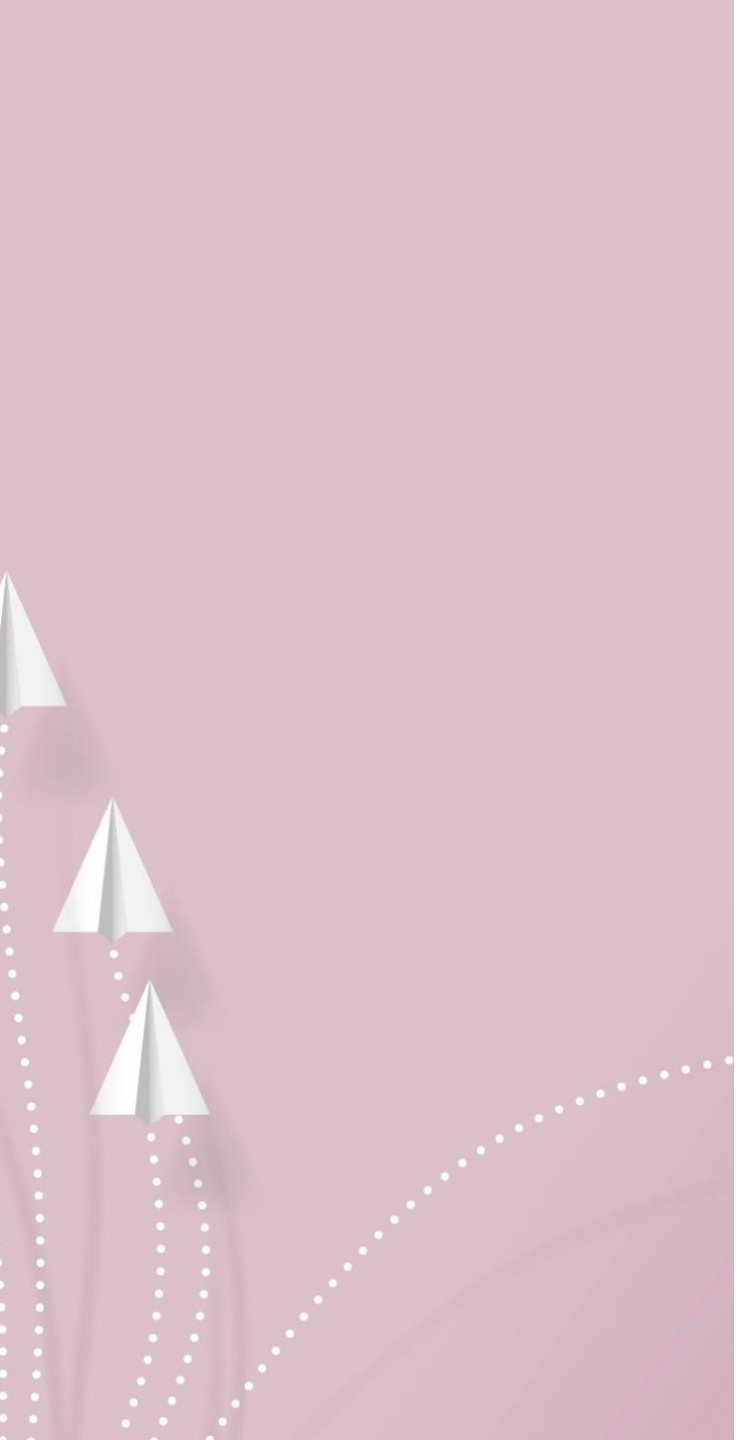


Analisi del tempo di esecuzione di algoritmi con le notazioni asintotiche

Giovedì 9 marzo 2023

Punto della situazione

- Cos'è un algoritmo
- Tempo di esecuzione T(n)
- Analisi di algoritmi: analisi asintotica di T(n)
- Notazioni asintotiche

Argomento di oggi

- Analisi del tempo di esecuzione di un algoritmo con le notazioni asintotiche

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: $O,\ \Omega\ ,\ \Theta,\ o,\ \omega$ rappresentano rispettivamente

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

in un'analisi asintotica

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).

Per stabilire la crescita di una funzione

Basterà usare:

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

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

E ancora

Informalmente….

- Nel confronto fra esponenziali conta la base
- Nel confronto fra polinomi conta il grado
- Nel confronto fra logaritmi … la base non conta

- Un polinomio cresce più velocemente di qualsiasi potenza di logaritmo

Per esempio:

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

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

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

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

Polinomi vs logaritmi

Un polinomio cresce più velocemente di qualsiasi potenza di logaritmo.
Per esempio:

- Proviamo che

$$\log_2 n = O(n).$$

Occorre provare che $\exists c, n_0 \ : \ \log_2 n \leq cn \quad \forall n \geq n_0$

Per induzione su n: Per $n = 1$ abbiamo $\log_2 1 = 0 \leq 1$.

In generale, per $n \geq 1$

$$\begin{aligned} \log_2(n+1) &\leq \log_2(n+n) = \log_2(2n) \\ &= \log_2 2 + \log_2 n = 1 + \log n \\ &\leq 1 + n \text{ (per ipotesi induttiva)} \end{aligned}$$

Abbiamo quindi provato che

$$\log n \leq n \quad \forall n \geq 1 \Longrightarrow \boxed{\log n = O(n)}$$

Lo proveremo con $c=1$ e $n_0=1$, cioè $\log_2 n \leq n, \quad \forall n \geq 1$

Domanda

Per questo genere di esercizi:

a cosa serve la calcolatrice?

Suggerimento:
ricorda che $f(n) = O(g(n))$ significa $f(n) \leq c\, g(n)$ **per ogni** $n \geq n_0$
cioè per un numero **infinito** di valori di n.

QUINDI: NON basta dimostrare che $f(n) \leq c\, g(n)$ per qualche costante c, per esempio che:
$f(1) \leq c\, g(1)$, $f(2) \leq c\, g(2)$, … , $f(10.000) \leq c\, g(10.000)$.

Perché potrebbe essere invece
$f(n) \geq c\, g(n)$ **per ogni** $n \geq 10.001$.

Properties

- 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!!)

Analogie e differenze fra notazioni asintotiche e numeri reali

Confronto fra notazioni asintotiche	Analogo confronto fra numeri reali
$a(n) = O(b(n))$	$a \leq b$
$a(n) = \Omega(b(n))$	$a \geq b$
$a(n) = \Theta(b(n))$	$a = b$
$a(n) = o(b(n))$	$a < b$
$a(n) = \omega(b(n))$	$a > b$

Differenza: due n° reali possono sempre essere confrontati mentre esistono funzioni asintoticamente non confrontabili (o incommensurabili), ad es.

- n e $n^{1+sin n}$
- $f(n) \begin{cases} n \text{ per } n \text{ dispari} \\ n^2 \text{ per } n \text{ pari} \end{cases}$ e $g(n) = \begin{cases} n \text{ per } n \text{ pari} \\ n^2 \text{ per } n \text{ dispari} \end{cases}$

Due regole fondamentali

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.

Tempo di esecuzione

Tempo di esecuzione T(n) è espresso rispetto al numero di operazioni elementari/atomiche per eseguire l'algoritmo su un input di taglia n

Sono operazioni elementari le operazioni che richiedono tempo costante (= non dipendente dalla taglia n dell'input)

Per esempio: assegnamento, incremento, confronto

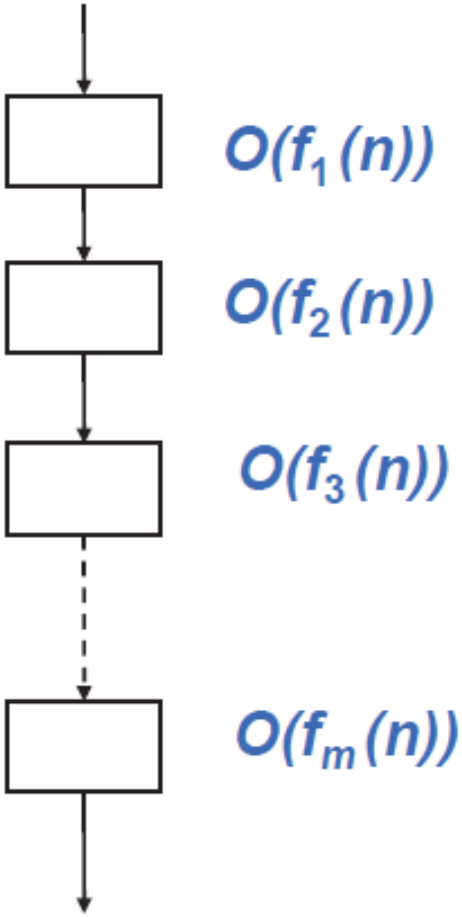
Nelle prossime slides vedremo come l'analisi asintotica può aiutarci nel calcolo del tempo di esecuzione di algoritmi di tipo iterativo (strutturati come for e while)

Operazioni Semplici

- *operazioni aritmetiche (+,*,…)*
- *operazioni logiche(&&, ||,….)*
- *confronti (≤ ,≥ , = ,…)*
- *assegnamenti* (`a = b`) senza chiamate di funzione
- *operazioni di lettura* (`read`)
- *operaioni di controllo* (`break, continue, return`)

$$T(n) = \Theta(1) \Rightarrow T(n) = O(1)$$

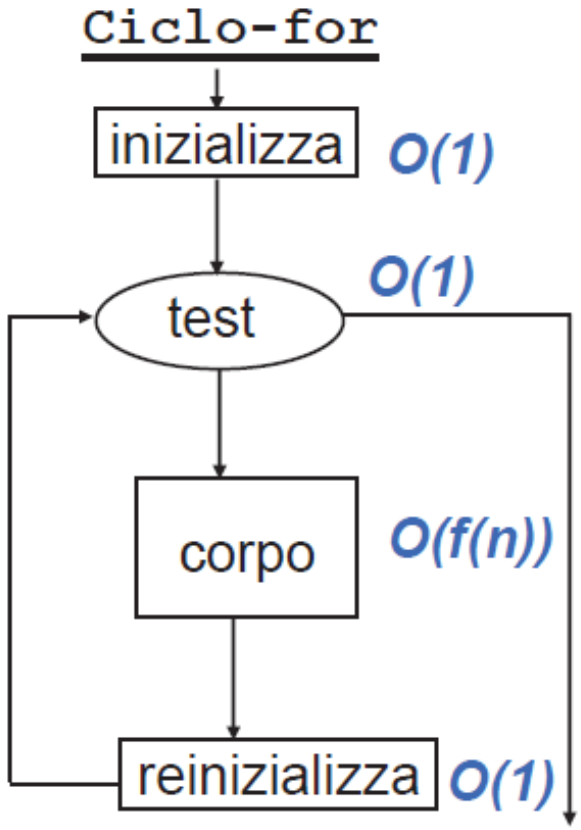
Tempo di esecuzione: blocchi sequenziali



$$O(f_1(n) + f_2(n) + \ldots + f_m(n))$$

$$O\left(\max_{i \in \{1 \ldots m\}} \{f_i(n)\}\right)$$

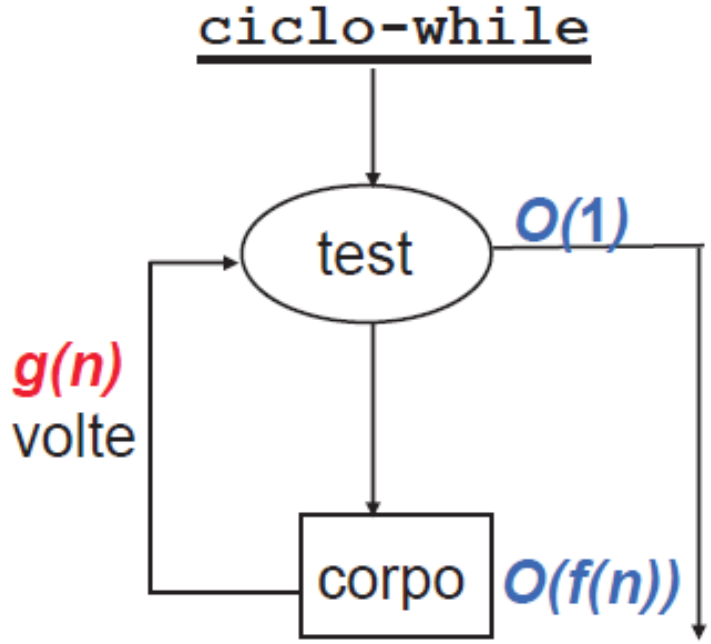
Tempo di esecuzione: ciclo for



stabilire ***g(n)*** è in genere semplice.

$$T(n) = O(g(n) \times f(n))$$

Tempo di esecuzione: ciclo while

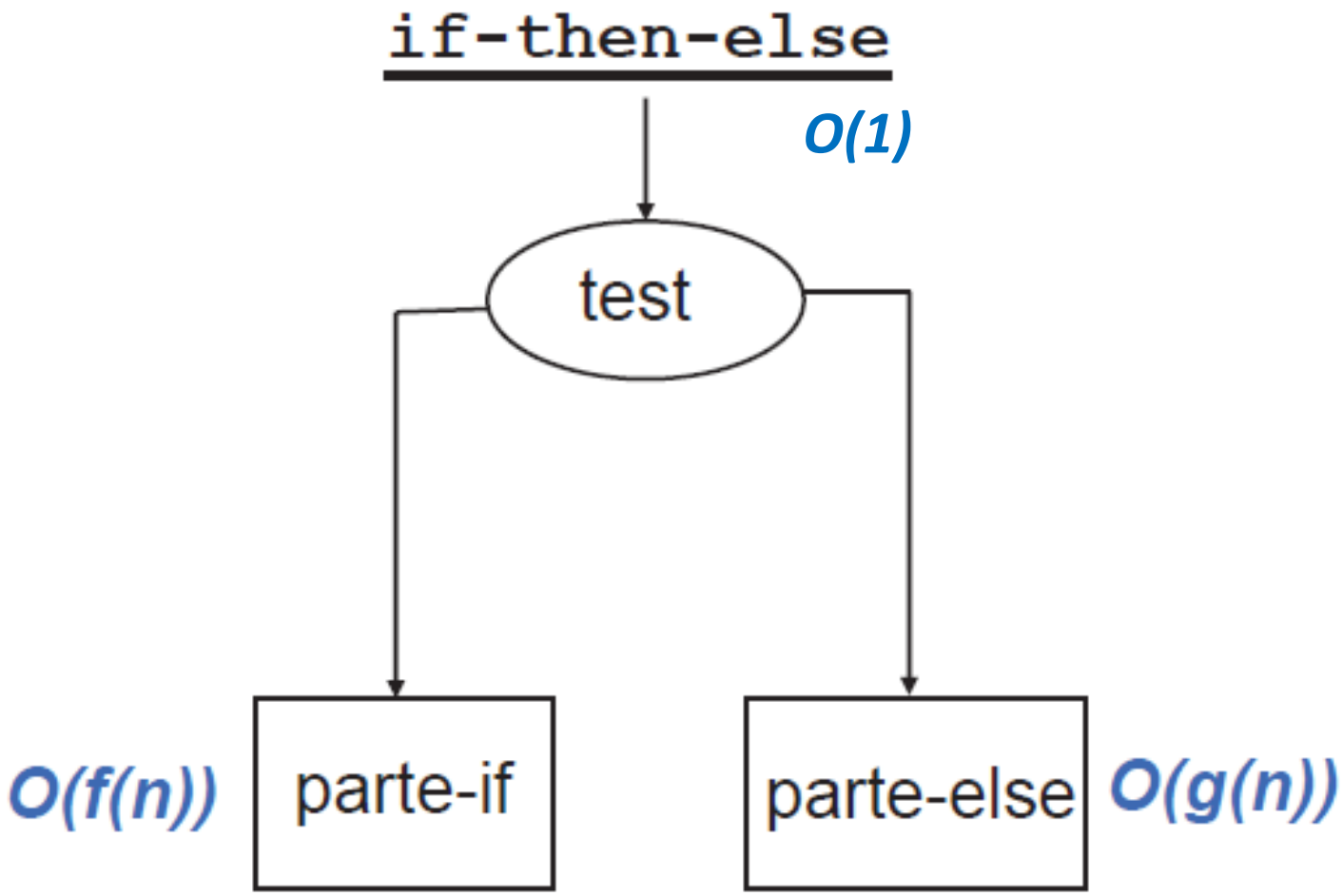


Bisogna stabilire un limite per il numero di iterazioni del ciclo, ***g(n)***.

Può essere necessaria una prova induttiva per ***g(n)***.

$$T(n) = O(g(n) \times f(n))$$

Tempo di esecuzione: If-Then-Else



Somma primi n numeri

$1 + 2 + 3 + \ldots + (n-1) + n = \sum_{i=1}^{n} i$

1 + 2 + 3 + … + (n-1) + n = ?

= 1 + 2 + 3 + … + (n-2) + (n-1) + n =

= (n+1) n/2 =

= Θ(??)

Somma quadrati (sup)

$1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2 = \Theta(??)$

$1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2 \leq$

$\leq n^2 + n^2 + n^2 + \ldots + n^2 + n^2 =$

$= (n^2)\, n = n^3$

$1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2 = O(n^3)$

D: $1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2 = \Omega(n^3)$?

Somma quadrati (inf)

$1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2 = \sum_{i=1}^{n} i^2$

$1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2 = \Omega(n^3)$?

$1^2 + 2^2 + \ldots + (n/2)^2 + \ldots + (n-1)^2 + n^2 \geq$

$\geq (n/2)^2 + \ldots + (n-1)^2 + n^2 \geq$

$\geq (n/2)^2 + \ldots + (n/2)^2 + (n/2)^2 =$

$= (n/2)^2 \, n/2 =$

$= (n^2/4) \, n/2 =$

$= 1/8 \, n^3$

$1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2 = \Omega(n^3)$

$1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2 = O(n^3)$

$1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2 = \Theta(n^3)$

Esercizi

Usando la notazione Θ, stimare il numero di volte che la istruzione $x = x + 1$ viene eseguita:

1.
for $i = 1$ **to** $2n$
  $x = x + 1$

2.
for $i = 1$ **to** $2n$
  for $j = 1$ **to** n
    $x = x + 1$

3.
for $i = 1$ **to** n
  for $j = 1$ **to** i
    for $k = 1$ **to** j
      $x = x + 1$

Esercizi

Usando la notazione Θ, stimare il numero di volte che la istruzione $x = x + 1$ viene eseguita:

1.
```
for i = 1 to 2n
    x = x + 1
```
$\Theta(n)$

2.
```
for i = 1 to 2n
    for j = 1 to n
        x = x + 1
```
$\Theta(n^2)$

3.
```
for i = 1 to n
    for j = 1 to i
        for k = 1 to j
            x = x + 1
```
Numero di volte in cui eseguo $x = x + 1$ è

$$\sum_{i=1}^{n} \sum_{j=1}^{i} j = \sum_{i=1}^{n} \frac{i(i+1)}{2} = \sum_{i=1}^{n} \Theta(i^2)$$

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

Esercizi (continua)

4.
$i = n$
while $i \geq 1$ **do**
$x = x + 1,\ i = i/2$

Il **while** è eseguito per $i = n, n/2, n/4, \ldots, n/2^k, \ldots, n/2^t = 1$ cioè per $k = 0, 1, \ldots, t$.
Si noti che il simbolo «/» indica la divisione intera (che scarta le cifre decimali).
Quindi $t = \lfloor \log_2 n \rfloor$ (dove $\lfloor\ \rfloor$ indica l'arrotondamento all'intero inferiore)
e il numero di volte in cui viene eseguito il **while** è: $\lfloor \log_2 n \rfloor + 1 = \Theta(\log_2 n)$.

Si noti che arrotondamenti del genere NON incidono nell'analisi asintotica:

$\log_2 n$
$\lfloor \log_2 n \rfloor + 1$
$\lfloor \log_2 n \rfloor - 1$

sono tutte funzioni in $\Theta(\log_2 n)$.

Esempio: InsertionSort

Algoritmo di ordinamento di A[1…n] ottenuto mantenendo ad ogni iterazione A[1…j-1] ordinato e inserendovi A[j].

```
InsertSort(array A[1…n])
   for j = 2 to n
      key = A[j]
      i = j - 1
      while i > 0 and A[i] > key
            A[i+1] = A[i]
            i = i - 1
      A[i+1] = key
```

Analisi di InsertionSort

$O(n^2) =$

```
InsertSort(array A[1…n])
    for j = 2 to n
      key = A[j]                          = O(1)
      i = j - 1                           = O(1)
      while i > 0 and A[i] > key
            A[i+1] = A[i]                 } = O(n)
            i = i - 1
      A[i+1] = key                        = O(1)
```

Più precisamente:

Fissato j, il test del while è eseguito un numero di volte fra 1 e j.
Da cui

$$T(n) \leq \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 = \frac{1}{2}n^2 + \frac{1}{2}n - 1$$

e quindi $T(n) = O(n^2)$. Inoltre $T(n) = \Omega(n)$.

Esercizio 2

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

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

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

Esercizi «per casa»

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

Tempo di esecuzione 1

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. Θ (n log n)

C. $\Theta(n^2)$

D. Nessuna delle risposte precedenti

Esercizi analisi tempo di esecuzione

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

```
for i=1 to n/2
   for j=1 to logn
      x=i*j
 return x
```

A. O(log n)

B. o(n log n)

C. $\Theta(n^2)$

D. Nessuna delle precedenti

Multiple Prima prova 2021 (Moodle)

Confronto2

Siano $\mathbf{f(n) = 4n + 2^{\log_2 n}}$ e $\mathbf{g(n) = n + \log_2 n + 1000}$. Allora

A. f(n) = o (g(n))

B. f(n) = ω (g(n))

C. f(n) = Θ (g(n))

D. nessuna

Prima prova intercorso 2020

3) (18 punti)
Indicare la corretta successione delle funzioni seguenti affinché compaiano da sinistra a destra in **ordine crescente** di crescita asintotica, **motivando** adeguatamente la successione proposta:

$$F_1(n) = 2^{(2 \log_2 n)}$$

$$F_2(n) = n^2 \sqrt{n}$$

$$F_3(n) = 2^{n+1}$$

$$F_4(n) = n^2 \log n$$

TEMPO ESECUZIONE

1) 1 ☐

Un algoritmo ha tempo di esecuzione T(n) polinomiale se:

A. $T(n) = \Theta(n^c)$ per una costante $c>0$
B. $T(n) = \Omega(n^c)$ per una costante $c>0$
C. $T(n) = O(n^c)$ per una costante $c>0$
D. Nessuna delle precedenti

2) 2 ☐

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. Θ (n)
C. $\Theta(n^2)$
D. Nessuna delle risposte precedenti

3) 3 ☐

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

```
        for i=1 to logn
                for j=1 to logn
                        x=i*j
        return x
```

A. O(log n)
B. O(n log n), ma non Θ (n log n)
C. $\Theta (n^2)$
D. Nessuna delle risposte precedenti

Dal file in Materiale del corso

1. Dimostrare che
   a) $3n + 5 = O(n)$
   b) $n = O(3n + 5)$
2. Dimostrare che
   a) $3n - 5 = O(n)$
   b) $n = O(3n - 5)$
3. Sapreste dimostrare che, comunque scelgo due costanti a e b positive, valgono le due affermazioni seguenti?
   i) $an + b = O(n)$
   ii) $n = O(an + b)$
4. i) E' vero che $7n = O(n^2)$?
   ii) E' vero che $n^2 = O(7n)$?
   In entrambi i casi e' necessario giustificare la risposta.

Dal file in Materiale del corso

5. Dimostrare che

   i) $n^2 - 3n + 5 = O(n^2)$

   ii) $n^2 = O(n^2 - 3n + 5)$

6. Dimostrare che

   i) $n^2 + 3n + 5 = O(n^2)$

   ii) $n^2 = O(n^2 + 3n + 5)$

7. Si dimostri che

   a) $4\sqrt{n}\log n + 7n = \Theta(n)$

   b) $n^{\log n} = O(n^n + 2^n)$