



Notazioni asintotiche

Venerdì 4 marzo 2022

- Punto della situazione
 - Cos'è un algoritmo
 - Tempo di esecuzione $T(n)$
 - Quando un algoritmo è efficiente
- Obiettivo:
 - Confrontare tempi di esecuzione di algoritmi fra loro o con funzioni standard (lineare, polinomiale, esponenziale...)
 - Pseudo-codice \Rightarrow costanti non quantificate
 \Rightarrow analisi asintotica di $T(n)$
- Argomento di oggi:
 - Notazioni asintotiche: strumento per confronto crescita di funzioni

Tempo di esecuzione

Tempo di esecuzione $T(n)$ sarà misurato in termini del **numero di operazioni elementari** per eseguire l'algoritmo su un input di taglia n

Assegnamento, incremento, confronto sono considerate **operazioni elementari** all'interno dell'algoritmo della ricerca del massimo.

Richiedono tempo **costante** (= non dipendente dalla taglia n dell'input) ma a priori non quantificabile

Il tempo di esecuzione di un algoritmo sarà calcolato a partire dalle operazioni elementari seguendo la **struttura** dell'algoritmo e applicando delle semplici **regole**.

(*Esempi*: ricerca del massimo, Insertion-Sort su [CLRS], tutti gli algoritmi che vedremo)

Funzioni $T(n)$

Se $T(n)$ rappresenta un tempo di esecuzione su un input di taglia n , allora:

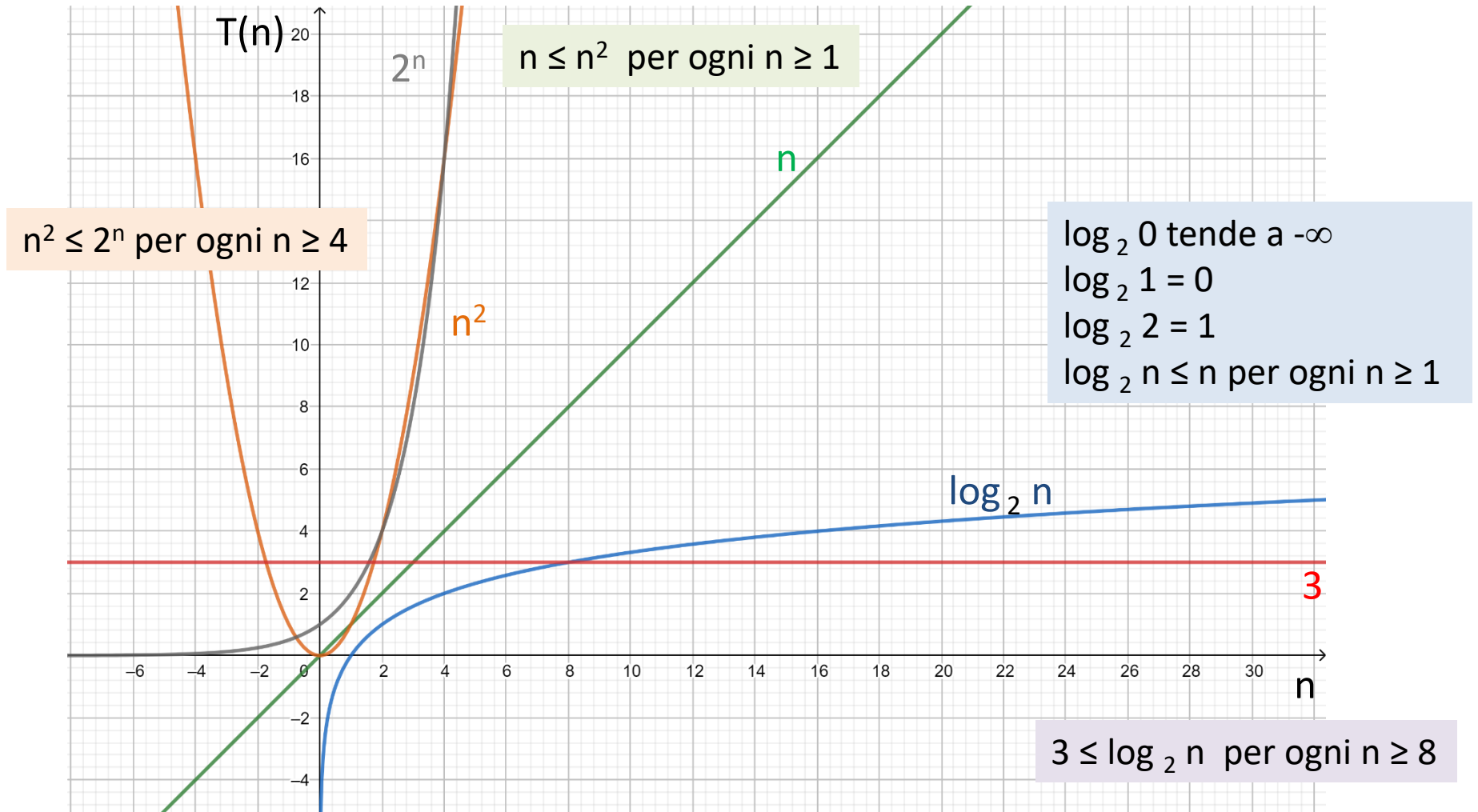
n è un intero positivo

$T(n)$ è un reale positivo

$$T: \mathbb{N} \rightarrow \mathbb{R}_+$$

Inoltre $T(n)$ è una funzione **non decrescente**

Funzioni standard



Analisi asintotica

Vogliamo analizzare l'efficienza dell'algoritmo

- Indipendentemente da implementazione, hardware etc.
- Al crescere della taglia dell'input

Studieremo il tempo in funzione della taglia dell'input : $T(n)$

Studieremo la crescita della funzione $T(n)$ al crescere di n

«asintotica»:

- per n arbitrariamente grande
- per n che tende a infinito (*se conosciamo i limiti...*)
- da un certo punto in poi
- per ogni $n \geq n_0$ (*altrimenti...*)

Caso peggiore, migliore, medio

A volte il tempo di esecuzione di un algoritmo, non dipende soltanto dalla taglia. Anche fissata la taglia, l'algoritmo può avere un diverso tempo di esecuzione, a seconda della distribuzione dell'input.

Esempio: InsertionSort su n elementi impiega un numero lineare in n di confronti se gli elementi si presentano già in ordine, invece un numero quadratico in n , se si presentano nell'ordine inverso.

Analisi del **caso peggiore**: qualunque sia la distribuzione dell'input $T(n)$ è limitata **superiormente** da $f(n)$

Analisi del **caso migliore**: qualunque sia la distribuzione dell'input $T(n)$ è limitata **inferiormente** da $g(n)$ (poco significativa)

Analisi del **caso medio**: nel caso di una distribuzione media o random (difficile da determinare)

Notazioni asintotiche

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

1. A meno di costanti moltiplicative
2. Asintoticamente

Le notazioni asintotiche:

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

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

Idea di fondo: $O, \Omega, \Theta, o, \omega$ rappresentano rispettivamente

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

in un'analisi asintotica

Nota: le notazioni asintotiche ci **semplificheranno** l'analisi del tempo di esecuzione perché non dovremo più badare al valore esatto di certe costanti!

Analisi asintotica di $T(n)$

Obiettivo:

- Trovare una **limitazione superiore** alla crescita di $T(n)$:
 $T(n) = O(g(n))$ significa che il tempo di esecuzione, anche nel caso peggiore, è limitato superiormente da $g(n)$
- Trovare una **limitazione inferiore** alla crescita di $T(n)$:
 $T(n) = \Omega(g(n))$ significa che il tempo di esecuzione, anche nel caso migliore, è limitato inferiormente da $g(n)$
- Determinare l'esatto ordine di crescita di $T(n)$, se possibile:
 $T(n) = \Theta(g(n))$ significa che nel caso peggiore è $O(g(n))$ e nel caso migliore è $\Omega(g(n))$ (in pratica non vi è distinzione fra tempo di esecuzione nel caso peggiore e migliore)

Adesso **definiamo** la notazione **O** .

Tutte le altre potranno essere definite **di conseguenza**.

In termini di analisi matematica

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

$$f(n) = O(g(n)) \quad \text{e} \quad g(n) = O(f(n)) \quad (\text{ovvero } f(n) = \Theta(g(n)))$$

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

$$f(n) = O(g(n)) \quad \text{ma} \quad g(n) \neq O(f(n)) \quad (\text{ovvero } f(n) = o(g(n)))$$

● se $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$ allora

$$f(n) \neq O(g(n)) \quad \text{ma} \quad g(n) = O(f(n)) \quad (\text{ovvero } g(n) = o(f(n)))$$

Vediamo ora la definizione per «informatici»

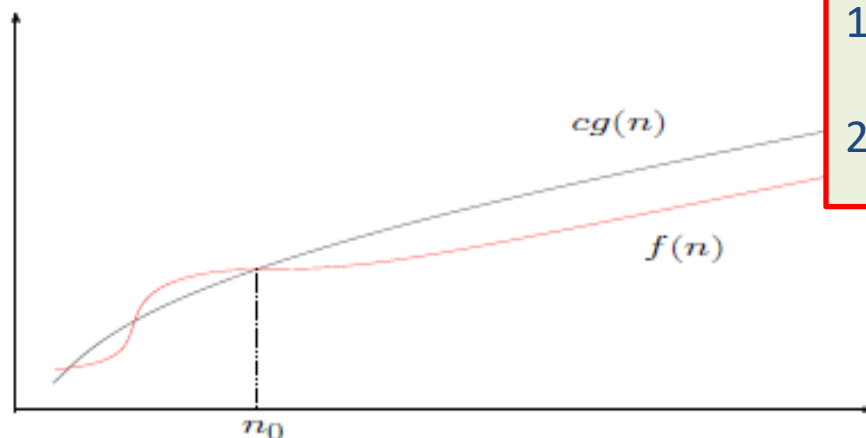
Notazioni Asintotiche: notazione O

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

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

$\Leftrightarrow \exists c > 0, \exists n_0$ 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



1. A meno di costanti moltiplicative
2. Asintoticamente

Esempi

• $10n^3 + 2n^2 + 7 = O(n^3)$

Occorre provare che

$$\exists c, n_0 : 10n^3 + 2n^2 + 7 \leq cn^3, \forall n \geq n_0$$

Si ha: $10n^3 + 2n^2 + 7 \leq 10n^3 + 2n^3 + 7$

$7 \leq n^3$ sse $n \geq 2$ $\rightarrow \leq 10n^3 + 2n^3 + n^3 = 13n^3, \forall n \geq 2.$

Quindi la disuguaglianza è soddisfatta per $c = 13$ e $n_0 = 2$.

Un altro esempio

$$\log_2(2n+1) = O(n+5)$$

$\exists c, n_0$, t.c. $\log_2(2n+1) \leq c(n+5)$ per ogni $n \geq n_0$

Se $1 \leq n$



$$\begin{aligned}\log_2(2n+1) &\leq \log_2(2n+n) = \log_2(3n) = \\ &= \log_2(3) + \log_2(n) \leq 1,585 + \log_2 n \leq 5 + \log_2 n \leq \\ &\leq 5+n\end{aligned}$$



Se $\log_2 n \leq n$
 $n \geq 1$

$$c = 1$$

$$n_0 = 1$$

Esempio

$$n^2 - 2n + 1 = O(n^2)$$

1. (tecnica per i polinomi)

$$n^2 - 2n + 1 \leq n^2 + 2n + 1 \leq n^2 + 2n^2 + 1 \leq n^2 + 2n^2 + n^2 = 4n^2$$

$$c = 4, n_0 = 1$$

2. $n^2 - 2n + 1 \leq n^2 + 1 \leq n^2 + n^2 = 2n^2$

$$c = 2, n_0 = 1$$

3. (diseguazioni) $n^2 - 2n + 1 \leq 2n^2$ sse $n^2 + 2n - 1 \geq 0$

$$n \leq -1 - \sqrt{2} \text{ ed } n \geq -1 + \sqrt{2}$$

$$c = 2, n_0 = 1$$

4. $n^2 - 2n + 1 = (n - 1)^2 \leq n^2$

$$c = 1, n_0 = 1$$

Nota: se esiste una coppia (c, n_0) , ne esistono infinite, ma per dimostrare O ne basta esibire 1 sola.

Esempi: un polinomio è O del suo primo termine

Più in generale, possiamo provare che:

$$a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0 = O(n^k)$$

Infatti

$$\begin{aligned} a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0 & \leq |a_k| n^k + |a_{k-1}| n^{k-1} + \dots + |a_1| n + |a_0| \\ & \leq |a_k| n^k + |a_{k-1}| n^k + \dots + |a_1| n^k + |a_0| n^k \\ & = (|a_k| + |a_{k-1}| + \dots + |a_1| + |a_0|) n^k \\ & = c n^k \end{aligned}$$

$$\implies a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0 = O(n^k)$$

quindi

$$n^3 + 100n + 200 = O(n^3)$$

$$20n^3 + n^5 + 100n = O(n^5)$$

$$10n^2 + n^{5/2} + 7n = O(n^{5/2})$$

$$10n + 3n^7 + 5n^6 + 9n^3 + 34n^2 + 22n^5 + n^{8/3} + 4n^{7/2} + 23n^{11/2} = O(n^7)$$

⋮

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

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

Notazioni Asintotiche: notazione Θ

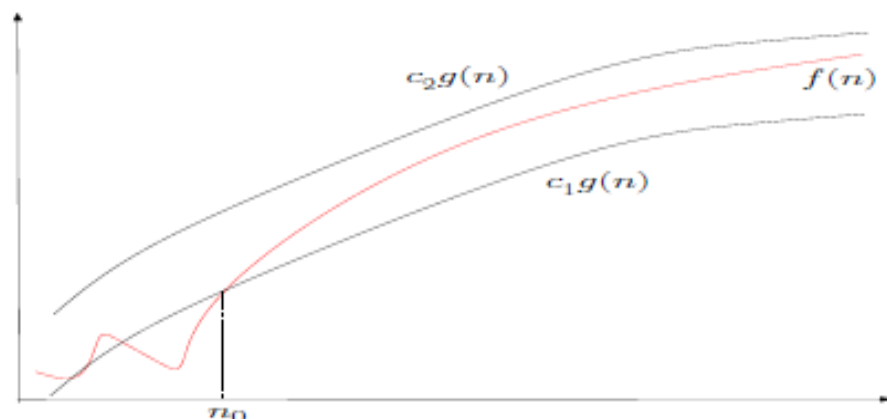
Date $f : n \in N \rightarrow f(n) \in R_+$, $g : n \in N \rightarrow g(n) \in R_+$,
scriveremo

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

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

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

$$T(n) = 32n^2 + 17n + 3$$

$$T(n) = \Omega(n^2)$$

$\exists c, n_0$, t.c. $32n^2 + 17n + 3 \geq c n^2$ per ogni $n \geq n_0$

$$32n^2 + 17n + 3 \geq 32 n^2$$

$$c = 32, n_0 = 1$$

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

Da cui $T(n) = \Theta(n^2)$

Θ limitazione esatta

- *Ex:* $T(n) = 32n^2 + 17n + 3$
 - $T(n)$ is $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, and $\Theta(n^2)$
 - $T(n)$ is **not** $O(n)$, $\Omega(n^3)$, $\Theta(n)$, or $\Theta(n^3)$

Θ limitazione esatta

- È perfettamente legittimo dire che

$$n^3 + n\sqrt{n} \log n + 10 = O(n^3),$$

ma è più preciso dire che $n^3 + n\sqrt{n} \log n + 10 = \Theta(n^3)$

- È corretto dire che $n^{\frac{1}{\log n}} = O(n)$,

ma è più preciso dire che

$$n^{\frac{1}{\log n}} = \left(2^{\log n}\right)^{\frac{1}{\log n}} = 2 = \Theta(1)$$

- È quindi una questione di precisione nel linguaggio...

Θ è una limitazione esatta (*tight bound*)

Esempio

$$\log_2 n = \Theta(\log_3 n)$$

$\exists c_1, c_2, n_0$, t.c. $c_1 \log_3 n \leq \log_2 n \leq c_2 \log_3 n$ per ogni $n \geq n_0$

$$\log_2 n \geq \log_3 n = c_1 \times \log_3 n, \text{ per } n \geq 1$$

$$\log_2 n = \log_2 3 \times \log_3 n = c_2 \times \log_3 n, \text{ per } n \geq 1$$

$$c_1=1, c_2=\log_2 3, n_0=1$$

Calcolo del tempo di esecuzione (3): esercizio

Esempio: algoritmo per la ricerca del massimo fra n numeri a_1, \dots, a_n

```
1.  max = a1
2.  For i = 2 to n
3.      If (ai > max) then
4.          max = ai
5.      Endif
6.  Endfor
```

Taglia dell'input = n

Tempo di un assegnamento = c_1
(costante = non dipende da n)

Tempo di un confronto = c_2

Tempo di un incremento = c_3

Cosa posso dire adesso?

$$T(n) \leq 2c_1 + (n-1)c_3 + nc_2 + (n-1)(c_2 + c_1) = (c_3 + 2c_2 + c_1)n + (c_1 - c_3 - c_2)$$

$$T(n) \leq An + B$$

dove A e B sono costanti non quantificabili a priori (dipendono dall'implementazione)

Quindi: $T(n) = O(n)$

Esercizio:

Analizzando l'algoritmo, riuscite a dire che $T(n) = \Omega(n)$ e quindi $T(n) = \Theta(n)$?

.... semplificata dalle notazioni asintotiche

```
1.  max = a1
2.  For i = 2 to n
3.      If (ai > max) then
4.          max = ai
5.      Endif
6.  Endfor
```

E' un **blocco** di 2 istruzioni:

1. Assegnamento
- 2-6. For

L'istruzione **For** delle linee 2-6 è:

For i=a to b {istr0} Endfor

dove **istr0** è l'istruzione if delle linee 3-5

L'istruzione **If** delle linee 3-5 è:

If(cond) then istr1 Else istr2 Endif

dove **cond** è il confronto: **a_i > max**

istr1 è l'assegnamento: **max = a_i**

istr2 è vuota

Non è
più semplice così?

Tempo esecuzione algoritmo:

c_1 + Tempo del For.

Scriveremo $O(1) + O(n) = O(n)$

Tempo esecuzione For:

$\leq c_1 + (n-1) c_3 + n c_2 + (n-1) (c_2 + c_1)$


Scriveremo $(n-1)O(1)+O(1) = O(n)$


Tempo esecuzione If:


$\leq c_2 + c_1$ scriveremo $O(1)$


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 2

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

● $f(n) = (n^2 - n)/2, \quad g(n) = 6n$

● $f(n) = n + 2\sqrt{n}, \quad g(n) = n^2$

● $f(n) = n + \log n, \quad g(n) = n\sqrt{n}$

● $f(n) = n^2 + 3n, \quad g(n) = n^3$

● $f(n) = n \log n, \quad g(n) = n\sqrt{n}/2$

● $f(n) = n + \log n, \quad g(n) = \sqrt{n}$

● $f(n) = 2(\log n)^2, \quad g(n) = \log n + 1$

● $f(n) = 4n \log n + n, \quad g(n) = (n^2 - n)/2$

● $f(n) = (n^2 + 2)/(1 + 2^{-n}), \quad g(n) = n + 3$

● $f(n) = n + n\sqrt{n}, \quad g(n) = 4n \log(n^3 + 1)$

I calcoli riusciranno più semplici dopo che avremo studiato le **proprietà** delle notazioni asintotiche

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

Esercizi analisi tempo di esecuzione

Qual è il tempo di esecuzione del seguente frammento di pseudocodice?

```
for i=1 to n/2
```

```
    if i>10 then
```

```
        x=2x
```

```
return x
```

A. $O(\log n)$

B. $\Theta(n)$

C. $\Theta(n^2)$

D. Nessuna delle risposte precedenti

Esercizi

- Formalizzazione problemi computazionali alla base di problemi reali (vedi slide lezione scorsa)

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.