Problemi

Un **problema** specifica in termini generali una *relazione* che intercorrere tra dei dati di ingresso (input) e dei dati di uscita (output).

Problema del minimo Trovare l'elemento minimo in un insieme di n elementi {x1, x2 ,…,xn}

Problema della ricerca di un elemento in una sequenza di elementi ordinati e distinti

L'importante è che sugli elementi sia definito un ordinamento

Algoritmi

Un **algoritmo** è una *procedura computazionale* che prende un valore in **ingresso** (**input**) e restituisce un valore in **uscita** (**output**).

Un algoritmo è **corretto** rispetto ad un problema oppure **risolve** un problema se, per ogni ingresso del problema, l'algoritmo termina fornendo l'uscita del problema.

Programmi

Con il termine **programma** si intende la trascrizione (implementazione) dell'algoritmo in un linguaggio formale di programmazione (Pascal, C, Java, ...).

La differenza tra algoritmo e programma è la seguente: un algoritmo è una procedura computazionale intelliggibile dall'uomo, un programma è una procedura computazionale comprensibile dalla macchina.

Ne deriva che il linguaggio in cui scriviamo gli algoritmi è più astratto del linguaggio di programmazione.

Decidibilità

Un problema è **decidibile** se esiste almeno un algoritmo che lo risolve. Esistono problemi indecidibili? Quanti?

In seguito: esistono dei problemi indecidibili

Algoritmi

1) Algoritmo per il Problema del minimo: Per trovare il minimo di un insieme confronta ogni elemento con tutti gli altri, per ogni elemento.

2) Algoritmo per il Problema della ricerca di un elemento in una sequenza ordinata: Per trovare un valore v nella sequenza ordinata confronta v con tutti gli elementi della sequenza (se presente restituisci la posizione corrispondente, altrimenti -1)

NO ALGORITMI "FURBI". Nel caso peggiore
Soluzione 1) richiede circa n(n-1) confronti
Soluzione 2) richiede n confronti

Risoluzione di Problemi

Scelta del linguaggio formale per la specifica degli algoritmi

Valutazione della bontà (efficienza) di algoritmi (assumendo la loro correttezza)

Complessità computazionale

In generale, esistono più algoritmi diversi che risolvono un problema decidibile. Quale scegliere?

Complessità computazionale

In generale, esistono più algoritmi diversi che risolvono un problema decidibile. Quale scegliere?

Quello che ha costo o **complessità** inferiore. La complessità computazionale di un algoritmo è la **quantità di risorse** che l'algoritmo richiede per terminare.

Complessità computazionale

In generale, esistono più algoritmi diversi che risolvono un problema decidibile. Quale scegliere?

Quello che ha costo o **complessità** inferiore. La complessità computazionale di un algoritmo è la **quantità di risorse** che l'algoritmo richiede per terminare.

Quali risorse sono significative per il calcolo della complessità?

Complessità computazionale

La complessità di un dato algoritmo è funzione della **dimensione** (e della **forma**) dei dati in ingresso.

Prendiamo il problema di ordinamento degli elementi di un array. Ci aspettiamo che più è lunga la sequenza, più risorse l'algoritmo impieghi per terminare.

Complessità computazionale

Bontà o efficienza in termini di consumo di risorse-
Tempo e Memoria (in funzione della dimensione dell'input)

TEMPO: # di **azioni elementari** in funzione delle dimensioni dell'input
(tempo richiesto per eseguire le azioni elementari)
SPAZIO: necessario per memorizzare e manipolare i dati

La risorsa più importante è il TEMPO (perché?)

Es. Primalità Input n intero positivo
Algoritmo provare la divisione per tutti i numeri minori o uguali a n.
Dimensione dell'input (numero di bit per la memorizzazione $x=\log_2 n$)
Complessità: esponenziale $n=2^x$

Azioni Elementari

danno un risultato certo, unico e ripetibile

Livello di dettaglio: nel caso di linguaggio macchina sono necessari moltissimi dettagli;
Una buona misura dell'efficienza di un algoritmo deve prescindere dal calcolatore utilizzato;
Occorre una misura "astratta" che tenga conto del "metodo di risoluzione" con cui l'algoritmo effettua la computazione

Calcolo dell'efficienza: un esempio

Trovare l'elemento minimo in un insieme di n numeri {x1, x2, …,xn}

Nuova Soluzione 1a)

1. considero x1 come candidato ad essere il minimo
2. confronto il candidato con tutti gli altri elementi e se trovo un elemento più piccolo lo faccio diventare il nuovo candidato
3. al termine dei confronti, il candidato corrente è sicuramente il minimo

Complessivamente sono stati eseguiti n confronti

L'efficienza dell'algoritmo 1a) è quindi direttamente proporzionale alla dimensione dell'input

L'efficienza dell'algoritmo 1) era invece circa il quadrato della dimensione dell'input

Misurare il tempo di calcolo

La complessità dei problemi da risolvere dipende dalla dimensione dei dati in ingresso
Il tempo di calcolo si può quindi esprimere come: # azioni elementari in funzione della dimensione dei dati in ingresso

Le operazioni elementari sono quelle:

- aritmetiche logiche
- di confronto
- di assegnamento

Solitamente, il numero di operazioni considerate è valutato nel caso peggiore, ossia nel caso di dati in ingresso più sfavorevoli tra tutti quelli di dimensione n.

ESEMPIO nella ricerca di un elemento v all'interno di un array di n elementi, o v non è presente o è l'ultimo elemento dell'array

Efficienza degli algoritmi

L'efficienza (il costo) di un algoritmo si può esprimere mediante una funzione f(n) nella variabile n (dimensione dell'input):

esprime il numero di operazioni elementari compiute per un problema di dimensione n

rappresenta la complessità computazionale dell'algoritmo

Se A e B sono due algoritmi che risolvono lo stesso problema e se $f_A(n)$ e $f_B(n)$ esprimono la complessità dei due algoritmi allora:

A è migliore di B se, al crescere di n, risulta $\mathbf{f_A(n) \leq f_B(n)}$

Efficienza degli algoritmi: esempio

Supponiamo di avere due algoritmi diversi per ordinare n numeri interi

- Il primo algoritmo riesce ad ordinare gli n numeri con n^2 istruzioni
- Il secondo con n * log n istruzioni

Supponiamo che l'esecuzione di un'istruzione avvenga in 10^{-6} sec

Tempo di esecuzione:

	n=10	n=10000	$n=10^6$
n^2 op.	0,1 msec	100 sec (1,5 min)	10^6 sec (12 gg.)
n*log n op.	23* 10^{-6} sec	92 msec	13,8 sec

Il modello di costo

Per giungere a un modello di costo è necessario definire:

- Dimensione dell'input
- Istruzione di costo unitario o costante (passo base)
- Calcolo della complessità in numero di passi base
- Complessità
  - ✓Nel caso migliore
  - ✓Nel caso medio
  - ✓Nel caso peggiore
- Complessità di programmi strutturati
- Complessità asintotica

Dimensione dell'input

- A seconda del problema, per dimensione dell'input si indicano cose diverse:
  - La grandezza di un numero (es: problemi di calcolo)
  - Quanti elementi sono in ingresso (es: ordinamento)
  - Quanti bit compongono un numero
- Indipendentemente dal tipo di dati, indichiamo con *n* la dimensione dell'input

Operazione (Azione) Elementare

È un'operazione la cui esecuzione non dipende dai valori e dai tipi delle variabili

- Assegnamento e operazioni aritmetiche di base
- Accesso ad un elemento qualsiasi di un vettore residente in memoria centrale
- Valutazione di un'espressione booleana qualunque
- Istruzioni di input/output

Per queste operazioni assumeremo un costo (tempo di esecuzione) COSTANTE

Valuteremo la complessità temporale in modo
- indipendente dall'HW
- in funzione della "dimensione dell'input"

Complessita dell'algorimo A
f_A: Naturali -> Naturali
dimensione dell'input -> # passi elementari dell'algoritmo A sull'input considerato

Esempio $f_A(n) = 2n^2$

Complessità Asintotica

nel valutare il tempo di calcolo T(n) di un algoritmo è difficile quantificare con esattezza il numero di operazione elementari.

Si valutano quindi il numero di operazioni in **ordine di grandezza**, cioè come limite della funzione T(n) al tendere di n all'infinito e trascurando le costanti moltiplicative ed additive.

Si parla quindi di *complessità computazionale asintotica*

Limiti asintotici

Limite superiore asintotico. Date due funzioni $f(n)$ e $g(n)$, scriveremo

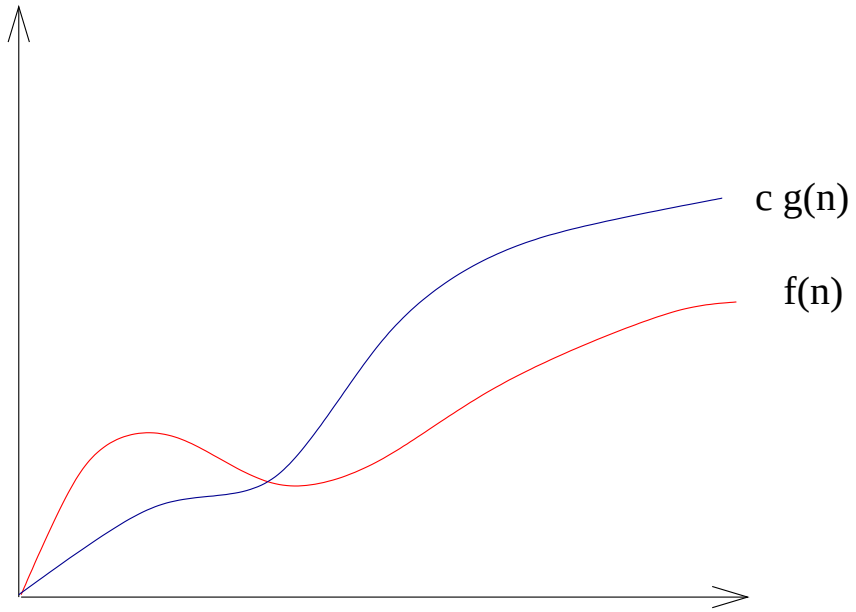
$$f(n) = O(g(n))$$

qualora esistano $c > 0$ e $n_0 \geq 0$ tali che

$$f(n) \leq cg(n)$$

per ogni $n \geq n_0$.

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



O delimitazione superiore della complessità, a meno di costanti

f(n)=O(g(n)) se esistono c > 0 e $n_0 \geq 0$ tali che f(n) ≤cg(n) per ogni $n \geq n_0$

ESEMPI

$2n^2=O(n^2)$
Basta prendere c=2 $n_0=1$ $2n^2 \leq \mathbf{2}n^2$

$3n^2 -55n =O(n^2)$
Basta prendere c=3 $n_0=1$ $3n^2 -55n \leq \mathbf{3}n^2$

$3n^2 +55n =O(n^2)$

basta prendere ad es c=58

$3n^2 + 55n \leq 3n^2 + 55n^2 = \mathbf{58}n^2$ (lo stesso per $3n^2+55$)

Limiti asintotici

Limite inferiore asintotico Ω. Date due funzioni $f(n)$ e $g(n)$, scriveremo

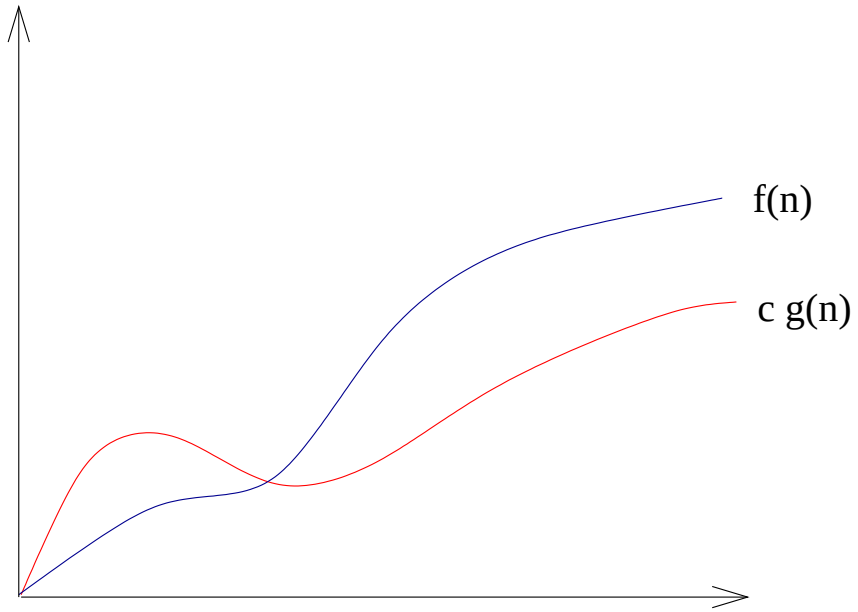
$$f(n) = \Omega(g(n))$$

qualora esistano $c > 0$ e $n_0 \geq 0$ tali che

$$f(n) \geq cg(n)$$

per ogni $n \geq n_0$;

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



Limiti asintotici

Limite asintotico stretto Θ. Date due funzioni $f(n)$ e $g(n)$, scriveremo

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

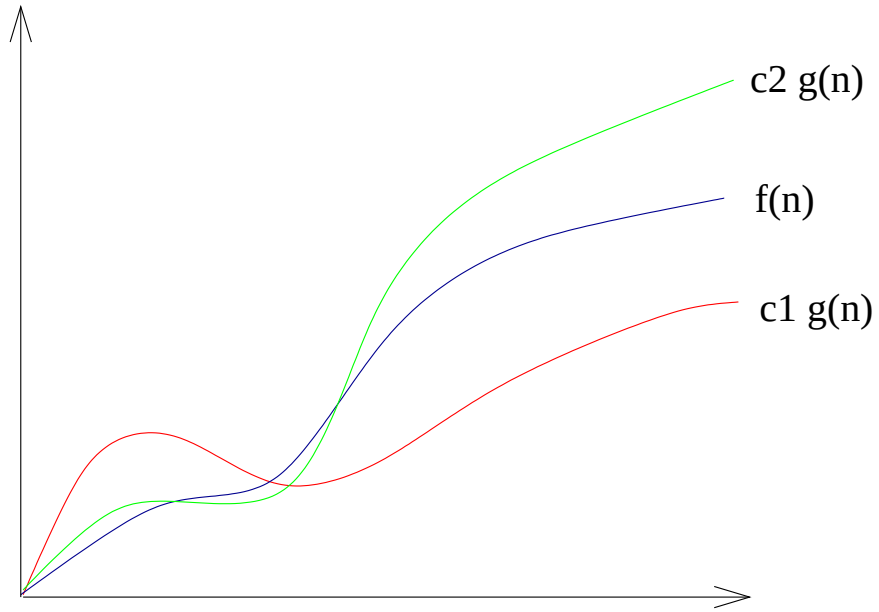
qualora esistano $c_1, c_2 > 0$ e $n_0 \geq 0$ tali che

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

per ogni $n \geq n_0$.

Si noti che $f(n) = \Theta(g(n))$ se e solo se $f(n) = \Omega(g(n))$ e $f(n) = O(g(n))$.

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



PROPRIETA'

Dato un qualsiasi polinomio P(n) di grado g (grado massimo dei monomi)

$$P(n)=\Theta(n^g)$$

Esempio $10n^4+ 890\ n^2+ 7000n= \Theta(n^4)$

Limiti asintotici

Date due funzioni $f(n)$ e $g(n)$, diremo che le funzioni $f(n)$ e $g(n)$ sono **asintoticamente equivalenti**, scritto $f(n) \sim g(n)$, se $f(n) = \Theta(g(n))$.

Diremo che $f(n)$ è **asintoticamente inferiore** a $g(n)$, scritto $f(n) \prec g(n)$, se $f(n) = O(g(n))$ e $f(n) \neq \Omega(g(n))$.

Esercizio 1 *Dimostrare che:*

- $\sim$ *è una relazione di equivalenza (cioè* $\sim$ *è riflessiva, simmetrica e transitiva); e*
- $\prec$ *è una relazione di ordinamento stretto (cioè* $\prec$ *è irriflessiva e transitiva).*

Limiti asintotici

Di seguito è indicata una scala asintotica crescente di complessità:

$$\log n \prec \sqrt[p]{n} \prec n \log n \prec n^k \prec 2^n \prec n! \prec n^n \prec 2^{2^n},$$

ove $p \geq 1$ e $k > 1$.

Diremo che un algoritmo ha **complessità polinomiale** se

$$f(n) = \Theta(n^k),$$

con $k \geq 1$, ed ha **complessità esponenziale** se

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

Limiti e limiti asintotici

E' possibile usare i limiti matematici per confrontare asintoticamente l'andamento di due funzioni di complessità.

Teorema *Date due funzioni di complessità* $f(n)$ *e* $g(n)$*, vale che:*

- *se* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$*, allora* $f(n) \prec g(n)$*;*
- *se* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$*, allora* $g(n) \prec f(n)$*;*
- *se* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = l \neq 0$*, allora* $f(n) \sim g(n)$*;*
- *se infine* $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ *non esiste, allora* $f(n)$ *e* $g(n)$ *non sono confrontabili.*

Limiti e limiti asintotici

Formulazione alternativa

Teorema *Date due funzioni di complessità* $f(n)$ *e* $g(n)$*, vale che:*

- *se* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$*, allora* $f(n) = O(g(n))$ *o* $g(n) = \Omega(f(n))$
- *se* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$*, allora* $f(n) = \Omega(g(n))$ *o* $g(n) = O(f(n))$*;*
- *se* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = l \neq 0$*, allora* $f(n) = \Theta(g(n))$

Usando i limiti matematici, è facile vedere che:

1. $c = \Theta(1)$;
2. $2n = \Theta(n)$ e $n + 1 = \Theta(n)$;
3. $2^{n+1} = \Theta(2^n)$ e $2^{2n} \neq \Theta(2^n)$.

La **morale** è (ripetere per 3 volte al giorno!):

1. ogni costante è asintotica all'unità;
2. le costanti additive e moltiplicative sono asintoticamente irrilevanti se applicate alla **base della funzione**;
3. le costanti additive ad esponente sono asintoticamente irrilevanti, ma le costanti moltiplicative **ad esponente** sono significative.

Mostriamo che

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

Infatti

$$\lim_{n\to\infty} \frac{\log(n!)}{n \log n} = 1,$$

e dunque la tesi per il teorema precedente. Per la **formula di Stirling**, abbiamo che

$$n! = \sqrt{2\pi n}(n/e)^n e^{\alpha_n},$$

ove $1/(12n+1) < \alpha_n < 1/(12n)$. Dunque

$$\log(n!) = \log\sqrt{2\pi n} + n\log(n/e) + \alpha_n \log e.$$

Abbiamo che

$$\lim_{n\to\infty} \frac{\log\sqrt{2\pi n}}{n \log n} = 0, \quad \lim_{n\to\infty} \frac{n\log(n/e)}{n \log n} = 1, \quad \lim_{n\to\infty} \frac{\alpha_n \log e}{n \log n} = 0,$$

e dunque la tesi.

Gli operatori asintotici distribuiscono rispetto alla addizione e alla moltiplicazione, cioè:

1. $\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$;
2. $\Theta(f(n))\Theta(g(n)) = \Theta(f(n)g(n))$.

Ad esempio:

$\Theta(n) + \Theta(1) = \Theta(n + 1) = \Theta(n)$

$\Theta(n) + \Theta(n) = \Theta(n + n) = \Theta(2n) = \Theta(n)$

$\Theta(n)\Theta(1) = \Theta(n1) = \Theta(n)$

$\Theta(n)\Theta(n) = \Theta(nn) = \Theta(n^2)$.

Complessità di un problema

Le classi di complessità O, Ω e Θ servono per stimare la complessità computazionale di un algoritmo
Le stesse classi possono essere applicate ai problemi (quanto è difficile un problema?)
Che significa che la complessità di un PROBLEMA è O(f(n))?
Vuol dire che esiste un algoritmo che lo risolve in tempo O(f(n))

Che significa che la complessità di un PROBLEMA è Ω(f(n))?
Vuol dire che qualunque algoritmo che lo risolve ha tempo Ω(f(n)) (più difficile da dimostrare)

Esempi:
Problema della ricerca di un elemento in un array GENERICO (ricerca lineare)
Problema dell'ordinamento di un array GENERICO

Definizione

Un problema ha complessita computazionale $\Theta(f(n))$ se è contemporaneamente $O(f(n))$ e $\Omega(f(n))$

Un problema si dice

facile se ha complessità al più polinomiale (è $O(n^k)$), e
difficile se ha complessità almeno esponenziale (è $\Omega(k^n)$ con $k>=1$).

Equazioni ricorsive

Una **equazione ricorsiva** è una equazione del tipo

$$C(n) = \ldots C(m) \ldots$$

in cui a destra e a sinistra del segno di uguaglianza compare la stessa funzione C. In sostanza, C è definita **implicitamente in termini di sè stessa**.

La funzione di complessità di una **procedura ricorsiva**, cioè di una procedura che richiamano se stessa, è spesso una equazione ricorsiva.

Per capire l'andamento asintotico di una equazione ricorsiva, occorre calcolare la sua **forma esplicita**, cioè eliminare l'occorrenza della funzione alla destra del segno di uguaglianza.

Equazioni ricorsive

Vi sono varie tecniche per risolvere una equazione ricorsiva. Forse la più efficacie consiste nel seguire i seguenti passi:

1. **svolgere l'equazione** un numero sufficiente di volte e **congetturare** una possibile forma esplicita;
2. **dimostrare la congettura** mediante l'uso di strumenti formali.

Quali strumenti formali? Master Theorem, induzione matematica...

Equazioni ricorsive

$$
\begin{aligned}
C(1) &= 1 \text{ (const1)} \\
C(n) &= C(n/2) + 1 \text{ (+const2)}
\end{aligned}
$$

Equazioni ricorsive

$$
\begin{aligned}
C(n) &= 1 + C(n/2) \\
&= 1 + 1 + C(n/4) \\
&= 1 + 1 + 1 + C(n/8) \\
&= \dots \\
&= \textstyle\sum_{i=0}^{\log n} 1 = \log n + 1 \sim \log n
\end{aligned}
$$

Equazioni ricorsive

$$
\begin{aligned}
C(1) &= 1 \\
C(n) &= C(n/2) + n
\end{aligned}
$$

Equazioni ricorsive

$$
\begin{aligned}
C(n) &= n + C(n/2) \\
&= n + n/2 + C(n/4) \\
&= n + n/2 + n/4 + C(n/8) \\
&= \dots \\
&= \textstyle\sum_{i=0}^{\log n} n/2^i = n \sum_{i=0}^{\log n} 1/2^i \sim n
\end{aligned}
$$

Infatti

$$1 \leq \sum_{i=0}^{\log n} 1/2^i < 2$$

Equazioni ricorsive

$$
\begin{aligned}
C(1) &= 1 \\
C(n) &= 2C(n/2) + 1
\end{aligned}
$$

Equazioni ricorsive

$$
\begin{aligned}
C(n) &= 1 + 2C(n/2) \\
&= 1 + 2[1 + 2C(n/4)] \\
&= 1 + 2 + 4C(n/4) \\
&= 1 + 2 + 4[1 + 2C(n/8)] \\
&= 1 + 2 + 4 + 8C(n/8) \\
&= \ldots \\
&= \textstyle\sum_{i=0}^{\log n} 2^i = 2^{\log n + 1} - 1 = 2n - 1 \sim n
\end{aligned}
$$

Equazioni ricorsive

$$
\begin{aligned}
C(1) &= 1 \\
C(n) &= 2C(n/2) + n
\end{aligned}
$$

Equazioni ricorsive

$$
\begin{aligned}
C(n) &= n + 2C(n/2) \\
&= n + 2[n/2 + 2C(n/4)] \\
&= n + n + 4C(n/4) \\
&= n + n + 4[n/4 + 2C(n/8)] \\
&= n + n + n + 8C(n/8) \\
&= \ldots \\
&= \textstyle\sum_{i=0}^{\log n} n = n(\log n + 1) \sim n \log n
\end{aligned}
$$

Equazioni ricorsive

Master Theorem *Siano $a \geq 1$ e $b > 1$ due costanti intere, e $f(n)$ una funzione di complessità. Sia $C(n) = aC(n/b) + f(n)$ una equazione ricorsiva definita sui naturali, ove per n/b si intende $\lfloor n/b \rfloor$ oppure $\lceil n/b \rceil$. Allora:*

1. *Se $f(n) = O(n^{\log_b a - \epsilon})$ per qualche $\epsilon > 0$, allora $C(n) = \Theta(n^{\log_b a})$.*
2. *Se $f(n) = \Theta(n^{\log_b a})$, allora $C(n) = \Theta(n^{\log_b a} \log n)$.*
3. *Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ per qualche $\epsilon > 0$, e se esiste una costante $c < 1$ tale che $af(n/b) \leq cf(n)$ per tutti gli n sufficientemente grandi, allora $C(n) = \Theta(f(n))$.*

Equazioni ricorsive

Sia

$$C(n) = C(n/2) + 1.$$

Allora $a = 1$, $b = 2$, $f(n) = 1$. Dunque $f(n) = \Theta(n^{\log_2 1}) = \Theta(1)$. Il caso 2 del Master Theorem si applica, e quindi $C(n) = \Theta(\log n)$.

Sia

$$C(n) = 2C(n/2) + 1.$$

Allora $a = 2$, $b = 2$, $f(n) = 1$. Dunque $f(n) = \Theta(n^{\log_2 2-\epsilon}) = \Theta(1)$, ponendo $\epsilon = 1$. Il caso 1 del Master Theorem si applica, e quindi $C(n) = \Theta(n)$.

Equazioni ricorsive

Alcune volte il Master Theorem non è applicabile. Ad esempio nei seguenti casi:

1. $C(1) = 1$ e, per $n > 0$, $C(n) = C(n-1) + 1$;
2. $C(1) = 0$ e, per $n > 0$, $C(n) = C(n-1) + \log n$;
3. $C(1) = 1$ e, per $n > 0$, $C(n) = C(n-1) + n$;

Risolviamo la terza. **Congetturiamo** una forma esplicita srotolando l'equazione...

$$\begin{aligned} C(n) &= n + C(n-1) \\ &= n + n - 1 + C(n-2) \\ &= n + n - 1 + n - 2 + \ldots 1 \\ &= \textstyle\sum_{i=1}^{n} i = n(n+1)/2 \end{aligned}$$

Dimostriamo la congettura per induzione. Se $n = 1$, allora $C(1) = 1 \cdot 2/2 = 1$. Sia $n > 1$. Allora $C(n) = n + C(n-1)$ per definizione. Applicando l'ipotesi induttiva, abbiamo che

$$C(n) = n + (n-1)n/2 = (n^2 + 2n - n)/2 = n(n+1)/2$$