Lezione_03_DeA

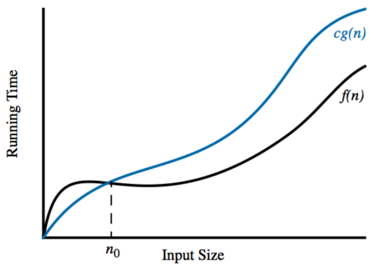
2.3. Ordini di grandezza

Siano $f(n)$, $g(n)$ funzioni da $\mathbb{N}$ a $\mathbb{R}^+ \cup \{0\}$.

2.3.1. O-grande

> $f(n) \in O(g(n))$ se $\exists c > 0$ e $\exists n_0 \geq 1$, *costanti rispetto a n*, tali che
> $$f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$$

Cioè se $f(n)$ è al più proporzionale a $g(n)$, ovvero se $f(n)$ non cresce asintoticamente più di $c \cdot g(n)$.



2.3.1.1. Esempi

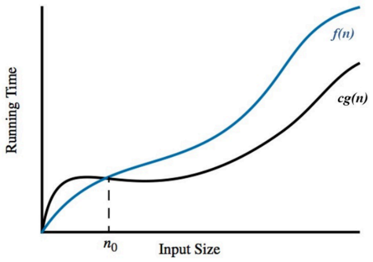
$f(n)$	$O(\cdot)$	c	n_0
$3n+4$ per $n \geq 1$	$O(n)$	4	4
$n+2n^2$ per $n \geq 1$	$O(n^2)$	3	1
2^{100} per $n \geq 1$	$O(1)$	2^{100}	1
$c_1 n + c_2$ per $n \geq 1$, $c_1, c_2 > 0$	$O(n)$	$c_1 + c_2$	1

Si può dire che $3n+4 \in O(n^5)$, $c = 7$, $n_0 = 1$, ma non sarebbe molto utile.

2.3.2. Omega-grande

> $f(n) \in \Omega(g(n))$ se $\exists c > 0$ e $\exists n_0 \geq 1$, *costanti rispetto a n*, tali che
> $$f(n) \geq c \cdot g(n) \quad \forall n \geq n_0$$

Cioè $f(n)$ è almeno proporzionale a $g(n)$.



2.3.2.1. Esempi

$f(n)$	$\Omega(\cdot)$	c	n_0
$3n+4$ per $n \geq 1$	$\Omega(n)$	1	4
$n+2n^2$ per $n \geq 1$	$\Omega(n^2)$	1	1
2^{100} per $n \geq 1$	$\Omega(1)$	2^{100}	1
$c_1 n + c_2$ per $n \geq 1$, $c_1, c_2 > 0$	$\Omega(n)$	1	1

2.3.3. Theta

> $f(n) \in \Theta(g(n))$ se
> $$f(n) \in O(g(n)) \text{ e } f(n) \in \Omega(g(n))$$

Cioè $f(n)$ è (*esattamente*) proporzionale a $g(n)$.

2.3.3.1. Esempi

- $f(n) = 3n+4 \in \Theta(n)$;
- $f(n) = n+2n^2 \in \Theta(n^2)$;
- $f(n) = 2^{100} \in \Theta(1)$;
- $t_{\text{arrayMax}}(n) \in \Theta(n)$.

2.3.4. o-piccolo

> $f(n) \in o(g(n))$ se
> $$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0$$

Cioè $f(n)$ è *asintoticamente* più piccola (cresce meno) di $g(n)$.

2.3.4.1. Esempi

- $f(n) = 100n$ per $n \geq 1 \implies f(n) \in o(n^2)$;
- $f(n) = \frac{3n}{\log_2 n}$ per $n \geq 1 \implies f(n) \in o(n)$.

2.3.5. Proprietà degli ordini di grandezza

1. $max\{f(n), g(n)\} \in \Theta(f(n)+g(n))$ per ogni $f(n), g(n) : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$;
2. $\sum_{i=0}^{k} a_i n^i \in \Theta(n^k)$, se $a_k > 0$, k, a_i costanti, e $k \geq 0$.
   Ad esempio: $(n+1)^5 \in \Theta(n^5)$.
   Cioè tengo il termine con l'esponente maggiore;
3. $\log_b n \in \Theta(\log_a n)$, se $a, b > 1$ sono costanti;

> **Osservazione**
> La proprietà deriva dalla relazione $\log_b n = (\log_a n)(\log_b a)$ e, grazie a essa, *la base dei logaritmi*, se costante, *si omette negli ordini di grandezza*, a meno che il logaritmo non sia all'esponente.

4. $n^k \in o(a^n)$, se $k > 0$, $a > 1$ sono costanti;
5. $(\log_b n)^k \in o(n^h)$ se b, k, h sono costanti, con $b > 1$ e $h, k > 0$.

2.3.6. Strumenti matematici

2.3.6.1. Parte bassa

> $\forall x \in \mathbb{R}$, si definisce $\lfloor x \rfloor$ come il più grande intero tale che sia $\leq x$
> .

2.3.6.1.1. Esempi

- $\lfloor \frac{3}{2} \rfloor = 1$;
- $\lfloor 3 \rfloor = 3$.

2.3.6.2. Parte alta

> $\forall x \in \mathbb{R}$, si definisce $\lceil x \rceil$ come il più piccolo intero tale che sia $\geq x$.

2.3.6.2.1. Esempi

- $\lceil \frac{3}{2} \rceil = 2$;
- $\lceil 3 \rceil = 3$.

2.3.6.3. Modulo

> $\forall x, y \in Z$, con $y \neq 0$, si definisce $x \bmod y$ come il resto della divisione intera x/y (l'operatore "%" in Java).

2.3.6.3.1. Esempi

- $29 \bmod 7 = 4$;
- $80 \bmod 4 = 0$.

2.3.6.4. Sommatorie notevoli

$\forall n \in \mathbb{Z}$ e $a \in \mathbb{R}$, con $n \geq 0$ e $a > 0$ vale:

- $\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \in \Theta(n^2)$;
- $\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1} \in \Theta(a^n)$ per $a > 1$;
- $\sum_{i=1}^{n} a^i = \frac{a^{n+1}-1}{a-1} - 1 \in \Theta(a^n)$ per $a > 1$.

2.4. Analisi di complessità in pratica

Dato un algoritmo A e detta $t_A(n)$ la sua complessità al caso pessimo, si cercano limiti asintotici superiori e/o inferiori a $t_A(n)$.

2.4.1. Limite superiore (upper bound) - definizione

> $$t_A(n) \in O(f(n))$$

Si prova argomentando che per ogni n "abbastanza grande" e per ciascuna istanza di taglia n l'algoritmo esegue *$\leq c \cdot f(n)$ operazioni*, con c costante (e che non serve determinare).

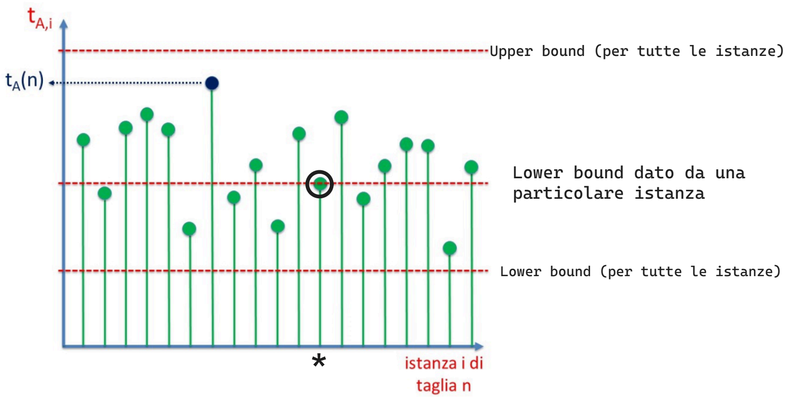
2.4.2. Limite inferiore (lower bound) - definizione

> $$t_A(n) \in \Omega(f(n))$$

Si prova argomentando che per ogni n "abbastanza grande", *esiste* un'istanza di taglia n per la quale l'algoritmo esegue *$\geq c \cdot f(n)$ operazioni*, con c costante (e che non serve determinare).
In alcuni casi è comodo argomentare che per *ciascuna* istanza di taglia n l'algoritmo esegue $\geq c \cdot f(n)$ operazioni.

> ⚠ **Attenzione**
>
> Sia che si provi $t_A(n) \in O(f(n))$ o che si provi $t_A(n) \in \Omega(f(n))$
>
> - $f(n)$ deve essere più vicino possibile alla complessità vera (*tight bound*)
> - $f(n)$ deve essere più semplice possibile, quindi senza costanti e termini additivi di ordine inferiore, solo con i *termini essenziali*!

2.4.3. Limiti superiori e inferiori



2.4.4. Terminologia per complessità

> - logaritmica: $\Theta(\log n)$, base 2 o costante > 1
> - lineare: $\Theta(n)$
> - quadratica: $\Theta(n^2)$
> - cubica: $\Theta(n^3)$
> - polinomiale $\Theta(n^c)$, $c > 0$ costante
> - esponenziale: $\Omega(a^n)$, $a > 1$ costante
> - polilogaritmica: $\Theta((\log n)^c)$, $c > 0$ costante

2.4.5. Esempio (prefix averages)

Si consideri il seguente problema computazionale.
Dato un array di n interi $X[0 \ldots n-1]$ calcolare un array $A[0 \ldots n-1]$ dove $A[i] = \left(\sum_{j=0}^{i} X[j]\right)\frac{1}{i+1}$, per $0 \leq i < n$.

Vedremo adesso due algoritmi di cui uno banale e inefficiente, poi uno più furbo ed efficiente. Per entrambi gli algoritmi vale la seguente specifica di input-output:
Input: $X[0 \ldots n-1]$ array di n interi.
Output: $A[0 \ldots n-1] : A[i] = \left(\sum_{j=0}^{i} X[j]\right)\frac{1}{i+1}$ per $0 \leq i < n$.

2.4.5.1. Algoritmo inefficiente

Algoritmo `prefixAverages1`

```
for i <- 0 to n-1 di{
    a <- 0;
    for j <- 0 to i do {
        a <- a+X[j];
    }
    A[i] <- a/(i+1);
```

```
}
return A
```

- Fuori dal for esterno: $\Theta(1)$ operazioni
- Per ciascuna iterazione i del for esterno ($i = 0, \ldots, n-1$):
  - $\Theta(i+1)$ operazioni nel for interno
  - $\Theta(1)$ altre operazioni

La complessità è quindi:
$t_{pA1}(n) \in \Theta\left(1 + \sum_{i=0}^{n-1} i + 1\right) = \Theta\left(\sum_{i=0}^{n-1} i\right) = \Theta\left(\frac{(n-1)n}{2}\right) = \Theta(n^2)$.

2.4.5.2. Algoritmo efficiente

Algoritmo `prefixAverages2`

```
s <- 0;
for i <- 0 to n-1 do{
     s <- s + X[i];
     A[i] <- s/(i+1);
}
return A
```

- Fuori dal for: $\Theta(1)$ operazioni
- Iterazioni i del for ($i = 0, \ldots, n-1$): $\Theta(1)$ operazioni

La complessità è quindi: $t_{pA2}(n) \in \Theta\left(1 + \sum_{i=0}^{n-1} 1\right) = \Theta(n)$

Possiamo affermare che $t_{pA2}(n) \in o(t_{pA1}(n))$, cioè è migliore.