Algoritmi e Strutture Dati[1]

Corso di Laurea in Ingegneria dell'Informazione
Sapienza Università di Roma – sede di Latina

Fabio Patrizi

Dipartimento di Ingegneria Informatica, Automatica e Gestionale (DIAG)
Sapienza Università di Roma – Italy
www.dis.uniroma1.it/~patrizi
patrizi@dis.uniroma1.it

[1]Slides prodotte a partire dal materiale didattico fornito con il testo *Demetrescu, Finocchi, Italiano: Algoritmi e strutture dati, McGraw-Hill, seconda edizione*.

Modelli di Calcolo e Metodologie di Analisi

Modelli di Calcolo

Il modello di calcolo che adottiamo è detto a costo uniforme:

- operazioni a costo costante, indipendente dalla dimensione degli operandi
- trascura il fatto che il costo delle operazioni può dipendere dalla dimensione degli operandi

Modello a costo logaritmico:

- costo delle operazioni dipendente dalla dimensione degli operandi (logaritmo del valore)
- più accurato ma di maggiore complessità

Entrambi:

- forniscono una stima dell'*andamento del costo* al variare della dimensione dell'input
- permettono il confronto tra algoritmi (nello stesso modello)

La Notazione Asintotica

Le approssimazioni adottate nel modello di costo rendono trascurabili le differenze di costo dovute a costanti additive o moltiplicative

Ci interessiamo pertanto all'andamento delle funzioni di costo, non ai valori specifici assunti per particolari dimensioni dell'input

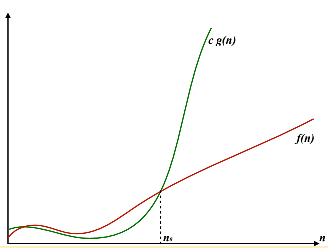
La notazione asintotica è lo strumento matematico che permette il confronto tra gli andamenti di due funzioni al crescere della dimensione dell'input

La Notazione O-grande

Definition

Scriviamo $f(n) = \mathcal{O}(g(n))$ ($f(n)$ è "O grande di $g(n)$"), se esistono due costanti $c > 0$ ed $n_0 \geq 0$ tali che, per ogni $n \geq n_0$,

$$f(n) \leq c \cdot g(n)$$

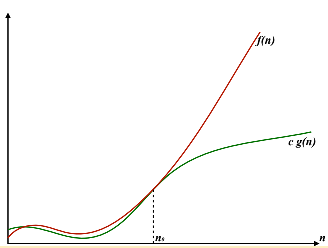


La Notazione Ω-grande

Definition

Scriviamo $f(n) = \Omega(g(n))$ ($f(n)$ è "Ω grande di $g(n)$"), se esistono due costanti $c > 0$ ed $n_0 \geq 0$ tali che, per ogni $n \geq n_0$,

$$f(n) \geq c \cdot g(n)$$

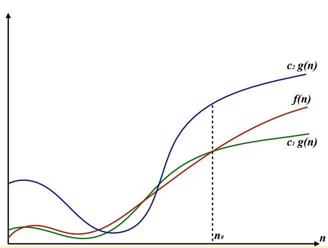


La Notazione Θ-grande

Definition

Scriviamo $f(n) = \Theta(g(n))$ ($f(n)$ è "Θ grande di $g(n)$"), se esistono tre costanti $c_1, c_2 > 0$ ed $n_0 \geq 0$ tali che, per ogni $n \geq n_0$,

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$



La Notazione Asintotica

Risultati notevoli

Theorem

- *Se* $f(n) = \mathcal{O}(g(n))$ *allora* $g(n) = \Omega(f(n))$
- *Se* $f(n) = \Omega(g(n))$ *allora* $g(n) = \mathcal{O}(f(n))$

Theorem

$f(n) = \Theta(g(n))$ *se e solo se* $f(n) = \mathcal{O}(g(n))$ *e* $f(n) = \Omega(g(n))$

Theorem

$f(n) = \Theta(g(n))$ *se e solo se* $g(n) = \Theta(f(n))$

Theorem

Per ogni polinomio $p(n) = a_0 + a_1 n + \cdots + a_m n^m$, *t.c.* $a_m > 0$, *si ha:* $p(n) = \Theta(n^m)$.

Proof.

Consideriamo solo $n > 0$

Per $p(n) = \mathcal{O}(n^m)$: $p(n) \leq cn^m$, dove $c = \sum_{i=0}^{m} |a_i| > 0$

Per $p(n) = \Omega(n^m)$: $p(n) \geq a_m n^m - Nn^{m-1}$, dove $N = \sum_{i=0}^{m-1} |a_i| > 0$

Cerchiamo ora $c > 0$ ed $n_0 \geq 0$ t.c. $a_m n^m - Nn^{m-1} \geq cn^m$, per $n \geq n_0$:
$a_m n^m - Nn^{m-1} \geq cn^m \Leftrightarrow (a_m - c)n \geq N \Leftrightarrow n \geq \frac{N}{a_m - c} = n_0$, con $c < a_m$

Quindi, per $0 < c < a_m$ ed $n \geq n_0$, abbiamo $p(n) \geq cn^m$, ovvero $p(n) = \Omega(n^m)$. $\square$

Esempi

Example

Si consideri la funzione $f(n) = 3n^2 + 10$
Abbiamo:

- $f(n) = \mathcal{O}(n^2)$ (es., $c = 4$ e $n_0 = \sqrt{10}$)
- $f(n) = \Omega(n^2)$ (es., $c = 1$ e $n_0 = 0$)
- $f(n) = \Theta(n^2)$ (dai punti sopra e dal teorema precedente)
- $f(n) = \mathcal{O}(n^3)$ ma $f(n) \neq \Theta(n^3)$ (perché $f(n) \neq \Omega(n^3)$)

La Notazione Asintotica

Definition (Costo temporale di un algoritmo (limitazione superiore))

Un algoritmo ha costo temporale $\mathcal{O}(f(n))$ se per ogni istanza di dimensione n, l'algoritmo risolve tale istanza con un numero $N(n)$ di operazioni elementari t.c. $N(n) = \mathcal{O}(f(n))$.

Definition (Costo temporale di un algoritmo (limitazione inferiore))

Un algoritmo ha costo temporale $\Omega(f(n))$ se per ogni istanza di dimensione n, l'algoritmo risolve tale istanza con un numero $N(n)$ di operazioni elementari t.c. $N(n) = \Omega(f(n))$.

Analoghe definizioni valgono per lo spazio

Metodi di Analisi

Caso Peggiore, Migliore, Medio

- Misuriamo le risorse di calcolo usate da un algoritmo in funzione della dimensione n dell'input (ovvero dell'istanza del problema da risolvere)
- Input di pari dimensioni possono richiedere quantità di risorse diverse
- Vogliamo una misura che non dipenda dalla configurazione dell'input, ma solo dalla sua dimensione

Caso Peggiore, Migliore, Medio

Example (Ricerca Sequenziale)

$Algoritmo\ RicercaSequenziale(lista\ L, elem\ x) \rightarrow Boolean$

for all $(y \in L)$ **do**
    if $(y = x)$ **then return** $true$
return $false$

Se forniamo in input una lista di n elementi, quanti confronti eseguirà l'algoritmo prima di terminare?

- Chiaramente dipende dalla posizione in lista dell'elemento cercato
- Ad esempio, se l'elemento si trova nella posizione favorevole (la prima), basterà un solo confronto
- Come eliminare questa dipendenza dalla configurazione dell'input?

Caso Peggiore, Migliore, Medio

Tre approcci classici all'analisi:

- *Caso peggiore*: quanto impiega l'algoritmo su un'istanza di dimensione n se la configurazione è la più sfavorevole?
- *Caso migliore*: sulla configurazione più favorevole?
- *Caso medio*: mediamente su tutte le istanze di dimensione n?

Caso Peggiore

Definition

- Indichiamo con $tempo(I)$ il tempo impiegato dall'algoritmo per risolvere la generica istanza I
- Indichiamo con $Istanze_n$ l'insieme delle istanze di dimensione n
- Definiamo la misura di *costo temporale nel caso peggiore*:

$$T_{worst}(n) = \max_{\{Istanze_n\}} \{tempo(I)\}$$

- $T_{worst}(n)$ ci dice quanto tempo impiega l'algoritmo sulle istanze più sfavorevoli di dimensione n
- Fornisce un *upper bound* ("peggio di così non può andare")
- Offre garanzia sulle prestazioni: se $T_{worst}(n)$ è accettabile, lo sarà anche $tempo(I)$ per qualsiasi istanza di dimensione n

Considereremo il caso peggiore per l'analisi degli algoritmi

Caso Migliore

Definition

- Definiamo la misura di *costo temporale nel caso migliore*:

$$T_{best}(n) = \min_{\{Istanze_n\}} \{tempo(I)\}$$

- $T_{best}(n)$ ci dice quanto tempo impiega l'algoritmo sulle istanze più favorevoli di dimensione n
- Fornisce un *lower bound* ("meglio di così non può andare")
- Non offre nessuna garanzia sulle prestazioni:
  - Se $T_{best}(n)$ non è accettabile, sicuramente non lo sarà $tempo(I)$ per nessuna istanza di dimensione n
  - Viceversa, se $T_{best}(n)$ è accettabile, non sappiamo se lo sarà $tempo(I)$ (per I di dimensione n)

Caso Medio

Definition

- Indichiamo con $P_n(I)$ la probabilità che un'istanza I di dimensione n sia fornita in input
- Definiamo la misura di *costo temporale nel caso medio*:

$$T_{avg}(n) = \sum_{\{I \in Istanze_n\}} P_n(I) \cdot tempo(I)$$

- $T_{avg}(n)$ ci dice quanto è il tempo atteso impiegato dall'algoritmo su una generica istanza di dimensione n
- Fornisce indicazioni circa il *tempo medio* d'esecuzione
- Informativa (e utile) ma non sempre applicabile: occorre conoscere la distribuzione di probabilità P_n

Esempio: Ricerca Sequenziale

$Algoritmo\ RicercaSequenziale(lista\ L, elem\ x) \rightarrow Boolean$

for all $(y \in L)$ **do**

  if $(y == x)$ **then return** $true$

return $false$

- Caso migliore: elemento in prima posizione, $T_{best}(n) = \mathcal{O}(1)$
- Caso peggiore: elemento assente, $T_{worst}(n) = \mathcal{O}(n)$
- Caso medio (per elemento in posizione i con probabilità $1/n$):

$$T_{avg}(n) =$$

$$\mathcal{O}(\sum_{1 \leq i \leq n} P(pos(x) = i) \cdot tempo(I)) = \mathcal{O}(\sum_{1 \leq i \leq n} 1/n \cdot i) = \mathcal{O}((n+1)/2)$$

Esempio: Ricerca Binaria

$Algoritmo\ RicercaBinaria(listaOrdinata\ L, elem\ x) \rightarrow Boolean$

$a \leftarrow 1$
$b \leftarrow$ dimensione di L
while $(L[(a+b)/2] \neq x)$ **do**
  $i \leftarrow (a+b)/2$
  if $(L[i] > x)$ **then** $b \leftarrow i-1$
  else $a \leftarrow i+1$
  if $(a > b)$ **then return** $false$
return $true$

- Caso migliore: elemento in posizione $(a+b)/2$, $T_{best}(n) = \mathcal{O}(1)$
- Caso peggiore: elemento assente, $T_{worst}(n) = \mathcal{O}(\log n)$
- Caso medio (per elemento in posizione i con probabilità $1/n$):

$$T_{avg}(n) = \mathcal{O}(\log n - 1 + 1/n)$$

Analisi Asintotica: Spazio

Tutte le definizioni, i risultati e le osservazioni viste fin qui trovano analoga applicazione al caso dell'analisi di costo spaziale.

Analisi di Algoritmi Ricorsivi

Analisi di algoritmi ricorsivi

- L'analisi di algoritmi ricorsivi è in generale più complessa del caso iterativo
- Occorre impostare e risolvere un'*equazione di ricorrenza*

Equazioni di Ricorrenza

Definition (Equazione di ricorrenza)

Un'*equazione di ricorrenza* è un'equazione che lega il valore di una funzione $f : \mathbb{N} \to \mathbb{N}$ sull'input n al valore della funzione f su input minori di n:

$$f(n) = F[f(n-1), \ldots, f(1)]$$

Risolvere un'equazione di ricorrenza significa *trovare una funzione* $f(n)$ *che la soddisfa*

Come risolverla? Diversi metodi...

Analisi dell'Albero della Ricorsione

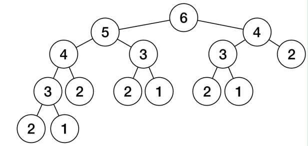
Intuizione

Costruire l'albero della ricorsione (in funzione di n) e considerare: per ciascun nodo foglia, il costo del passo base; per ciascun nodo interno, il costo del passo ricorsivo. La somma dei costi dei nodi fornisce il costo dell'esecuzione

Example (Equazione di ricorrenza di `fibonacci_2`)

$$T(n) = \begin{cases} 2, \text{ se } n \leq 2 \\ 3 + T(n-1) + T(n-2), \text{ se } n > 2 \end{cases}$$

Albero della ricorsione per $n = 6$:



Analisi dell'Albero della Ricorsione

Example (Costo d'esecuzione di `fibonacci_2`)

- Assegniamo a ciascun nodo interno costo 3 (v. eq. ricorrenza)
- Assegniamo a ciascun nodo foglia costo 2 (v. eq. ricorrenza)

Abbiamo quindi: $T(n) = 3N_{nodi\_interni} + 2N_{nodi\_foglia}$

Per un albero binario: $N_{nodi\_interni} = N_{nodi\_foglia} - 1$

Quindi: $T(n) = 3(N_{nodi\_foglia} - 1) + 2N_{nodi\_foglia} = 5N_{nodi\_foglia} - 3$

Siccome $N_{nodi\_foglia} = F_n$ e $F_n \approx \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n)$, otteniamo:

$$T(n) \approx \frac{5}{\sqrt{5}}(\phi^n - \hat{\phi}^n) - 3$$

Metodo dell'Iterazione

Intuizione

Consideriamo le chiamate ricorsive passo-passo, ottenendo una sommatoria dipendente dalla dimensione n dell'input iniziale e dal numero k di passi.

Example

$$\text{Equazione: } T(n) = \begin{cases} 1, \text{ se n} = 1 \\ c + T(n/2), \text{ altrimenti} \end{cases}$$

Per k passi abbiamo:

$$T(n) = c + T(n/2) = 2c + T(n/4) = \cdots = (kc + T(n/2^k))$$

Procediamo fino a $T(1)$, ovvero $n/2^k = 1$, cioè $k = \log_2(n)$, ottenendo:

$$T(n) = c\log_2(n) + T(1) = \mathcal{O}(\log n)$$

Metodo della Sostituzione

Intuizione

Scegliere un candidato (ragionevole) per la soluzione e dimostrare la correttezza della scelta tramite *induzione matematica*

Induzione Matematica

Induzione Matematica

Possiamo dimostrare che una proprietà P vale per ogni intero $n \in [n_0, \infty)$, dimostrando che:

(Passo Base) P vale per un intero n_0

(Passo Induttivo) Se P vale per tutti i valori $[n_0, n-1]$ (ipotesi induttiva), allora P vale anche per n

Example (Formula di Gauss per la somma dei primi n interi)

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

(Passo Base) $\sum_{i=1}^{1} i = 1 = 1(1+1)/2$

(Passo Induttivo) Assumendo che $\sum_{i=1}^{n-1} i = n(n-1)/2$, abbiamo:
$\sum_{i=1}^{n} i = n(n-1)/2 + n = n(n+1)/2$

Metodo della Sostituzione

Example

$$\text{Equazione: } T(n) = \begin{cases} 1, \text{ se } n = 1 \\ T(\lfloor n/2 \rfloor) + n, \text{ altrimenti} \end{cases}$$

Ipotizziamo che $T(n) \leq cn$, per qualche c, ovvero che $T(n) = \mathcal{O}(n)$

Dimostriamo per induzione che l'ipotesi è corretta:

- Caso base: $T(1) = 1 \leq c$, per ogni $c \geq 1$
- Passo induttivo: $T(n) = T(\lfloor n/2 \rfloor) + n \leq cn/2 + n = n(c/2 + 1)$
  Abbiamo che $n(c/2 + 1) \leq cn \Leftrightarrow c/2 + 1 \leq c \Leftrightarrow c \geq 2$

Abbiamo dimostrato che, per $c \geq 2$, $T(n) \leq cn$

La risoluzione per sostituzione richiede intuito e colpo d'occhio, qualità che si ottengono con un po' d'esercizio.

Teorema Master

o Teorema Fondamentale delle Ricorrenze

Theorem (Teorema Master)

Data la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1, \textit{ se } n = 1 \\ a \cdot T(n/b) + f(n), \textit{ se } n > 1 \end{cases}$$

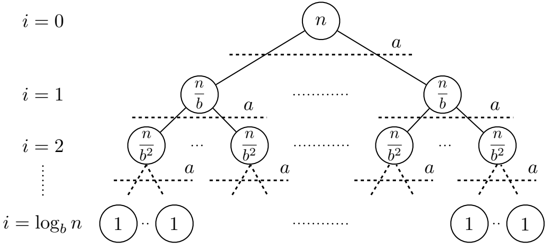
Abbiamo che:

1. $T(n) = \Theta(n^{\log_b a})$, *se* $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$, *per* $\epsilon > 0$;
2. $T(n) = \Theta(n^{\log_b a} \log n)$, *se* $f(n) = \Theta(n^{\log_b a})$;
3. $T(n) = \Theta(f(n))$, *se* $f(n) = \Omega(n^{\log_b a + \epsilon})$, *per* $\epsilon > 0$ *e* $a \cdot f(n/b) \leq c \cdot f(n)$, *per* $c < 1$ *ed* n *sufficientemente grande.*

Teorema Master

Dimostrazione: albero della ricorsione

(Assumiamo, per semplicità, che n sia una potenza di b)
Analizziamo l'albero della ricorsione per $T(n) = a \cdot T(n/b) + f(n)$



Abbiamo: $T(n) = \sum_{i=0}^{\log_b n} a^i \cdot f\left(\frac{n}{b^i}\right)$

- Per conoscere $T(n)$ occorre conoscere il valore della sommatoria
- Ci accontentiamo dell'andamento asintotico

Teorema Master

Dimostrazione: caso 1

Se $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$, per il generico termine della sommatoria, abbiamo:
$a^i \cdot f\left(\frac{n}{b^i}\right) = \mathcal{O}\left(a^i \cdot \left(\frac{n}{b^i}\right)^{\log_b a - \epsilon}\right) = \mathcal{O}\left(a^i \cdot \frac{n^{\log_b a - \epsilon}}{b^{i(\log_b a - \epsilon)}}\right) = \mathcal{O}\left(a^i \cdot \frac{n^{\log_b a - \epsilon}}{a^i b^{-i\epsilon}}\right) =$
$\mathcal{O}\left(n^{\log_b a - \epsilon} \cdot b^{i\epsilon}\right)$

Quindi: $T(n) = \sum_{i=0}^{\log_b n} a^i \cdot f\left(\frac{n}{b^i}\right) = \sum_{i=0}^{\log_b n} \mathcal{O}\left(n^{\log_b a - \epsilon} \cdot b^{i\epsilon}\right) =$
$\mathcal{O}\left(n^{\log_b a - \epsilon} \cdot \sum_{i=0}^{\log_b n} (b^\epsilon)^i\right) = \mathcal{O}\left(n^{\log_b a - \epsilon} \cdot \frac{(b^\epsilon)^{log_b n + 1} - 1}{b^\epsilon - 1}\right) =$
$\mathcal{O}\left(n^{\log_b a - \epsilon} \cdot \frac{b^\epsilon n^\epsilon - 1}{b^\epsilon - 1}\right) = \mathcal{O}\left(n^{\log_b a - \epsilon} \cdot n^\epsilon\right) = \mathcal{O}\left(n^{\log_b a}\right)$

(Si ricordi che: $\sum_{i=0}^{k} x^i = \frac{x^{k+1} - 1}{x - 1}$)

Inoltre, dall'ultimo livello dell'albero:
$T(n) = \Omega(a^{\log_b n}) = \Omega(a^{\log_b a \log_a n}) = \Omega(n^{\log_b a})$

Pertanto:

$$T(n) = \Theta(n^{\log_b a})$$

Teorema Master

Dimostrazione: caso 2

Se $f(n) = \Theta(n^{\log_b a})$, per il generico termine della sommatoria, abbiamo:
$a^i \cdot f\left(\frac{n}{b^i}\right) = \Theta\left(a^i \cdot \left(\frac{n}{b^i}\right)^{\log_b a}\right) = \Theta\left(a^i \cdot \frac{n^{\log_b a}}{b^{i \log_b a}}\right) = \Theta\left(a^i \cdot \frac{n^{\log_b a}}{a^i}\right) =$
$\Theta\left(n^{\log_b a}\right)$

Quindi: $T(n) = \sum_{i=0}^{\log_b n} a^i \cdot f\left(\frac{n}{b^i}\right) = \sum_{i=0}^{\log_b n} \Theta\left(n^{\log_b a}\right) =$
$\Theta\left(n^{\log_b a} \log_b n\right) = \Theta\left(n^{\log_b a} \log n\right)$

Teorema Master

Dimostrazione: caso 3

Se $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$, per il generico termine della sommatoria abbiamo:
$a^i \cdot f\left(\frac{n}{b^i}\right) = a^{i-1} \cdot a \cdot f\left(\frac{\frac{n}{b^{i-1}}}{b}\right) \leq c \cdot a^{i-1} \cdot f\left(\frac{n}{b^{i-1}}\right) \leq \cdots \leq c^i \cdot f(n)$

Quindi: $T(n) = \sum_{i=0}^{\log_b n} a^i \cdot f\left(\frac{n}{b^i}\right) \leq \sum_{i=0}^{\log_b n} c^i \cdot f(n) \leq f(n) \cdot \sum_{i=0}^{\infty} c^i =$
$f(n) \cdot \frac{1}{1-c} = \mathcal{O}(f(n))$

Inoltre, è immediato vedere che: $T(n) = \Omega(f(n))$

Pertanto: $T(n) = \Theta(f(n))$

Osservazione: l'ipotesi $f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$ non è utilizzata, in quanto implicata da $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$:
$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n) \Rightarrow f(n) \geq \frac{a}{c} \cdot f\left(\frac{n}{b}\right) \Rightarrow \cdots \Rightarrow f(n) \geq \left(\frac{a}{c}\right)^i \cdot f\left(\frac{n}{b^i}\right) \Rightarrow \cdots \Rightarrow$
$f(n) \geq \left(\frac{a}{c}\right)^{\log_b n} \cdot f(1) \Rightarrow f(n) = \Omega\left(\left(\frac{a}{c}\right)^{\log_b n}\right) = \Omega\left(\left(\frac{a}{c}\right)^{\log_{\frac{a}{c}} n \cdot \log_b \frac{a}{c}}\right) =$
$\Omega\left(n^{\log_b a+\log_b \frac{1}{c}}\right) = \Omega\left(n^{\log_b a+\epsilon}\right)$ (con $\epsilon = \log_b \frac{1}{c} > 0$, essendo $c < 1$)

Consideriamo la relazione di ricorrenza: $T(n) = 9T(n/3) + n$

Applichiamo il Teorema Master:

- $a = 9$
- $b = 3$
- $f(n) = n = \Theta(n)$

Poiché $n^{\log_b a} = n^2$, siamo nel caso 1 del teorema: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$, con $\epsilon = 1$

Pertanto:

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$$

Relazione di ricorrenza di $RicercaBinaria$: $T(n) = T(n/2) + c$

Applichiamo il Teorema Master:

- $a = 1$
- $b = 2$
- $f(n) = c = \Theta(1)$

Poiché $n^{\log_b a} = 1$, siamo nel caso 2 del teorema: $f(n) = \Theta(n^{\log_b a})$

Pertanto:

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

Relazione di ricorrenza: $T(n) = 3T(n/9) + n$

Applichiamo il Teorema Master:

- $a = 3$
- $b = 9$
- $f(n) = n$

Si ha $n^{\log_b a} = n^{1/2}$, quindi: $f(n) = \Omega(n^{\log_b a + \epsilon})$, per $\epsilon = 1/2$

Dimostriamo che, per $c < 1$, si ha $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$:

$$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n) \Leftrightarrow 3n/9 \leq cn \Leftrightarrow c \geq 1/3$$

Pertanto siamo nel caso 3 del teorema:

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