Proprietà delle notazioni asintotiche

Mercoledì 8 marzo 2023

- Punto della situazione
  - Cos'è un algoritmo
  - Tempo di esecuzione $T(n)$
  - Analisi di algoritmi: analisi asintotica di $T(n)$
  - Notazioni asintotiche O, Ω, Θ
  -
- Argomento di oggi
  - Notazioni asintotiche o, ω
- Motivazioni
  - Confrontare tempi di esecuzione di algoritmi fra loro o con funzioni standard (lineare, polinomiale, esponenziale…)

Notazioni asintotiche

Nell'analisi **asintotica** analizziamo T(n)

1. A meno di costanti moltiplicative (**perché non quantificabili**)
2. Asintoticamente (per considerare input di taglia arbitrariamente grande, quindi in numero infinito)

Le notazioni asintotiche:

$$O,\ \Omega,\ \Theta,\ o,\ \omega$$

ci permetteranno il **confronto** tra funzioni, mantenendo queste caratteristiche.

Idea di fondo: O, Ω , Θ, o, ω rappresentano rispettivamente

$$\leq\ ,\ \geq\ ,\ =,\ <,\ >$$

in un'analisi <u>asintotica</u>

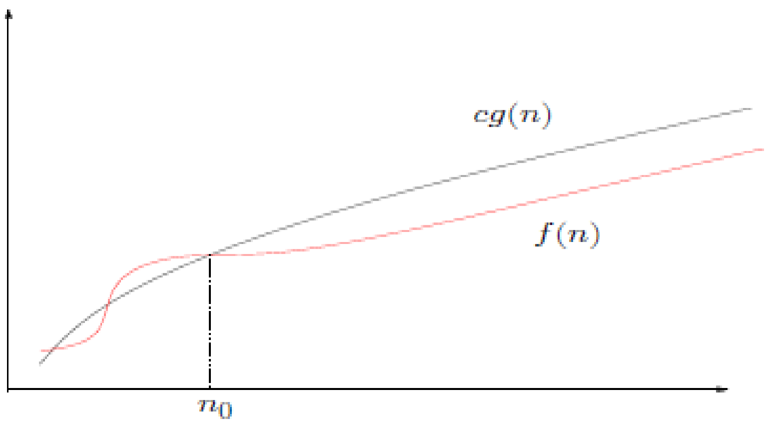
Notazioni Asintotiche: notazione O

Date $f : n \in N \rightarrow f(n) \in R_+$, $g : n \in N \rightarrow g(n) \in R_+$, scriveremo

$$\boxed{f(n) = O(g(n))}$$

$$\Leftrightarrow \exists c > 0,\ \exists n_0 \text{ tale che } f(n) \leq cg(n),\ \forall n \geq n_0$$

Informalmente, $f(n) = O(g(n))$ se $f(n)$ **non** cresce più velocemente di $g(n)$. Graficamente



Notazione asintotica Ω

Notazione duale di O:

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

se esistono costanti $c > 0$, $n_0 \geq 0$ tali che per ogni $n \geq n_0$

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

$$\boxed{f(n) = \Omega(g(n)) \text{ se e solo se } g(n) = O(f(n))}$$

Notazioni Asintotiche: notazione Θ

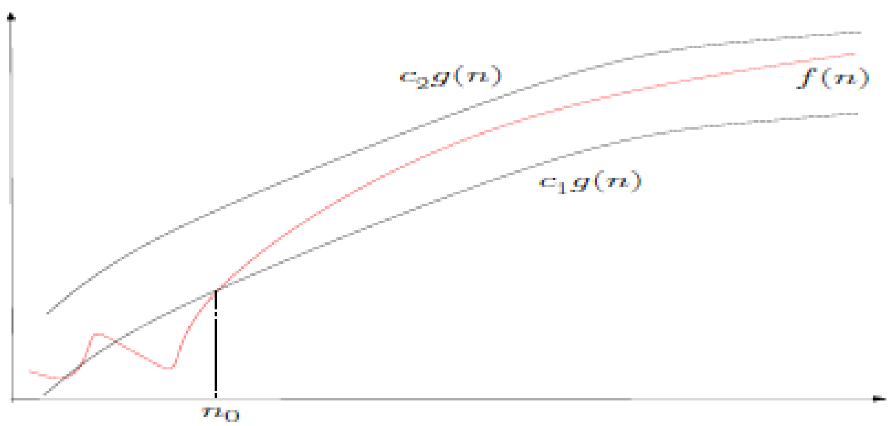
Date $f : n \in N \rightarrow f(n) \in R_+$, $g : n \in N \rightarrow g(n) \in R_+$, scriveremo

$$\boxed{f(n) = \Theta(g(n))}$$

$$\Leftrightarrow \exists n_0, c_1, c_2 > 0 \ : \ c_1 g(n) \leq f(n) \leq c_2 g(n), \ \forall n \geq n_0$$

Equivalentemente

$$f(n) = \Theta(g(n)) \ \Leftrightarrow \ f(n) = O(g(n)) \text{ e } f(n) = \Omega(g(n))$$



Notazione Θ

- Date due funzioni $f(n)$ scriveremo

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

se $f(n)$ non cresce più velocemente di $g(n)$

- Scriveremo invece

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

se $f(n)$ cresce almeno tanto velocemente di $g(n)$

- Scriveremo infine

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

se $f(n)$ e $g(n)$ **crescono allo stesso modo**

ovvero hanno lo stesso ordine di infinito

Esempio

$T(n) = 3n^2 - n$

$T(n) = \Omega(n^2)$ perchè $\exists$ c, n_0, t.c. $3n^2 - n \geq c\, n^2$ per ogni $n \geq n_0$

Infatti: $3n^2 - n \geq 2\, n^2$ sse $n^2 - n \geq 0$, sse $n - 1 \geq 0$ sse $n \geq 1$
quindi posso scegliere $c = 2$, $n_0 = 1$

Oppure: $3n^2 - n \geq n^2$ sse $2n^2 - n \geq 0$, sse $2n - 1 \geq 0$ sse $n \geq 1/2$
quindi posso scegliere $c = 1$, $n_0 = 1$

Però: $3n^2 - n \geq 3n^2$ MAI:
non esiste nessun n_0 per cui possa scegliere la coppia $c = 3$, n_0

Invece: $3n^2 - n \leq 3n^2$ per ogni $n \geq 0$. Da cui $T(n) = O(n^2)$ e infine $T(n) = \Theta(n^2)$

Nota: se $T(n) = \Theta(f(n))$, esistono c_1, c_2, n_0, t.c. $c_1 f(n) \leq T(n) \leq c_2 f(n)$ per ogni $n \geq n_0$.
Esistono quindi dei valori di c per cui $T(n) \geq c\, f(n)$ e dei valori di c per cui $T(n) \leq c f(n)$ (per opportuni valori di n)

Notazione «o piccolo»

Per indicare che $f(n)=O(g(n))$ ma $f(n) \neq \Theta(g(n))$ si scrive:

$$f(n)=o(g(n)) \quad \text{(o «piccolo»)}$$

Equivale a dire che:

per ogni costante $c > 0$, esiste $n_c \geq 0$, tale che

$$f(n) \leq cg(n) \text{ per ogni } n \geq n_c$$

Esempio 1: $n = o(n^2)$

Per ogni valore di c,

$n \leq c\, n^2$ non appena $c\, n^2 \geq n$, $c\, n \geq 1$, $n \geq 1/c$.

Quindi fissato c basterà scegliere come n_c un intero $n_c \geq 1/c$.
Per $c=2$, sceglierò un intero $n_2 \geq 1/2$, $n_2 = 1$.
Per $c=1/10$, sceglierò un intero $n_c \geq 10$, $n_c = 10$.
etc.

Notazione «o piccolo» (continua)

Per indicare che $f(n)=O(g(n))$ ma $f(n) \neq \Theta(g(n))$ si scrive:

$f(n)=o(g(n))$ (o «piccolo»)

Equivale a dire che:

per ogni costante $c > 0$, esiste $n_c \geq 0$, tale che

$f(n) \leq c\, g(n)$ per ogni $n \geq n_c$.

Esempio 2: $2n+1=o(n)$?

Dimostro che: $2n+1=O(n)$.

$2n+1 \leq 2n+n=3n$ quindi $2n+1 \leq c\,(n)$ é vera con $c=3$, $n_0=1$.

Ma $2n+1 \leq c\,(n)$ non é vera per ogni valore di c:

per $c=1, 2$, non esistono valori di n_0 opportuni.

Infatti $2n+1= \Theta(n)$

Notazione «omega piccolo»

Notazione duale di «o piccolo»:

$$f(n) = \omega\,(g(n))$$

$f(n)=\Omega(g(n))$ ma $f(n) \neq \Theta\,(g(n))$
ovvero
per ogni costante $c > 0$, esiste $n_c \geq 0$, tale che
$f(n) \geq c\, g(n)$ per ogni $n \geq n_c$.

$$f(n) = \omega\,(g(n)) \text{ se e solo se } g(n) = o(f(n))$$

In termini di analisi …. matematica

- se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \neq 0$ allora

  $f(n) = O(g(n))$ e $g(n) = O(f(n))$ (ovvero $f(n) = \Theta(g(n))$)

- se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ allora

  $f(n) = O(g(n))$ ma $g(n) \neq O(f(n))$ (ovvero *f(n)=o(g(n))*)

- se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ allora

  $f(n) \neq O(g(n))$ ma $g(n) = O(f(n))$ (ovvero *g(n)=o(f(n))*)

Esempio

Sia $f(n)=\log n$ e $g(n)=n^2$

Usando la regola di de l'Hôpital:

$$\lim_{n\to\infty} \frac{\log n}{n^2} = \lim_{n\to\infty} \frac{\frac{1}{n}}{2n} = \lim_{n\to\infty} \frac{1}{2\,n^2} = 0$$

Da cui $\log n = o(\, n^2 \,)$

Esempio

In generale, sia $f(n)=\log n$ e $g(n)=n^d$ con d reale diverso da 0.

Usando la regola di de l'Hôpital:

$$\lim_{n\to\infty} \frac{\log n}{n^d} = \lim_{n\to\infty} \frac{\frac{1}{n}}{d\, n^{d-1}} = \lim_{n\to\infty} \frac{1}{d\, n^d} = 0$$

Da cui $\log n = o(\, n^d\,)$

Notation

- Slight abuse of notation. $T(n) = O(f(n))$.
  - Asymmetric:
    - $f(n) = 5n^3$; $g(n) = 3n^2$
    - $f(n) = O(n^3) = g(n)$
    - but $f(n) \neq g(n)$.
  - Better notation: $T(n) \in O(f(n))$.

Mettendo in evidenza che c'è una classe di funzioni $O(f(n))$.

- Meaningless statement: "Any comparison-based sorting algorithm requires at least $O(n \log n)$ comparisons".
  - Use Ω for lower bounds.

Analisi di T(n)

Analizzare il tempo di esecuzione T(n) di un algoritmo significherà dimostrare che:

$T(n)= \Theta(f(n))$ se possibile

oppure

delimitare T(n) in un intervallo:

$T(n)=O(f(n))$ e $T(n)= \Omega(g(n))$

(nel caso in cui il caso peggiore sia diverso dal caso migliore).

Limitazioni più utilizzate

Scaletta:
Man mano che si scende troviamo funzioni che crescono **più** velocemente (in senso stretto):

ogni funzione f(n) della scaletta è
$f(n)=o(g(n))$
per ogni funzione che sta più in basso.

Quindi potremo utilizzare (negli esercizi) che per queste funzioni standard:

$f(n) \leq c\, g(n)$ per qualsiasi valore di c, ci possa servire, da un opportuno n_c in poi.

Espressione O	nome
$O(1)$	costante
$O(\log \log n)$	$\log \log$
$O(\log n)$	logaritmico
$O(\sqrt[c]{n}),\ c > 1$	sublineare
$O(n)$	lineare
$O(n \log n)$	$n \log n$
$O(n^2)$	quadratico
$O(n^3)$	cubico
$O(n^k)\ (k \geq 1)$	polinomiale
$O(a^n)\ (a > 1)$	esponenziale
$O(n!)$	fattoriale

Asymptotic Bounds for Some Common Functions

- Polynomials. $a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$.
- Polynomial time. Running time is $O(n^d)$ for some constant d independent of the input size n.
- Logarithms. $\log_a n = \Theta(\log_b n)$ for any constants $a, b > 0$.

  can avoid specifying the base

- Logarithms. For every $x > 0$, $\log n = o(n^x)$.

  log grows slower than every polynomial

- Exponentials. For every $r > 1$ and every $d > 0$, $n^d = o(r^n)$.

  every exponential grows faster than every polynomial

Più in dettaglio

Informalmente....

- Un esponenziale cresce più velocemente di qualsiasi polinomio

- Un polinomio cresce più velocemente di qualsiasi potenza di logaritmo

Più precisamente:

$n^d = o(r^n)$ per ogni d>0 e r>1

$\log_b n^k = o(n^d)$
per ogni k, d>0 e b>1

(vedi dimostrazione prima)

E ancora

Informalmente….

Per esempio:

- Nel confronto fra esponenziali conta la base

$$2^n = o(3^n)$$

- Nel confronto fra polinomi conta il grado

$$n^2 = o(n^3)$$

- Nel confronto fra logaritmi … la base non conta

$$\log_{10} n = \log_2 n\, (\log_{10} 2) = \Theta(\log_2 n)$$

Algoritmo della ricerca del massimo

- $T(n) \le (c_3 + 2c_2 + c_1)\, n + (c_1 - c_3 - c_2)$

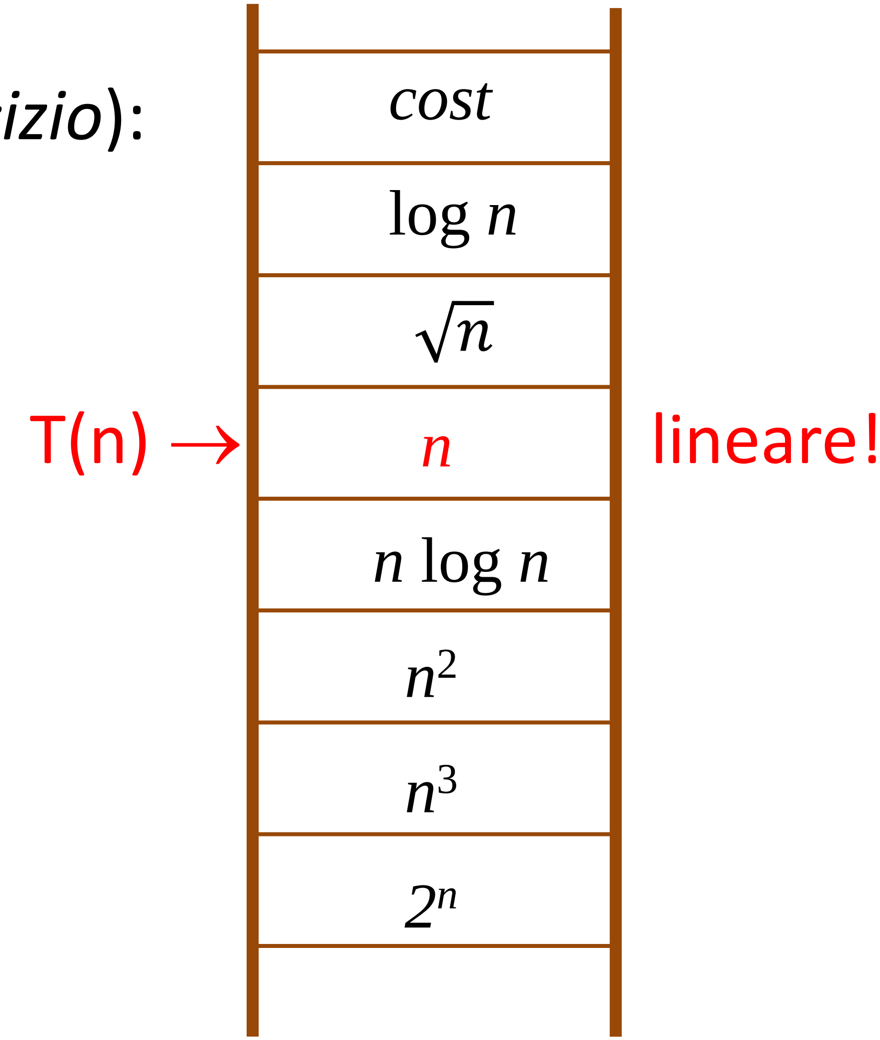
  $T(n) = O(n)$

- D'altronde (*verifica per esercizio*):

  $T(n) \ge n\, c_2$

  $T(n) = \Omega(n)$

- Da cui: $T(n) = \Theta(n)$



Algoritmo QuickSort

- Nel caso peggiore è quadratico:

  $T(n) = O(n^2)$

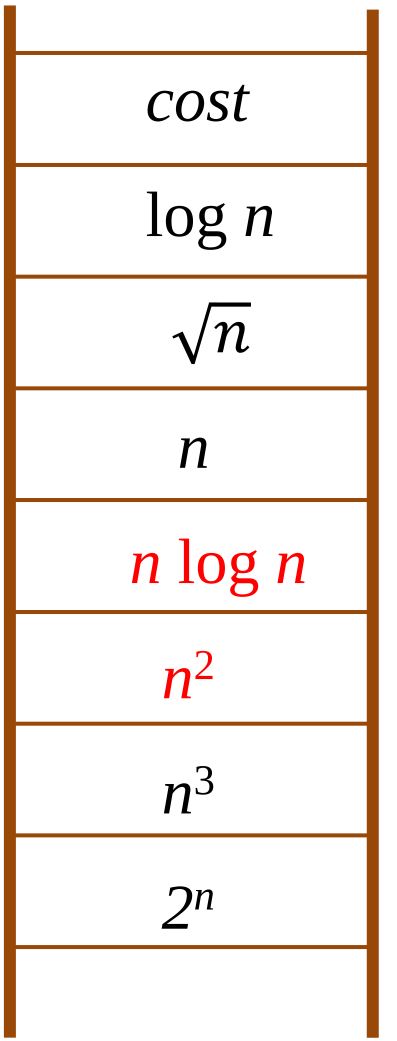
- Nel caso migliore è :

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

- Da cui: $T(n) \neq \Theta(f(n))$

  per qualsiasi f(n)

$T(n) \rightarrow$



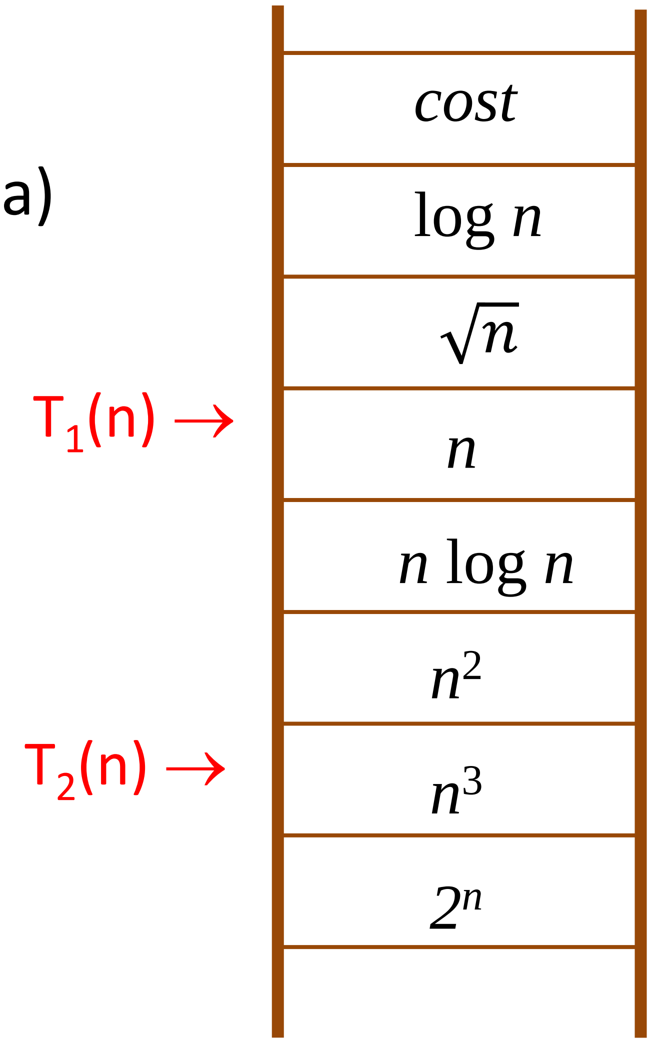
Confronto crescita asintotica funzioni

Da dimostrare in seguito …

$T_1(n)$ vs $T_2(n)$

- $T_1(n) = \Theta(f(n))$
- $T_2(n) = \Theta(g(n))$
- Con f(n) e g(n) standard (della scaletta)

  e $f(n)=o(g(n))$ allora:



$$T_1(n) = o(T_2(n))$$

Nella pratica

Per stabilire l’ordine di crescita di una funzione basterà tenere ben presente la «scaletta» e alcune proprietà delle notazioni asintotiche.

Properties

Da dimostrare in seguito …

- Transitivity

  (analoga ad a $\leq$b e b $\leq$c allora a $\leq$c per i numeri)

  - If f = O(g) and g = O(h) then f = O(h).
  - If f = Ω(g) and g = Ω(h) then f = Ω(h).
  - If f = Θ(g) and g = Θ(h) then f = Θ(h).

- Additivity.
  - If f = O(h) and g = O(h) then f + g = O(h).
  - If f = Ω(h) and g = Ω(h) then f + g = Ω(h).
  - If f = Θ(h) and g = Θ(h) then f + g = Θ(h).

  (Attenzione: l'analoga per i numeri sarebbe
  "se a $\leq$c e b $\leq$c allora a+b $\leq$c", che non è vera!!)

Applicazioni

Le proprietà (se vere) ci dicono che:

- Dato che $\log_2(2n+1) = O(n+5)$ e $n+5=O(n^2)$ allora $\log_2(2n+1)=O(n^2)$ (transitività)
- Dato che $n^2 - 2n +1 = O(n^2)$ e $\log_2(2n+1)=O(n^2)$ allora $n^2 - 2n +1 + \log_2(2n+1)=O(n^2)$ (additività)
- etc etc.....

Due regole fondamentali

Da dimostrare in seguito …

Nel determinare l'ordine di crescita asintotica di una funzione

1. Possiamo trascurare i termini additivi di ordine inferiore

2. Possiamo trascurare le costanti moltiplicative

ATTENZIONE!
Le regole NON servono però per determinare esplicitamente le costanti c ed n_0.

Applicazioni

Le due regole (se vere) ci dicono che:

$3n + \log_{10} n = \Theta(n)$

$\log^2 n + 234\sqrt{n^3} + 100 = \Theta(\sqrt{n^3})$

$3 \times 2^n + \sqrt{3^n} = \Theta(2^n)$

etc etc…..

Per stabilire la crescita di una funzione

Basterà usare:

- La «scaletta»
- Le proprietà di additività e transitività
- Le due regole fondamentali

Esercizio 1

Vero o Falso?

- $3n^5 - 16n + 2 = O(n^5)$?
- $3n^5 - 16n + 2 = O(n)$?
- $3n^5 - 16n + 2 = O(n^{17})$?
- $3n^5 - 16n + 2 = \Omega(n^5)$?
- $3n^5 - 16n + 2 = \Omega(n)$?
- $3n^5 - 16n + 2 = \Omega(n^{17})$?
- $3n^5 - 16n + 2 = \Theta(n^5)$?
- $3n^5 - 16n + 2 = \Theta(n)$?
- $3n^5 - 16n + 2 = \Theta(n^{17})$?

Esercizio 1bis

Dimostrare che

$$3n^5 - 16n + 2 = \Omega(n^5)$$

per definizione, cioè mostrando dei possibili valori per le costanti c ed n_0.

Esercizio 2

Per ciascuna delle seguenti coppie di funzioni $f(n)$ e $g(n)$, dire se $f(n) = O(g(n))$, oppure se $g(n) = O(f(n))$.

- $f(n) = (n^2 - n)/2, \qquad g(n) = 6n$
- $f(n) = n + 2\sqrt{n}, \qquad g(n) = n^2$
- $f(n) = n + \log n, \qquad g(n) = n\sqrt{n}$
- $f(n) = n^2 + 3n, \qquad g(n) = n^3$
- $f(n) = n \log n, \qquad g(n) = n\sqrt{n}/2$
- $f(n) = n + \log n, \qquad g(n) = \sqrt{n}$
- $f(n) = 2(\log n)^2, \qquad g(n) = \log n + 1$
- $f(n) = 4n \log n + n, \qquad g(n) = (n^2 - n)/2$
- $f(n) = (n^2 + 2)/(1 + 2^{-n}), \qquad g(n) = n + 3$
- $f(n) = n + n\sqrt{n}, \qquad g(n) = 4n \log(n^3 + 1)$

NOTA: Esistono anche funzioni (particolari) non confrontabili tramite O

Esercizio 3

Date le seguenti funzioni

$$\log n^5, n^{\log n}, \log^2 n, 10\sqrt{n}, (\log n)^n, n^n, n \log \sqrt{n},$$
$$n \log^3 n, n^2 \log n, \sqrt{n \log n}, 10 \log \log n, 3 \log n,$$

ordinarle scrivendole da sinistra a destra in modo tale che la funzione $f(n)$ venga posta a sinistra della funzione $g(n)$ se $f(n) = O(g(n))$.

Esercizio 4

- Esercizio 2 dell'appello 26 gennaio 2010

Si supponga di avere due algoritmi A ed A' che risolvono il medesimo problema in tempo $T_A(n)$ e $T_{A'}(n)$ rispettivamente. Se $T_A(n) = 5n^2 + 11\log n$ e $T_{A'}(n) = \sqrt(n^3)logn + n$, quale dei due algoritmi e' asintoticamente piu' efficiente in termini di tempo? E' necessario giustificare la risposta.

Esercizi «per casa»

- Esercizi dalle slides precedenti
- Es. 3, 4, 5 e 6 di pagg. 67-68 del libro [KT]
- Dalla piattaforma: Esercizi_O_2010.pdf

Appello 29 gennaio 2015

Quesito 2 (24 punti)
Dopo la Laurea in Informatica avete aperto un campo di calcetto che ha tantissime richieste e siete diventati ricchissimi. Ciò nonostante volete guadagnare sempre di più, per cui avete organizzato una sorta di asta: chiunque volesse affittare il vostro campo (purtroppo è uno solo), oltre ad indicare da che ora a che ora lo vorrebbe utilizzare, deve dire anche quanto sia disposto a pagare. Il vostro problema è quindi scegliere le richieste compatibili per orario, che vi diano il guadagno totale maggiore.
Formalizzate il problema reale in un problema computazionale.

DEFINIRE PROBLEMA COMPUTAZIONALE

Quesito (22 punti) (*Campi di calcetto*)
Dopo il successo del vostro primo campo di calcetto, avete aperto molti altri campi di calcetto, all'interno di un unico complesso. Ogni giorno raccogliete le richieste per utilizzare i vostri campi, ognuna specificata da un orario di inizio e un orario di fine. Oramai avete un numero di campi sufficiente ad accontentare sempre tutte le richieste. Volete però organizzare le partite nei campi in modo da accontentare tutti, senza che vi siano sovrapposizioni di orari, ma con il minimo numero possibile di campi (la manutenzione costa!). Formalizzate il problema reale in un problema computazionale.