresce più rapidamente e discuti il motivo, anche qualitativamente.

1. $n^{\log n}$ e 2^n
2. $\log(n!)$ e n
3. $n^{\sin n}$ e n
4. $n^{1+\sin n}$ e n
5. $\sqrt{n!}$ e 2^n
6. $n^{\log \log n}$ e n
7. n^n e 2^{n^2}
8. $n!$ e n^n
9. $2^{\sqrt{n}}$ e n^3
10. $\log n!$ e $n \log n$

6 Esercizi sulle notazioni asintotiche

In questa sezione vengono proposti esercizi per consolidare la comprensione e l'uso delle principali notazioni asintotiche (O, Ω, Θ, o, ω). Gli esercizi sono suddivisi per tipologia e affrontano diversi livelli di difficoltà, dalla comprensione intuitiva alla verifica formale.

1. Comprensione concettuale delle notazioni

Negli esercizi seguenti si richiede di spiegare, interpretare o confrontare le principali notazioni asintotiche. Rispondi in modo discorsivo, illustrando il significato delle relazioni e, ove utile, fornendo un esempio.

1. Spiega con parole tue che cosa significa scrivere $f(n) = O(g(n))$.
2. Descrivi la differenza tra $O(g(n))$ e $\Theta(g(n))$.
3. Qual è il significato pratico di dire che $T(n) = \Omega(n \log n)$?
4. Fornisci un esempio concreto di due funzioni $f(n)$ e $g(n)$ tali che $f(n) = o(g(n))$.
5. Spiega perché la notazione O rappresenta un limite superiore, mentre Ω rappresenta un limite inferiore.
6. Descrivi in quali contesti è utile usare $\Theta(g(n))$ invece di $O(g(n))$.
7. Se $f(n) = O(n^2)$, possiamo dire che $f(n) = O(n^3)$? Perché sì o perché no?
8. Che relazione c'è tra $o(g(n))$ e $O(g(n))$?

9. Fornisci un esempio di funzione che appartiene a $O(n^2)$ ma non a $o(n^2)$.
10. Spiega in quali casi $f(n) = \omega(g(n))$ e $g(n) = o(f(n))$ possono essere considerate relazioni equivalenti.

2. Verifica formale con la definizione ε–n_0

Applica la definizione formale delle notazioni asintotiche. Individua, ove richiesto, le costanti c, c_1, c_2 e la soglia n_0 che verificano le condizioni specificate.

1. Dimostra che $3n + 5 = O(n)$ specificando c e n_0.
2. Mostra che $2n^2 + 7n + 10 = O(n^2)$.
3. Dimostra che $5n^3 - 2n^2 = \Omega(n^3)$.
4. Verifica che $\log n = o(n)$.
5. Trova i valori di c_1, c_2, n_0 tali che $5n^2 + 4n \in \Theta(n^2)$.
6. Dimostra formalmente che $3^n = \omega(n^k)$ per ogni $k > 0$.
7. Verifica, con la definizione di limite, che $n = o(n^2)$.
8. Dimostra che $2^n = \Omega(n^3)$ ma non $O(n^3)$.
9. Verifica che $5n + 3\log n = O(n)$.
10. Dimostra che $n! = \omega(2^n)$.

3. Determinazione della notazione corretta

Per ciascuna delle seguenti coppie o singole funzioni, determina quale relazione asintotica le lega (O, Ω, Θ, o, ω). Motiva sempre la risposta indicando il comportamento per $n \to \infty$.

1. $f(n) = n^2$, $g(n) = 3n^2 + 2n$
2. $f(n) = n\log n$, $g(n) = n^2$
3. $f(n) = n^3$, $g(n) = 2^n$
4. $f(n) = \log n$, $g(n) = 1$
5. $f(n) = n^2 + \log n$, $g(n) = n^2$
6. $f(n) = n^2$, $g(n) = n^2 \log n$
7. $f(n) = \log(n!)$, $g(n) = n\log n$
8. $f(n) = n^n$, $g(n) = 2^{n^2}$
9. $f(n) = \sqrt{n}$, $g(n) = n$
10. $f(n) = n\log n$, $g(n) = n^{1.01}$

4. Interpretazione algoritmica

Negli esercizi seguenti, collega la notazione asintotica a situazioni reali relative all'analisi di algoritmi. Rispondi in modo discorsivo, ragionando sul significato pratico della crescita delle funzioni.

1. Due algoritmi hanno complessità $O(n^2)$ e $O(n \log n)$. Per quale valore di n il secondo diventa più efficiente?
2. Se un algoritmo ha tempo $O(2^n)$, in quali contesti pratici potrebbe comunque essere utilizzabile?
3. Spiega perché un algoritmo lineare è preferibile a uno quadratico per input molto grandi, anche se può risultare più lento per piccoli valori di n.
4. Fornisci un esempio di algoritmo con caso migliore $\Omega(n)$, caso peggiore $O(n^2)$ e caso medio $\Theta(n \log n)$.
5. Un algoritmo ha complessità $O(n^2)$. Cosa significa in termini di tempo richiesto se si raddoppia la dimensione dell'input?
6. Spiega perché un algoritmo $O(n)$ è più scalabile di uno $O(n^2)$.
7. Se un algoritmo impiega $\Theta(n^3)$ passi, quanti ne richiederà approssimativamente con un input 10 volte più grande?
8. Descrivi un caso in cui la notazione $o(g(n))$ è utile per stimare un miglioramento asintotico.
9. Fornisci un esempio di due algoritmi con la stessa complessità asintotica ma tempi pratici molto diversi.
10. Un algoritmo divide l'input in due e richiede tempo costante per combinare le soluzioni. Qual è la sua complessità asintotica in notazione O?

5. Esercizi di sintesi e confronto tra notazioni

In questi esercizi, le diverse notazioni vengono messe a confronto per analizzarne relazioni, implicazioni e significato formale.

1. Indica l'ordine corretto tra le seguenti notazioni: $\omega(g(n))$, $\Omega(g(n))$, $\Theta(g(n))$, $O(g(n))$, $o(g(n))$.
2. Se $f(n) = \Theta(g(n))$, cosa puoi dire su $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$?
3. Se $f(n) = o(g(n))$, è vero che $f(n) = O(g(n))$? Spiega.
4. Se $f(n) = O(g(n))$ e $g(n) = O(h(n))$, cosa puoi dire di $f(n)$ rispetto a $h(n)$?
5. Se $f(n) = O(g(n))$ e $g(n) = \Omega(f(n))$, quale notazione riassume meglio la relazione?
6. Fornisci un esempio in cui $f(n) = \Omega(g(n))$ ma non $f(n) = \Theta(g(n))$.

7. Spiega la differenza tra $f(n) = o(g(n))$ e $f(n) = O(g(n))$ in termini di limite del rapporto $\frac{f(n)}{g(n)}$.

8. Se $f(n) = n$ e $g(n) = n + \sin n$, quale relazione asintotica esiste tra le due funzioni?

9. Mostra che se $f(n) = \Theta(g(n))$ e $g(n) = \Theta(h(n))$, allora $f(n) = \Theta(h(n))$.

10. Determina per quali notazioni il limite è "raggiunto" e per quali no.