Complessità astratta: notazioni asintotiche, macchina RAM, pseudocodice, ricorrenze

Achille Frigeri
Dipartimento di Matematica
Politecnico di Milano

"O", "Θ" e "Ω"- Definizioni

Siano f, g funzioni non negative (monotone crescenti)

1. $f(n) \in O(g(n))$ sse $\exists c_1 \in \mathbb{R}^+,\ \exists n_0 \in \mathbb{N},\ \forall n \geq n_0,\ f(n) \leq c_1 g(n)$, ovvero si dice anche che f è di ordine inferiore rispetto a g
2. $f(n) \in \Theta(g(n))$ sse $\exists c_1, c_2 \in \mathbb{R}^+,\ \exists n_0 \in \mathbb{N},\ \forall n \geq n_0,\ c_1 g(n) \leq f(n)$ e $f(n) \leq c_2 g(n)$, ovvero si dice anche che f e g hanno la stessa magnitudine (grandezza)
3. $f(n) \in \Omega(g(n))$ sse $\exists c_1 \in \mathbb{R}^+,\ \exists n_0 \in \mathbb{N},\ \forall n \geq n_0,\ f(n) \geq c_1 g(n)$, ovvero si dice anche che f è di ordine superiore rispetto a g

Attenzione: sono proprietà "definitive" delle funzioni

“O”, “Θ” e “Ω”- Limiti

Siano f, g funzioni non negative (monotone crescenti)

1. $f(n) \in O(g(n))$ sse $\exists c_1 \in \mathbb{R}^+,\ \exists n_0 \in \mathbb{N},\ \forall n \geq n_0,\ f(n) \leq c_1 g(n)$, ovvero si dice anche che f è di ordine inferiore rispetto a g

Quindi (se il limite esiste)

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \leq c_1$$

Ovvero il limite del rapporto tra $f(n)$ e $g(n)$ non deve essere infinito.

“O”, “Θ” e “Ω”- Limiti

Siano f, g funzioni non negative (monotone crescenti)

2. $f(n) \in \Theta(g(n))$ sse $\exists c_1, c_2 \in \mathbb{R}^+,\ \exists n_0 \in \mathbb{N},\ \forall n \geq n_0,\ c_1 g(n) \leq f(n)$ e $f(n) \leq c_2 g(n)$, ovvero si dice anche che f e g hanno la stessa magnitudine (grandezza)

Quindi (se il limite esiste)

$$c_1 \leq \lim_{n \to \infty} \frac{f(n)}{g(n)} \leq c_2$$

Ovvero il limite del rapporto tra $f(n)$ e $g(n)$ deve essere finito e non nullo.

“O”, “Θ” e “Ω”- Limiti

Siano f, g funzioni non negative (monotone crescenti)

3. $f(n) \in \Omega(g(n))$ sse $\exists c_1 \in \mathbb{R}^+,\ \exists n_0 \in \mathbb{N},\ \forall n \geq n_0,\ f(n) \geq c_1 g(n)$, ovvero si dice anche che f è di ordine superiore rispetto a g

Quindi (se il limite esiste)

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \geq c_1 > 0$$

Ovvero il limite del rapporto tra $g(n)$ e $f(n)$ non deve essere nullo.

“O”, “Θ” e “Ω”- Limiti

La notazione “Θ” serve per indicare il termine più significativo di un’espressione.
Ad esempio

$$20x^4 + 40x^3 - 100x^2 + \log(x^3 - 10) + 15 = \Theta(x^4)$$

$$0.0000001x^2 + 1000000000000000 = \Theta(x^2)$$

Complessità astratta

Descrivere una MT deterministica che riconosca il linguaggio

$$\mathcal{L} = \{w \in \{a, b\}^* \mid \#_a w = \#_b w\}$$

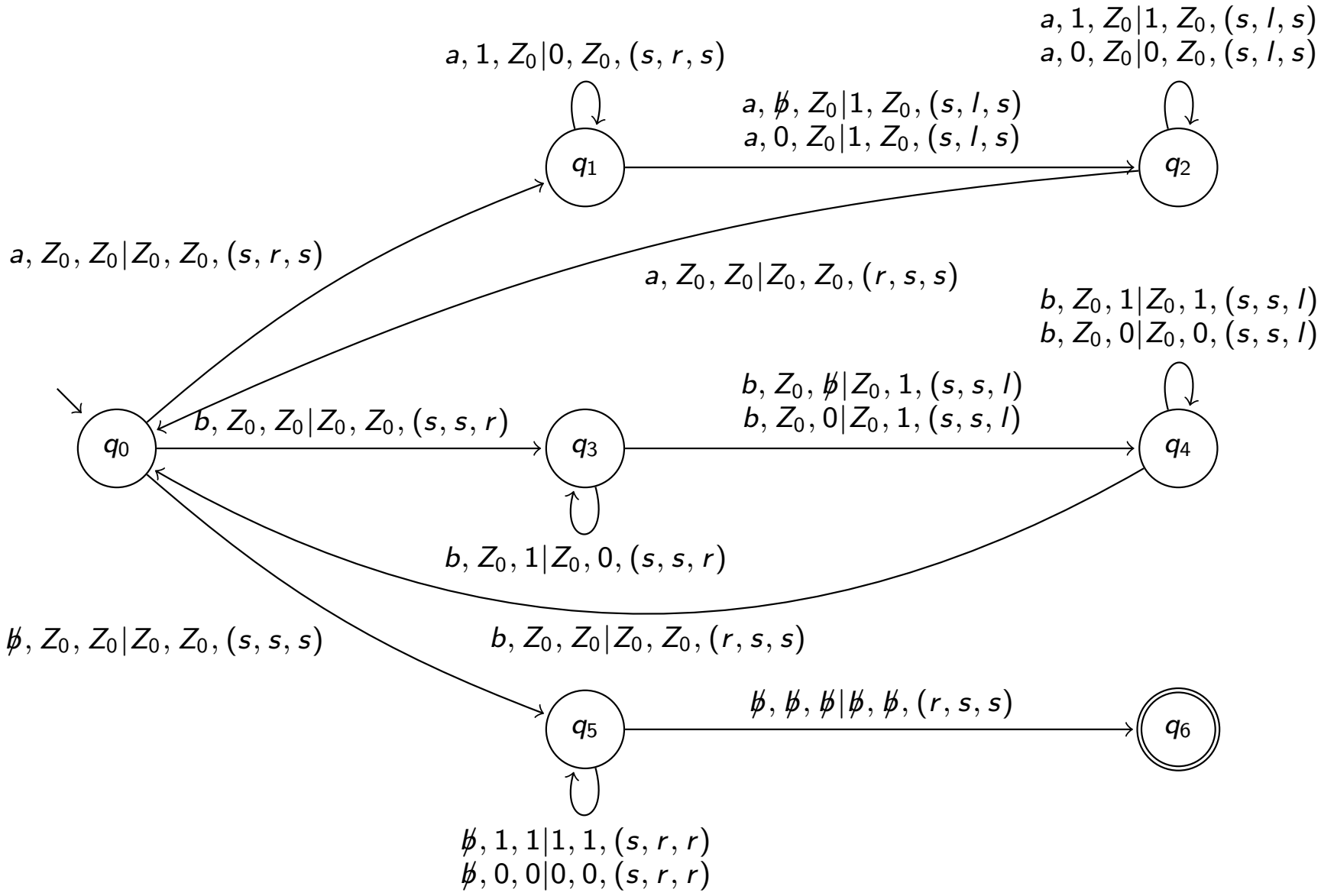
con complessità **spaziale** $\Theta(\log(n))$ dove $2n$ è la lunghezza della stringa di input.

Complessità astratta

$$\mathcal{L} = \{w \in \{a, b\}^* \mid \#_a w = \#_b w\}$$

Uso una MT a due nastri: sul primo memorizzo il numero di occorrenze delle "a" in binario (servono al più $\log_2(2n)$ celle), sull'altro quello delle "b" (in totale non più di $2\log_2(n)$ celle), poi confronto il contenuto dei nastri (non serve altra memoria): se sono uguali accetto, altrimenti rifiuto.

Complessità astratta



Complessità astratta

NB: la codifica è fatta in modo che la cifra più significativa sia più a destra.

Qual è la complessità temporale $T(n)$ della MT, misurata rispetto alla lunghezza dell'input, che assumo essere $2n$?

Iniziamo con dare una stima inferiore e una superiore per $T(n)$: la MT deve almeno leggere tutto l'input, quindi $T(n) = \Omega(n)$. Inoltre sicuramente $T(n) = O(n^2)$, visto che la conversione da base 1 a base 2 di un numero n richiede sicuramente meno di n^2 operazioni elementari, ma forse si può provare a dare una stima un po' più precisa.

Complessità astratta

Considero un momento generico della computazione in cui sul primo nastro c'è la stringa che codifica l'intero i e sto leggendo una "a" in input e sono nello stato q_0:

- passo da q_0 a q_1
- nel caso pessimo (tutti 1) ripeto il ciclo su q_1 per $\lceil \log_2(i) \rceil$ volte
- passo da q_1 a q_2
- riavvolgo il nastro ripetendo il ciclo su q_2 per $\lceil \log_2(i+1) \rceil$ volte
- passo da q_2 a q_0

Lo stesso vale per la lettera di una "b"

Complessità astratta

Terminata la lettura dell'input devo

- passare da q_0 a q_5
- ripetere il ciclo su q_5 al più (se il contenuto dei nastri è diverso mi fermo prima) $\lceil \log_2(n) \rceil$ volte
- passare da q_5 a q_6

Complessivamente

$$T(n) \leq 2 \sum_{i=1}^{n} (3 + \lceil \log_2(i) \rceil + \lceil \log_2(i+1) \rceil) + 2 + \log_2(n)$$

$$\approx \sum_{i=1}^{n} \log_2(i) = \log_2 \left(\prod_{i=1}^{n} i \right) = \log_2(n!) \leq$$

$\log_2(n^n) = \Theta(n \log(n))$

ovvero

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

Complessità astratta

Riassumendo:

con una MT a due nastri si ha $T(n) = \Omega(n)$ e $T(n) = O(n \log(n))$ (mentre $S(n) = \Theta(\log(n))$, la stima è precisa).

Si può migliorare ancora la stima per $T(n)$? Ovviamente sarà richiesta un'analisi ancora più raffinata di quella che abbiamo fatto.

Nel ragionamento precedente abbiamo introdotto un errore stimando che il costo per i riporti che si ha sommando 1 ad un generico intero i fosse sempre $\log_2(i)$, cosa che è chiaramente falsa.
In realtà metà dei numeri da 0 a $n-1$ (che sono circa $n/2$) terminano con la cifra 0, per questi il costo dovuto ai riporti è solo un'unità di tempo. Per la restante metà, la metà (cioè $n/4$) termina con "01" e quindi il costo dovuto ai riporti è 2. Per metà dei restanti ($n/8$), cioè quelli che terminano con "011", è 3, ...

Complessità astratta

Allora il costo complessivo dato dai riporti è

$$\frac{n}{2} \cdot 1 + \frac{n}{4} \cdot 2 + \frac{n}{8} \cdot 3 + \frac{n}{16} \cdot 4 + \ldots = \sum_{i=1}^{\lceil \log_2(n) \rceil} \frac{n}{2^i} \cdot i = n \cdot \sum_{i=1}^{\lceil \log_2(n) \rceil} \frac{i}{2^i}$$

Possiamo stimare per eccesso la sommatoria $\sum_{i=1}^{\lceil \log_2(n) \rceil} \frac{i}{2^i}$ con

$$\int_0^{\infty} x \cdot 2^{-x} dx$$

che, integrando per parti, risulta $\frac{1}{\log_e^2(2)} \approx 2$.

Allora il costo complessivo della conversione di due numeri naturali x e y, la cui somma è $2n$, da base 1 a base 2 è circa $2x + 2y = 2n$.
Aggiungendo i costi restanti (in pratica solo il confronto delle lunghezze dei due nastri, che richiede al più $\log(n/2)$ confronti), si ottiene comunque una complessità asintotica $T(n) = \Theta(n)$.

Macchina RAM

Una macchina RAM è un calcolatore semplificato che ha di un nastro di lettura (In) e uno di scrittura (Out) come una MT.
Opera su registri che contengono interi e a cui si può accedere direttamente con il loro indirizzo: $M[0]$, $M[1]$, ecc.
C'è un registro particolare, $M[0]$, detto accumulatore su cui avvengono tutte le operazioni.
Le istruzioni servono per effettuare le operazioni aritmetiche, gestire la memoria ed effettuare cicli (salti).

Macchina RAM

Istruzione	Significato
$LOAD\ X$	$M[0] \leftarrow M[X]$
$LOAD\ =X$	$M[0] \leftarrow X$
$LOAD\ \hat{}X$	$M[0] \leftarrow M[M[X]]$
$STORE\ X$	$M[X] \leftarrow M[0]$
$STORE\ \hat{}X$	$M[M[X]] \leftarrow M[0]$
$ADD\ X$	$M[0] \leftarrow M[0] + M[X]$
$ADD\ =X$	$M[0] \leftarrow M[0]X$
$ADD\ \hat{}X$	$M[0] \leftarrow M[0] + M[M[X]]$
$SUB\ X$	$M[0] \leftarrow M[0] - M[X]$
$MUL\ X$	$M[0] \leftarrow M[0] \cdot M[X]$
$DIV\ X$	$M[0] \leftarrow M[0]/M[X]$
$HALT$	fine programma

X è un intero, anche SUB, $MULT$ e DIV hanno le varianti con $=$ e $\hat{}$ come ADD; DIV è la divisione intera.

Macchina RAM

Istruzione	Significato
READ X	$M[X] \leftarrow In$
READ ^X	$M[M[X]] \leftarrow In$
WRITE X	$Out \leftarrow M[X]$
WRITE =X	$Out \leftarrow X$
WRITE ^X	$Out \leftarrow M[M[X]]$
JUMP tag	va all'istruzione con etichetta tag
JGTZ tag	va all'istruzione tag se $M[0] > 0$
JZERO tag	va all'istruzione tag se $M[0] = 0$

tag è un'etichetta qualunque, cioè una stringa di caratteri.
Qui `https://random-access-machine-emulator.netlify.app/` trovate un simulatore.

Complessità astratta

Dire cosa calcola è qual è la complessità a costo logaritmico della seguente macchina RAM (se in input riceve n, cosa calcola la macchina?)

			R0	R1	R2	R3
	READ	0	n	n	n	1
	STORE	1	1		$n-1$	n
	STORE	2	n		$n-2$	n^2
	LOAD =	1	n		...	...
	STORE	3	n		0	n^n
loop	LOAD	1	$n-1$			
	MULT	3	n			
	STORE	3	n^2			
	LOAD	2	$n-1$			
	SUB =	1	$n-2$			
	STORE	2	n			
	JGZ	loop	n^3			
	WRITE	3	...			
	HALT		n^n			

La funzione l misura la lunghezza del suo input, è "quasi" il logaritmo, infatti è definita anche su 0 ($l(0) = 1$) e su ogni stringa di caratteri, di cui misura la lunghezza di una codifica binaria.

Complessità astratta

Dire cosa calcola è qual è la complessità a costo logaritmico della seguente macchina RAM (se in input riceve n, cosa calcola la macchina?)

	READ	0	$l(n) + l(0)$
	STORE	1	$l(n) + l(1)$
	STORE	2	$l(n) + l(2)$
	LOAD =	1	$l(1)$
	STORE	3	$l(3) + l(1)$
loop	LOAD	1	$l(n) + l(1)$
	MULT	3	$l(n) + l(3) + l(n^{i-1})$
	STORE	3	$l(n^i) + l(3)$
	LOAD	2	$l(i-1) + l(3)$
	SUB =	1	$l(1)$
	STORE	2	$l(i-1) + l(2)$
	JGZ	loop	$l(i)$
	WRITE	3	$l(n^n) + l(3)$
	HALT		$l(1)$

Complessità astratta

Sommando tutte le complessità si ha

$$
\begin{aligned}
T(n) &= O\left(\sum_{i=1}^{n}\left(l(n^{i-1}) + l(n^{i}) + l(i-1) + cost\right) + l(n^{n}) + cost\right) \\
&= O\left(\sum_{i=1}^{n}\log(n^{i})\right) = O\left(\sum_{i=1}^{n} i \cdot \log(n)\right) \\
&= O\left(\log(n)\sum_{i=1}^{n} i\right) = O\left(n^{2}\log(n)\right)
\end{aligned}
$$

La complessità spaziale è data dalla massima occupazione dei registri, dunque $S(n) = \Theta(l(n^{n})) = \Theta(n\log(n))$.

Complessità astratta

Osservazione: la complessità ottenuta è la stessa del seguente frammento di pseudocodice

```
0        m = n;
1        for (int i = 1; i <= m; i++)
2             n = n*m;
```

assumendo costi logaritmici.

Infatti la terza istruzione ha un costo $O(\log(n^i))$ per un totale di $O(\sum_{i=1}^{n} \log(n^i))$.

Complessità astratta

Si consideri il problema di ordinare una stringa di caratteri appartenenti all'alfabeto latino in ordine alfabetico considerando anche le eventuali ripetizioni.
Otterremo una complessità temporale minore con una MT a k nastri o con una macchina RAM con costi logaritmici?

Su una MT uso tanti nastri quanti sono i caratteri (lo posso fare perché so a priori quanti caratteri diversi si possono presentare), ognuno è usato per memorizzare in unario le occorrenze di ogni carattere. Dopo aver letto la stringa e memorizzato le occorrenze di ogni carattere stampo i caratteri corrispondenti ai nastri con le ripetizioni corrette e nell'ordine corretto: complessità $\Theta(n)$.

N.B.: questa tecnica di ordinamento prende il nome di "counting sort" e ha senso solo se nel file da ordinare ci sono pochi dati diversi ripetuti molte volte.

Complessità astratta

Si consideri il problema di ordinare una stringa di caratteri appartenenti all'alfabeto latino in ordine alfabetico considerando anche le eventuali ripetizioni.
Conviene risolvere il problema con una MT a k nastri o con una macchina RAM con costi logaritmici?

Con una macchina RAM uso tanti registri quanti sono i caratteri e procedo come sulla MT. Però un singolo incremento del valore di un registro è $O(\log(i))$ se i è il contenuto del registro: la complessità totale è quindi $O(n \log(n))$.

La MT e la RAM eseguono lo stesso algoritmo, ma il risultato non deve sorprendere: l'analisi di una macchina RAM con costi logaritmici fornisce solo una stima (normalmente per eccesso) della complessità reale della procedura, notate infatti che ho usato la notazione O-grande e non Θ-grande.

Analisi di "pseudocodice"- 1

Valutare la complessità asintotica della procedura P (o frammento di procedura) al variare di n:

```
0  P(n)
1  int i = 2, j=0
2  while (i <= n)
3          j++
4          i=i*i
```

C'è un unico ciclo eseguito finché il contatore i non supera n, ad ogni esecuzione del ciclo i viene elevato al quadrato, dunque assume i valori

$$2,\ 4,\ 16,\ 256,\ \ldots$$

ovvero

$$2,\ 2^{2^1},\ 2^{2^2},\ 2^{2^3},\ \ldots$$

quindi se h è il numero di ripetizioni del ciclo, si ha che h è il più piccolo naturale per cui

$$2^{2^h} > n$$

ovvero

$$h \sim \log_2(\log_2(n))$$

Analisi di "pseudocodice"- 1

Valutare la complessità asintotica della procedura P (o frammento di procedura) al variare di n:

```
0  P(n)
1  int i = 2, j=0
2  while (i <= n)
3              j++
4              i=i*i
```

Poiché ogni ciclo consta di due istruzioni più una verifica, il numero totale di istruzione è circa

$$3 \cdot \log_2(\log_2(n)) + 1$$

quindi la complessità del frammento di codice è

$$\Theta(\log(\log(n)))$$

Analisi di "pseudocodice"- 2

Valutare la complessità asintotica della procedura P (o frammento di procedura) al variare di n:

```
0  P(n)
1  int s = 0, j, k
2  for (int i = 1; i <= n; i++)
3          j = 1
4          while (j < n)
5                  k = 1
6                  while (k < n)
7                          s++
8                          k = 3*k
9                  j= 2*j
```

Tre cicli annidati dipendenti da tre parametri (indipendenti tra loro):

1. nel primo i cresce linearmente fino a n (ripetuto n volte)
2. nel secondo j cresce come 2^h fino a n (ripetuto $\log_2(n)$ volte)
3. nel terzo k cresce come $3^{h'}$ fino a n (ripetuto $\log_3(n)$ volte)

Analisi di "pseudocodice"- 2

```
0  P(n)
1  int s = 0, j, k
2  for (int i = 1; i <= n; i++)
3           j = 1
4           while (j < n)
5                   k = 1
6                   while (k < n)
7                           s++
8                           k = 3*k
9                   j= 2*j
```

L'istruzione più interna (s++) viene eseguita (circa) $n \cdot \log_2(n) \cdot \log_3(n)$ volte, quindi la complessità del frammento di codice è

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

Analisi di "pseudocodice"- 3

Valutare la complessità asintotica della procedura P (o frammento di procedura) al variare di n:

```
0  P(n)
1  int a = 0, j = 1, k = 0, h
2  while (j < n)
3            k++
4            for (int i = 1; i <= k;    i++)
5                     h = 2
6                     while (h < 2^n)
7                                a++
8                                h = h*h
9            j = 2*j
```

Più difficile perché i cicli sono correlati.
Nel ciclo più esterno j assume i valori $1, 2, 4, 8, \ldots$, viene dunque eseguito circa $\log_2(n)$ volte.

Analisi di "pseudocodice"- 3

```
0  P(n)
1  int a = 0, j = 1, k = 0, h
2  while (j < n)
3          k++
4          for (int i = 1; i <= k;    i++)
5                  h = 2
6                  while (h < 2^n)
7                          a++
8                          h = h*h
9          j = 2*j
```

k assume i valori da 1 a $\log_2(n)$, e l'istruzione alla riga 5 viene eseguita k volte ogni volta, quindi in totale

$$1+2+\ldots+\log_2(n) = \sum_{i=1}^{\log_2(n)} i = \frac{\log_2(n)\cdot(\log_2(n)+1)}{2} = \Theta(\log_2^2(n)) \text{ volte}$$

Nel "while" più annidato, h cresce come 2^{2^x} fino ad arrivare a 2^n, viene dunque eseguito $\log_2(n)$ volte

Analisi di "pseudocodice"- 3

```
0  P(n)
1  int a = 0, j = 1, k = 0, h
2  while (j < n)
3            k++
4            for (int i = 1; i <= k;    i++)
5                     h = 2
6                     while (h < 2^n)
7                               a++
8                               h = h*h
9            j = 2*j
```

Riassumendo: la complessità complessiva dei due cicli correlati è $\Theta(\log^2(n))$, quella del ciclo più interno è $\Theta(\log(n))$, la complessità totale del frammento di codice è quindi il prodotto delle due, cioè

$$\Theta(\log^3(n))$$

Analisi di "pseudocodice"- 4

Valutare la complessità asintotica della procedura P (o frammento di procedura):

```
0  P(n)
1  int sum = 0, j
2  for (int i = 1; i <= log(n); i++)
3           j = 2
4           while (j < 2^n)
5                   sum = sum + 1
6                   j = 2*j
```

Nel ciclo più esterno i assume i valori da 1 a $\log(n)$, viene dunque eseguito $\log(n)$ volte. Nel ciclo più interno j cresce come 2^x fino a 2^n, quindi ogni volta viene eseguito n.
Il costo dell'istruzione "$sum = sum + 1$" è assunto costante (nel modello a costi costanti).

Complessivamente la complessità del frammento di codice è

$$\Theta(n \cdot \log(n))$$

Analisi di "pseudocodice"- 4 bis

Sia $f(n)$ una procedura il cui costo è $\Theta(\log(n))$, valutare la complessità asintotica della procedura P (o frammento di procedura):

```
0  P(n)
1  int sum = 0, j
2  for (int i = 1; i <= log(n); i++)
3           j = 2
4           while (j < 2^n)
5                   sum = sum + f(n)
6                   j = 2*j
```

Nel ciclo più esterno i assume i valori da 1 a $\log(n)$, viene dunque eseguito $\log(n)$ volte. Nel ciclo più interno j cresce come 2^x fino a 2^n, quindi ogni volta viene eseguito n.
Il costo dell'istruzione "$sum = sum + f(n)$" è $\Theta(\log(n))$ (nel modello a costi costanti).

Complessivamente la complessità del frammento di codice è

$$\Theta(n \cdot \log^2(n))$$

Analisi di "pseudocodice"- 4 ter

Sia $f(n)$ una procedura il cui costo è $\Theta(\log(n))$, valutare la complessità asintotica della procedura P (o frammento di procedura):

```
0  P(n)
1  int sum = 0, j
2  for (int i = 1; i <= log(n); i++)
3           j = 2
4           while (j < 2^n)
5                   sum = sum + f(i)
6                   j = 2*j
```

Per i cicli valgono le stesse considerazioni svolte prima, ma il costo dell'istruzione "$sum = sum + f(i)$" dipende dal parametro i e quindi non è costante durante l'esecuzione di P (nel modello a costi costanti).
Il numero di istruzione eseguite è quindi asintotico a

$$\sum_{i=1}^{\log(n)} n \cdot \log(i)$$

Analisi di "pseudocodice"- 4 ter

"Quanto fa" $\sum_{i=1}^{\log(n)} n \cdot \log(i)$?
Osservo che per le proprietà dei logaritmi

$$\sum_{i=1}^{n} \log(i) = \log\left(\prod_{i=1}^{n} i\right)$$

quindi

$$\sum_{i=1}^{\log(n)} n \cdot \log(i) = n \cdot \sum_{i=1}^{\log(n)} \log(i) = n \cdot \log\left(\prod_{i=1}^{\log(n)} i\right) \sim n \cdot \log(\lceil \log(n) \rceil !)) =$$

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

dove ho usato la formula

$$\log(A!) = \Theta(A \cdot \log(A))$$

con $A = \log(n)$.

Analisi di "pseudocodice"- 5

Valutare la complessità asintotica della procedura P (o frammento di procedura) al variare di n:

```
0  P(n)
1  int ris = 0, m
2  if (n%2 == 0)
3           m = n/2
4           ris = 1
5           for (int i = 1; i <= m; i++)
6                   ris = ris*n
7  else ris = n*n*n
```

Il programma P calcola la funzione

$$f(n) = \begin{cases} n^{\frac{n}{2}}, & \text{se } n \text{ è pari;} \\ n^3, & \text{se } n \text{ è dispari.} \end{cases}$$

Analisi di "pseudocodice"- 5

```
0  P(n)
1  int ris = 0, m
2  if (n%2 == 0)
3          m = n/2
4          ris = 1
5          for (int i = 1; i <= m; i++)
6                  ris = ris*n
7  else ris = n*n*n
```

Se n è dispari viene eseguita una sola istruzione, la complessità è dunque costante (cioè $\Theta(1)$) nel caso ottimo.
Il caso pessimo è chiaramente quello in cui n è pari: il ciclo "for " viene eseguito $\frac{n}{2}$ volte la complessità risulta essere $\Theta(n)$.

Procedure ricorsive

A volte una procedura P può richiamare al suo interno se stessa: come calcolarne la complessità?

Esempio (classico): fattoriale

```
0  int factorial(int n){
1  if (n<=0) return 1;
2          return n*factorial(n-1);
3  }
```

(Questa è una function in C++ funzionante)
Sia $T(n)$ la funzione di costo temporale della procedura "factorial", allora

$$T(n) = \begin{cases} 2, & \text{se } n = 0; \\ 2 + T(n-1), & \text{se } n > 0. \end{cases}$$

Procedure ricorsive

$$T(n) = \begin{cases} 2, & \text{se } n = 0; \\ 2 + T(n-1), & \text{se } n > 0. \end{cases}$$

Si tratta di una "equazione alle ricorrenze" proviamo ad "espanderla".
Se $n > 0$ si ha:

$$\begin{aligned} T(n) &= 2 + T(n-1) = 2 + (2 + T(n-2)) = \ldots = \\ &= 2 + (2 + (2 + \ldots + (2 + T(0)) \ldots)) = 2 + 2n \end{aligned}$$

quindi il costo della procedura è

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

Procedure ricorsive - 1

Siano A e B due procedure, valutare la complessità asintotica della procedura A:

```
0  A(n)
1  int s = 0
2  for (int i = 1; i <= n; i++)
3           s = s+B(i)
4  return s
```

```
0  B(m)
1  int t = 0
2  for (int i = 1; i <= m; i++)
3           t = t+i
4  return t
```

Non c'è alcuna ricorsione, semplicemente A chiama B. Iniziamo calcolando la funzione di costo temporale $T_B(m)$ di B. È facile osservare che $T_B(m) = \Theta(m)$, allora la complessità $T_A(n)$ è data da:

$$T_A(n) = \sum_{i=1}^{n} \Theta(i) = \Theta\left(\sum_{i=1}^{n} i\right) = \Theta\left(\frac{n \cdot (n+1)}{2}\right) = \Theta(n^2)$$

Procedure ricorsive - 1 bis

Siano A e B due procedure, valutare la complessità asintotica della procedura A:

```
0  A(n)
1  int s = 0
2  for (int i = 1; i <= n; i++)
3           s = s+B(n)
4  return s
```

```
0  B(m)
1  if (m = 1)
2           return 1
3  else return B(m/2)+m
```

Ora c'è ricorsione all'interno di B.
Il costo $T_B(m)$ di B è

$$T_B(m) = \begin{cases} 1, & \text{se } m = 1; \\ T_B(m/2) + c, & \text{altrimenti, con } c \text{ costante opportuna.} \end{cases}$$

Procedure ricorsive - 1 bis

$$T_B(m) = \begin{cases} 1, & \text{se } m = 1; \\ T_B(m/2) + c, & \text{altrimenti, con } c \text{ costante opportuna.} \end{cases}$$

Svolgendo la ricorrenza, con $m > 1$, si ha

$$T_B(m) = c + T_B\left(\frac{m}{2}\right) = c + \left(c + T_B\left(\frac{m}{4}\right)\right) = \ldots =$$
$$= c + \left(c + \left(c + \ldots + \left(c + T_B\left(\frac{m}{2^h}\right)\right)\ldots\right)\right)$$

fino a che $2^h = m$, ovvero $h = \log(m)$.
Quindi

$$T_B(m) = c \cdot \log(m) = \Theta(\log(m))$$

Allora la complessità $T_A(n)$ è data da:

$$T_A(n) = \sum_{i=1}^{n} T_B(n) = \sum_{i=1}^{n} \Theta(\log(n)) = \Theta(\log(n)) \sum_{i=1}^{n} 1 = \Theta(n \cdot \log(n))$$

Procedure ricorsive - 2

Siano P una procedura tale che $T_P(m) = \Theta(\sqrt{m})$ e sia Fun(A,n) una procedura che prende in ingresso un naturale n e un array A lungo n, valutare la complessità asintotica di Fun (rispetto a n):

```
0  Fun(A,n)
1  if (n < 1)
2          return 1
3  t = Fun(A,n/2)
4  if (t > n^2)
5          t = t- Fun(A,n/2)/2
6  for (int i = 1; i <= n; i++)
7          t = t + A[i]+P(n)
8  return t
```

La procedura è ricorsiva e nel caso peggiore Fun viene richiamata due volte su un input di lunghezza $n/2$.
Il ciclo "for" viene eseguito n volte ed ha un costo totale $\Theta(n\sqrt{n})$.
Allora (nel caso pessimo) il costo $T(n)$ di $Fun(A, n)$ è

$$T(n) = \begin{cases} 1, & \text{se } n = 1; \\ 2 \cdot T(n/2) + \Theta(n\sqrt{n}), & \text{altrimenti.} \end{cases}$$

Procedure ricorsive - 2

$$T(n) = \begin{cases} 1, & \text{se } n = 1; \\ 2 \cdot T(n/2) + \Theta(n\sqrt{n}), & \text{altrimenti.} \end{cases}$$

Svolgendo la ricorrenza, con $n > 1$, si ha

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n\sqrt{n}) = 2\left(2T\left(\frac{n}{4}\right) + \Theta\left(\frac{n}{2}\sqrt{\frac{n}{2}}\right)\right) + \Theta(n\sqrt{n}) =$$

$$= \ldots = 2\left(2\left(2\ldots\left(2T\left(\frac{n}{2^h}\right) + \Theta\left(\frac{n}{2^{h-1}}\sqrt{\frac{n}{2^{h-1}}}\right)\right)\ldots\right)\right) + \Theta(n\sqrt{n})$$

fino a che $2^h = n$, ovvero $h = \log(n)$.
Allora la complessità $T(n)$ è data da:

$$T(n) = \sum_{i=0}^{\log(n)-1} 2^i \cdot \Theta\left(\frac{n}{2^i}\sqrt{\frac{n}{2^i}}\right)$$

Procedure ricorsive - 2

Svolgendo i conti si ha

$$T(n) = \sum_{i=0}^{\log(n)-1} 2^i \cdot \Theta\left(\frac{n}{2^i}\sqrt{\frac{n}{2^i}}\right) = \Theta\left(\sum_{i=0}^{\log(n)-1} 2^i \cdot \frac{n}{2^i}\sqrt{\frac{n}{2^i}}\right) =$$

$$= \Theta\left(n\sqrt{n} \cdot \sum_{i=0}^{\log(n)-1} \frac{1}{\sqrt{2^i}}\right)$$

L'ultima sommatoria converge per $n \to \infty$ (è una geometrica di ragione $1/\sqrt{2}$), quindi è limitata da una costante opportuna, ne segue

$$T(n) = \Theta(n\sqrt{n})$$

Per la cronaca, assumendo che P abbia complessità esatta $\sqrt{n}$, la soluzione esatta dell'equazione alle ricorrenze sarebbe
$T(n) = (1+\sqrt{2})(\sqrt{2n}-1)n$.

Teorema principale

Alcune ricorrenze possono essere risolte direttamente applicando il cosiddetto teorema principale.

Data l'equazione alle ricorrenze

$$T(n) = \begin{cases} d, & n = 1 \\ aT\left(\frac{n}{b}\right) + f(n), & n > 1 \end{cases}$$

allora

1. se esiste $\varepsilon > 0$ tale che $f(n) = O(n^{\log_b(a)-\varepsilon})$, allora $T(n) = \Theta(n^{\log_b(a)})$
2. se esiste $k \geq 0$ tale che $f(n) = \Theta(n^{\log_b(a)} \log^k(n))$, allora $T(n) = \Theta(n^{\log_b(a)} \log^{k+1}(n))$
3. se esiste $\varepsilon > 0$ tale che $f(n) = \Omega(n^{\log_b(a)+\varepsilon})$ e se definitivamente $a \cdot f\left(\frac{n}{b}\right) < c \cdot f(n)$ per qualche $c < 1$, allora $T(n) = \Theta(f(n))$

Teorema principale: un caso difficile

L'equazione

$$T(n) = \begin{cases} 1, & \text{se } n = 1; \\ 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n\sqrt{n}), & \text{altrimenti} \end{cases}$$

potrebbe rientrare nel terzo caso (quello più difficile). Infatti $a = b = 2$, quindi $\log_b(a) = 1$, e $f(n) = \Theta(n\sqrt{n})$. Non vale né $f(x) = O(n^{1-\varepsilon})$ con $\varepsilon > 0$, né $f(x) = \Theta(n)$. Vale invece $f(x) = \Omega(n^{1+\varepsilon})$ con $\varepsilon > 0$.

Resta però da provare l'ulteriore condizione su $f(n)$ (la regolarità), ovvero che definitivamente $2f\left(\frac{n}{2}\right) < cf(n)$ per qualche $c < 1$. Ma questo in generale è falso.

Teorema principale: un caso difficile

Infatti, poiché $f(n) = \Theta(n\sqrt{n})$, possiamo dire che definitivamente $c_1 n\sqrt{n} \leq f(n) \leq c_2 n\sqrt{n}$. Allora possiamo ricavare

$$f(n) \geq c_1 n\sqrt{n} = 2\sqrt{2}c_1 \frac{n}{2}\sqrt{\frac{n}{2}} \geq \frac{2\sqrt{2}c_1}{c_2} f\left(\frac{n}{2}\right) = \frac{\sqrt{2}c_1}{c_2} 2f\left(\frac{n}{2}\right)$$

da cui

$$2f\left(\frac{n}{2}\right) \leq \frac{c_2}{\sqrt{2}c_1} f(n)$$

ma $\frac{c_2}{\sqrt{2}c_1}$ non è necessariamente minore di 1.

Teorema principale: un caso difficile

Ad esempio, sia f la funzione definita in questo modo:

$$f(n) = \begin{cases} n\sqrt{n}, & \text{se } n \text{ è pari;} \\ 2n\sqrt{n}, & \text{se } n \text{ è dispari} \end{cases}$$

Allora $f(n) = \Theta(n\sqrt{n})$ e non soddisfa la condizione di regolarità. Infatti per ogni $n = 2m$ con m dispari, la condizione di regolarità impone che sia $2m\sqrt{m} < c2m\sqrt{m}$, che vale solo per $c > 1$. Tuttavia, separatamente le funzioni $g_1(n) = n\sqrt{n}$ e $g_2(n) = 2n\sqrt{n}$ soddisfano la condizione, e questo permette di fare un'importante osservazione.

Teorema principale: un caso difficile

Consideriamo l'equazione

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

e sia $c_1 g(n) \leq f(n) \leq c_2 g(n)$ per ogni $n > n_0$. Se consideriamo le due equazioni

$$T_1(n) = aT_1\left(\tfrac{n}{b}\right) + c_1 \cdot g(n)$$
$$T_2(n) = aT_2\left(\tfrac{n}{b}\right) + c_2 \cdot g(n)$$

con dati iniziali $T_1(n) = T_2(n) = T(n)$ per ogni $1 \leq n \leq n_0$, allora non è difficile provare per induzione che per ogni $n \in \mathbb{N}$ si ha

$$T_1(n) \leq T(n) \leq T_2(n).$$

Nel caso precedente abbiamo $f(n) = \Theta(n\sqrt{n})$ e $g(n) = n\sqrt{n}$ soddisfa la condizione di regolarità (provate), allora ne segue che $T_1(n) = \Theta(n\sqrt{n})$ e $T_2(n) = \Theta(n\sqrt{n})$, da cui anche $T(n) = \Theta(n\sqrt{n})$.