

# Tipici tempi di esecuzione Esercizi

Venerdì 11 marzo 2022

# 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

## Notazioni Asintotiche: notazione $O$

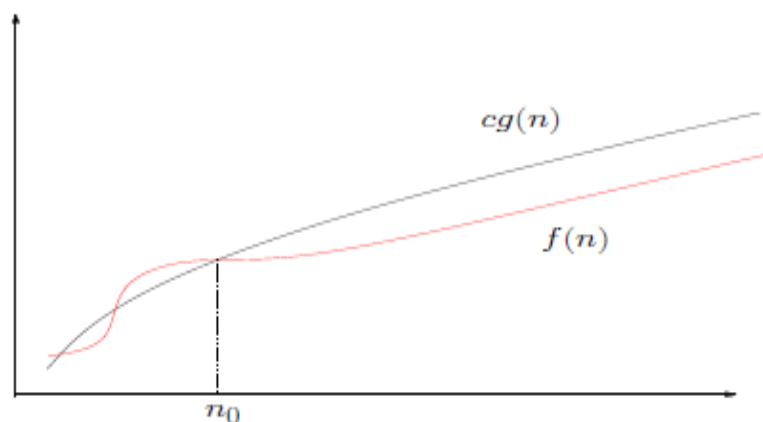
---

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



# 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

# Per stabilire la crescita di una funzione

Basterà usare:

- La «**scaletta**»
- Le proprietà di **additività** e **transitività**
- Le due **regole fondamentali**:
  1. **Possiamo trascurare i termini additivi di ordine inferiore**
  2. **Possiamo trascurare le costanti moltiplicative**

# Per confrontare la crescita di due funzioni

$T_1(n)$  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:

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

$T_1(n) \rightarrow$

$T_2(n) \rightarrow$

$cost$
$\log n$
$\sqrt{n}$
$n$
$n \log n$
$n^2$
$n^3$
$2^n$

Verifichiamolo:

# Confronto funzioni

- $T_1(n) = \Theta(f(n))$
- $T_2(n) = \Theta(g(n))$       allora     $T_1(n) = o(T_2(n))$
- $f(n) = o(g(n))$

*Dimostrazione.* Esistono  $c_1, d_1, c_2, d_2, n_1, n_2$  tali che:

$c_1 f(n) \leq T_1(n) \leq d_1 f(n)$  per ogni  $n \geq n_1$  e

$c_2 g(n) \leq T_2(n) \leq d_2 g(n)$  per ogni  $n \geq n_2$  e

per ogni  $c > 0$ , esiste  $n_c$  tale che  $f(n) \leq c g(n)$  per ogni  $n \geq n_c$ .

Allora: per ogni  $k > 0$ , per ogni  $n \geq \max\{n_1, n_2, n_c\}$ , con  $c = \frac{k c_2}{d_1}$

$$T_1(n) \leq d_1 f(n) \leq d_1 \frac{k c_2}{d_1} g(n) = k c_2 g(n) \leq k T_2(n)$$

A survey on common running times

[KT] par. 2.4



# Linear Time: $O(n)$

**Linear time.** Running time is at most a constant factor times the size of the input.

```
max ← a1
for i = 2 to n {
    if (ai > max)
        max ← ai
}
```

Computing the **maximum**. Compute maximum of  $n$  numbers  $a_1, \dots, a_n$ .

Esistono anche casi più complessi in cui il tempo di esecuzione è  $O(n)$ , ma non si tratta di un singolo for. Per esempio, vedremo procedura **Merge**

# Quadratic Time: $O(n^2)$

**Quadratic time.** Enumerate all pairs of elements.

**Closest pair of points.** Given a list of  $n$  points in the plane  $(x_1, y_1), \dots, (x_n, y_n)$ , find the pair that is closest (here **min** is the square of the minimum distance)

$O(n^2)$  solution. Try all pairs of points.

```
min ←  $(x_1 - x_2)^2 + (y_1 - y_2)^2$ 
for i = 1 to n {
  for j = i+1 to n {
    d ←  $(x_i - x_j)^2 + (y_i - y_j)^2$ 
    if (d < min)
      min ← d
  }
}
```

← don't need to  
take square roots

Remark.  $\Omega(n^2)$  seems inevitable, but this is just an illusion.

# Cubic Time: $O(n^3)$

**Cubic time.** Enumerate all triples of elements.

**Set disjointness.** Given  $n$  sets  $S_1, \dots, S_n$  each of which is a subset of  $1, 2, \dots, n$ , is there some pair of these which are disjoint?

```
foreach set  $S_i$  {  
    foreach other set  $S_j$  {  
        foreach element  $p$  of  $S_i$  {  
            determine whether  $p$  also belongs to  $S_j$   
        }  
        if (no element of  $S_i$  belongs to  $S_j$ )  
            report that  $S_i$  and  $S_j$  are disjoint  
    }  
}
```

$O(n^3)$  solution. For each pairs of sets, determine if they are disjoint.

# Polynomial Time: $O(n^k)$ Time

**Independent set of size  $k$ .** Given a graph, are there  $k$  nodes such that no two are joined by an edge?

$k$  is a constant

$O(n^k)$  solution. Enumerate all subsets of  $k$  nodes.

```
foreach subset S of k nodes {  
    check whether S is an independent set  
    if (S is an independent set)  
        report S is an independent set  
    }  
}
```

Check whether  $S$  is an independent set =  $O(k^2)$ .

$$\text{Number of } k \text{ element subsets} = \binom{n}{k} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{k(k-1)(k-2)\cdots(2)(1)} \leq \frac{n^k}{k!}$$
$$O(k^2 n^k / k!) = O(n^k).$$

# Exponential Time

**Independent set.** Given a graph, what is maximum size of an independent set?

$O(n^2 2^n)$  solution. Enumerate all subsets.

```
S* ←  $\phi$ 
foreach subset S of nodes {
    check whether S is an independent set
    if (S is largest independent set seen so far)
        update S* ← S
    }
}
```

Note the differences with **Independent set of size k**.

# Sub-linear Time: $O(\log n)$

**Tempo lineare**: esamina tutto l'input eseguendo operazioni di tempo costante ad ogni passo

Tempo **sub**-lineare: Non è necessario esaminare **tutto** l'input!

*Esempio.* **Ricerca binaria**: ricerca di un elemento in un array **ordinato** (per esempio un vocabolario)

## Tempo Logaritmico: $O(\log n)$

---

Esempio: Ricerca Binaria. Data una lista ordinata  $A = a_1, \dots, a_n$  ed un valore  $key$ , determina l'indice  $i$  per cui  $a_i = key$ , se esso esiste.

```
first  $\leftarrow$  1, last  $\leftarrow$  n
while (first  $\leq$  last)
    mid  $\leftarrow$  (first + last)/2; (calcola punto mediano)
    if (key >  $a_{mid}$ )
        first = mid + 1; (ripete la ricerca nella metà di destra)
    else if (key <  $a_{mid}$ )
        last = mid - 1; (ripete la ricerca nella metà di sinistra)
    else
        return(mid)
return(non c'è)
```

# Analisi

---

```
first ← 1, last ← n
while (first ≤ last)
    mid ← (first + last)/2; (calcola punto mediano)
    if (key > amid)
        first = mid + 1; (ripete la ricerca nella metà di destra)
    else if (key < amid)
        last = mid - 1; (ripete la ricerca nella metà di sinistra)
    else
        return(mid)
return(non c'è)
```

Dopo la prima iterazione, al più l'algoritmo riefettua la ricerca su  $n/2$  elementi dopo la seconda iterazione, al più l'algoritmo riefettua la ricerca su  $n/4 = n/2^2$  elementi, ... dopo la  $k$ -esima iterazione, al più l'algoritmo riefettua la ricerca su  $n/2^k$  elementi. L'algoritmo si fermerà sicuramente al primo  $k$  per cui  $n/2^k \leq 1$  (se non prima)  
 $\Rightarrow k = O(\log n) \Rightarrow$  poichè il numero di operazioni in ciascuna delle  $O(\log n)$  iterazioni è costante, il tempo di esecuzione totale è  $O(\log n)$

---



# $O(n \log n)$ Time

## Molto comune perché

- E' il running time di **algoritmi divide-and-conquer** che dividono l'input in due parti, le risolvono ricorsivamente e poi combinano le soluzioni in tempo lineare.
- Running time di **algoritmi di ordinamento**.  
Mergesort e Heapsort usano  $O(n \log n)$  confronti.
- Molti algoritmi usano l'ordinamento come passo più costoso. Per esempio molti algoritmi basati sulla **tecnica greedy**

# ESERCIZI

Ho attivato sulla pagina del corso nella piattaforma dipartimentale

<http://elearning.informatica.unisa.it/>

alcuni **Compiti da svolgere** e **Domande a risposta multipla** sugli argomenti finora trattati.

Vi invito caldamente a svolgerli così che possiamo discuterne le soluzioni in aula.

# Tempo di esecuzione 1

svolto

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.  $\Theta(n \log n)$

C.  $\Theta(n^2)$

D. Nessuna delle risposte precedenti

**Soluzione:** D

$$T(n) = c \times n/2 = \Theta(n)$$

# Esercizi analisi tempo di esecuzione

svolto

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

### Analisi1

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

```
for i=1 to n/2
    x=0
    for j=1 to logn
        x=i+j
    return x
```

- A.  $\Theta(n)$
- B.  $\Theta(n \log n)$
- C.  $O(n \log n)$ , ma non  $\Theta(n \log n)$

Nota: sono corrette  
sia la A che la C

## Confronto2

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

- A.  $f(n) = o(g(n))$
- B.  $f(n) = \omega(g(n))$
- C.  $f(n) = \Theta(g(n))$
- D. nessuna

# Prima prova intercorso 2020

svolto

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.  $\Theta(n)$
- 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  $\Theta(n \log n)$
- 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)  $n^{\log n} = O(n^n + 2^n)$

# Dal file in Materiale del corso

8. Si considerino le seguenti funzioni:  $4\sqrt{n} + \log n$ ,  $\log \log n$ ,  $2^n$ ,  $n^{\log n}$ ,  $13n^3$ ,  $n + 15$ .

a) Si ordinino le funzioni scrivendole da sinistra a destra, in modo tale che la funzione  $f(n)$  sia posta a sinistra di  $g(n)$  se  $f(n) = O(g(n))$ .

b) Si dimostri formalmente (cioè fornendo le costanti) almeno due (a scelta) dei confronti affermati al punto a). In altre parole se l'ordine proposto è:  $f_1(n)$ ,  $f_2(n)$ ,  $f_3(n)$ ,  $f_4(n)$ ,  $f_5(n)$ ,  $f_6(n)$ , allora occorre dimostrare che  $f_i(n) = O(f_{i+1}(n))$  per almeno due diversi indici  $i$ .

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

# 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 **molto 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.

# Appello 14 febbraio 2017

## Quesito 2 (23 punti) (*San Valentino*)

Oggi è San Valentino e volete comprare un regalo per il vostro partner. Andate nel suo negozio preferito e selezionate un insieme  $S$  di  $n$  oggetti che sicuramente sarebbero graditi. Ma, guardando nel vostro portafoglio, vi accorgete che con la somma che avete, di certo non potete comprarli tutti! Decidete allora di assegnare ad ogni oggetto il presunto valore di gradimento e di selezionare quindi un insieme di oggetti di  $S$  che abbiano un valore di gradimento massimo, ma che non superi la somma che avete a disposizione. Il problema non è però di facile soluzione, anche perché non avete molto tempo a disposizione, in quanto il negozio sta per chiudere.

Formalizzate il problema reale in un problema computazionale e risolverlo nel modo più efficiente possibile con la tecnica che ritenete più opportuna. E' necessario descrivere l'algoritmo soluzione e valutarne l'efficienza.