

# Tempo di esecuzione e Analisi asintotica

2 marzo 2022

# Argomenti in breve

- Metodologie di **analisi** di algoritmi
- **Tecniche di progettazione** di algoritmi:
  1. Divide et impera
  2. Programmazione dinamica
  3. Tecnica greedy
  4. Algoritmi esaustivi intelligenti
- **Grafi** e principali algoritmi su grafi

# Algoritmi e pseudocodice

Un **algoritmo** è un procedimento per risolvere un **problema** mediante una sequenza finita di operazioni **elementari**.

Il procedimento deve essere descritto in maniera precisa e **univoca** allo scopo di essere eseguito automaticamente dall'**esecutore**.

Un **problema computazionale** è una relazione fra alcuni dati in ingresso (input) e alcuni dati in uscita (output)

L'algoritmo potrà essere implementato in un **programma** mediante un linguaggio di programmazione e una opportuna struttura dei dati.

# Obiettivi del corso

Abbiamo bisogno di algoritmi ogni qual volta vogliamo scrivere un programma.

**Obiettivo finale:** programmare in maniera **consapevole** e creativa

Dal problema reale al programma che lo risolve:

1. **Formalizzazione** del problema
2. Scelta della **tecnica** per progettare algoritmo
3. Scelta della **struttura dati** per organizzare i dati
4. Dimostrazione della **correttezza**
5. **Analisi** risorse usate / efficienza

# Esempio

Ricerca del massimo

**INPUT:** n numeri  $a_1, \dots, a_n$  su cui è definito un ordine  $\leq$

**OUTPUT:** elemento  $a_i$  massimo (cioè t.c. ogni altro elemento  $a_j$  sia  $a_j \leq a_i$ )

(Variante: output indice i)

# Pseudocode

Ricerca del massimo fra  $n$  numeri  $a_1, \dots, a_n$ .

Un algoritmo (in pseudo-codice):

```
max =  $a_1$ 
For i = 2 to n
    If ( $a_i > \text{max}$ ) then
        max =  $a_i$ 
    Endif
Endfor
```

Pseudocodice di [KT]

```
max ←  $a_1$ 
for i = 2 to n {
    if ( $a_i > \text{max}$ )
        max ←  $a_i$ 
}
```

Pseudocodice di [CLRS]

# Quale il tempo di esecuzione?

- Numero di secondi?
- Implementato con quale struttura dati, linguaggio, macchina, architettura, compilatore.....?
- Su quanti numeri? 100, 1.000, 1.000.000?

# Analisi

Studieremo il tempo in **funzione** della taglia **n** dell'input :

$$T(n)$$

Vogliamo analizzare l'efficienza dell'**algoritmo** (non di un programma) prima di scegliere implementazione, hardware etc.

**Cosa posso dire?**



# 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  $n$ .

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

**Per esempio:** assegnamento, incremento, confronto

Calcoleremo il tempo di esecuzione di un algoritmo, **ricorsivamente**, partendo dalla sua struttura.

# Struttura di un algoritmo (non ricorsivo)

Può essere costituito da:

- Una singola **istruzione elementare**
- Un **blocco** sequenziale di istruzioni (allo stesso livello di indentazione)
- Un'istruzione if: 

```
If(cond) then istr1  
                        Else istr2 Endif
```
- Un **ciclo** o loop for, while, repeat:
  - **For** i=a to b {istr} Endfor
  - **While** (cond){istr}
  - **Repeat** {istr} until (cond)

Pseudocodice di [KT]

# Struttura dell'esempio

```
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:

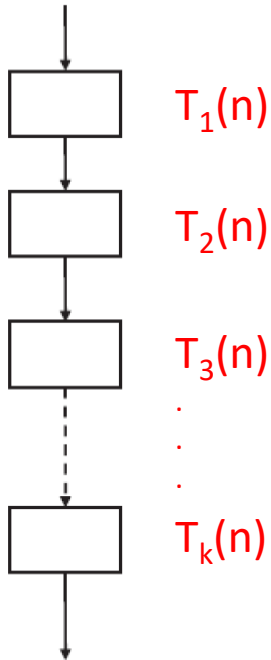
```
If(cond) then istr1
      Else istr2 Endif
```

dove **cond** è il confronto: **a<sub>i</sub> > max**

**istr1** è l'assegnamento: **max = a<sub>i</sub>**

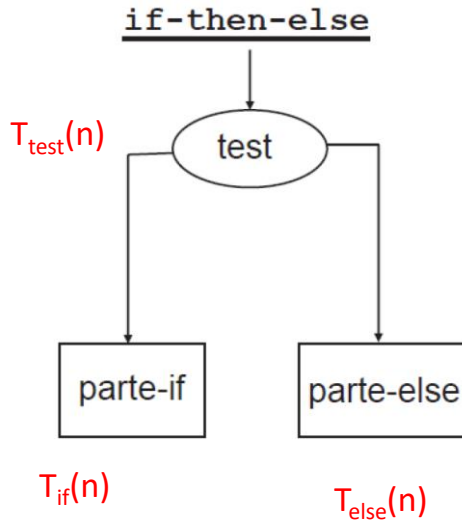
**istr2** è vuota

# Blocco sequenziale



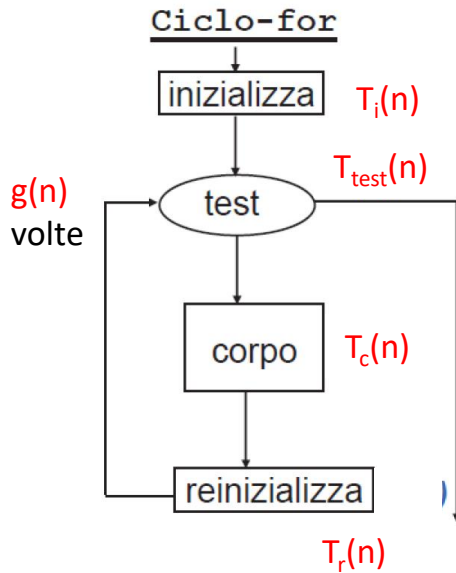
$$T(n) = T_1(n) + T_2(n) + T_3(n) + \dots + T_k(n)$$

# If then else



$$T(n) \leq T_{\text{test}}(n) + \max \{T_{\text{if}}(n), T_{\text{else}}(n)\}$$

# Ciclo for



$$T(n) = T_i(n) + T_{test}(n) + g(n) \times (T_{test}(n) + T_c(n) + T_r(n))$$

# Regole per il calcolo di $T(n)$ (caso non ricorsivo)

- il costo di istruzioni semplici, quali assegnazione, letture/scrittura di variabili é  $O(1)$
- il costo di un blocco sequenziale di istruzioni è pari alla somma dei costi delle singole istruzioni
- il costo di istruzioni tipo **if.... then ....else** é pari al tempo per effettuare il test (tipicamente  $O(1)$ ) piú  $O(\text{costo della alternativa piú costosa})$
- il costo di loop (**for, while, repeat**) é pari alla somma su tutte le iterazioni del costo di ogni iterazioni
- Usando queste regole si può ottenere la complessità di tempo di ogni algoritmo (ad eccezione degli algoritmi ricorsivi, che richiedono una tecnica diversa)

# Calcolo del tempo di esecuzione (1)

*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
```

Esecuzione del for (2-6):

n-1 {	i=2	2<=n?	if
	i=2+1=3	3<=n?	if
	i=3+1=4	4<=n?	if
	...	...	...
	i=n	n<=n?	if
	i=n+1	n+1<=n?	

Tempo di esecuzione = tempo esecuzione linea 1 + tempo esecuzione for(2-6)

Linea 1: 1 assegnamento

Linee 3-5 (una esecuzione): 1 confronto + 1 assegnamento eventualmente

Linee 2-6: 1 assegnamento (i=2) + (n-1) incrementi + n confronti + (n-1) if



# Calcolo del tempo di esecuzione (2)

```
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 =  $c_3$

Linea 1: 1 assegnamento =  $c_1$

Linee 3 – 5 (una esecuzione): 1 confronto + 1 assegnamento eventualmente  $\leq c_2 + c_1$

Linea 2 - 6: 1 assegnamento + (n-1) incremento + n confronti + (n-1) if  $\leq$

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

$$\begin{aligned} T(n) &\leq 2 c_1 + (n-1) c_3 + n c_2 + (n-1) (c_2 + c_1) = \\ &= (c_3 + 2c_2 + c_1) n + c_1 - c_3 - c_2 \end{aligned}$$

# Calcolo del tempo di esecuzione (3)

*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?

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

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

$T(n)$  è lineare!

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

# Analisi asintotica di $T(n)$

Nell'analisi dell'**algoritmo** possiamo studiare  **$T(n)$**

- **Indipendentemente** dal valore delle **costanti** (costo di operazioni elementari)
- Al crescere della taglia dell'input

Questa si chiama **analisi asintotica** dove «**asintotica**» significa

- per  $n$  che tende a infinito
- per  $n$  arbitrariamente grande
- da un certo punto in poi
- per ogni  $n \geq n_0$

# Vantaggi dell'analisi asintotica

- **Indipendente** da hardware
- Effettuabile con pseudocodice **prima** di implementare l'algoritmo
- Considera **infiniti** input

**Alternativa?** Analisi su dati campione.

*Svantaggi:* bisogna avere **già** implementato l'algoritmo;  
analizza numero **finito** di dati

# 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**

Ma vi sono vari modi di **crescita** della funzione  $T(n)$  al crescere di  $n$  (lineare, quadratica, polinomiale, esponenziale, ....): questa informazione può dirci già molto sull'**efficienza** di quella che sarà un'implementazione dell'algoritmo.

# Funzioni standard

Confronteremo la crescita della funzione  $T(n)$  con alcune funzioni standard.

Fra le funzioni non decrescenti  $T: \mathbb{N} \rightarrow \mathbb{R}_+$  ci interesseranno principalmente le seguenti (e le loro combinazioni):

$$T_1(n) = c, \text{ con } c \text{ costante}$$

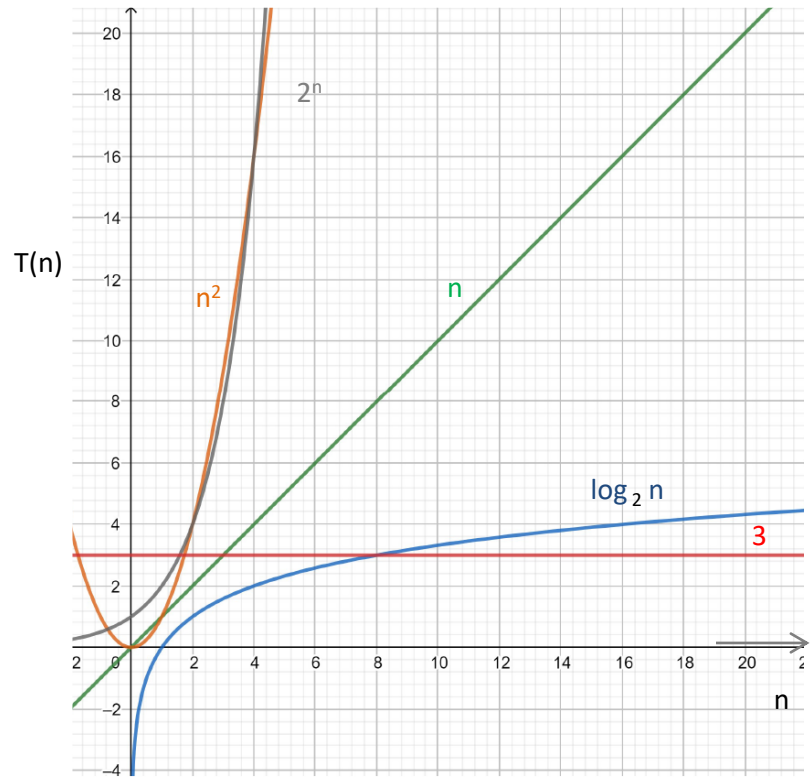
$$T_2(n) = \log n$$

$$T_3(n) = n$$

$$T_4(n) = n^2$$

$$T_5(n) = 2^n$$

# Grafici delle funzioni



# Casi di interesse

*Esempio:* problema dell'ordinamento

**INPUT:**  $n$  numeri  $a_1, \dots, a_n$

**OUTPUT:** permutazione dei numeri in cui ogni numero sia minore del successivo

Esistono svariati algoritmi che lo risolvono

**Qual è il tempo di esecuzione** per ordinare  $n$  elementi con un fissato algoritmo (per esempio InsertionSort)?

Anche per una stessa taglia  $n$  fissata, il tempo può dipendere dalla

**distribuzione** dei numeri fra di loro

(es.: sono già ordinati, sono ordinati in senso inverso, sono tutti uguali, etc.)



# Caso peggiore, migliore, medio

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)

Quando un algoritmo può essere considerato efficiente?

# Worst-Case Polynomial-Time

- **Def.** An algorithm is **efficient** if its running time is polynomial, i.e. **upper bounded** by a polynomial function (e.g.  $T(n) \leq A n^3 + B n + C$ )
- **Justification:** **It really works in practice!**
  - Although  $6.02 \times 10^{23} \times N^{20}$  is technically poly-time, it would be useless in practice.
  - In practice, the poly-time algorithms that people develop almost always have low constants and low exponents.
  - Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.
- **Exceptions.**
  - Some poly-time algorithms do have high constants and/or exponents and are useless in practice.
  - Some exponential-time (or worse) algorithms are widely used because the worst-case instances seem to be rare.

# Efficiency = polynomial

---

Polynomial-time solvability emerged as a formal notion of efficiency by a gradual process, motivated by the work of a number of researchers including Cobham, Rabin, Edmonds, Hartmanis, and Stearns.

Similarly, the use of asymptotic order of growth notation to bound the running time of algorithms—as opposed to working out exact formulas with leading coefficients and lower-order terms—is a modeling decision that was quite non-obvious at the time it was introduced;

# Confronto efficienza

$n \log n$  vs  $2^n$

	$n$	$n \log_2 n$	$n^2$	$n^3$	$1.5^n$	$2^n$	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	$10^{25}$ years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	$10^{17}$ years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	2 days	31,710 years	very long	very long	very long

accettabile

non accettabile

# Crescita funzioni

funzione	n	n+1	n+2	2×n
	$2^n$	$2^{n+1} = 2^n \times 2$	$2^{n+2} = 2^n \times 4$	$2^{2 \times n} = (2^n)^2$
	$n^2$	$(n+1)^2 = n^2 + 2n + 1$	$(n+2)^2 = n^2 + 4n + 4$	$(2n)^2 = 4 \times n^2$
	n	n+1	n+2	2×n
	$\log_2 n$	$\log_2 (n+1) = \log_2 n + \epsilon$	$\log_2 (n+2) = \log_2 n + \delta$	$\log_2 (2 \times n) = \log_2 n + 1$

funzione	n=8	n+1	n+2	2×n
$2^n$	$2^8=256$	$2^{8+1} = 512$	$2^{8+2} = 1024$	$2^{2 \times 8} = (2^8)^2 = 65.536$
$n^2$	$8^2 = 64$	$(8+1)^2 = 81$	$(8+2)^2 = 100$	$(2 \times 8)^2 = 256$
n	8	$8 + 1 = 9$	$8 + 2 = 10$	$2 \times 8 = 16$
$\log_2 n$	$\log_2 8 = 3$	$\log_2 (8+1) = 3, 169$	$\log_2 (8+2) = 3,321$	$\log_2 (2 \times 8) = 4$

# 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)

Le notazioni asintotiche:  $O$ ,  $\Omega$ ,  $\Theta$

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

Diremo per esempio che l'algoritmo per la ricerca del massimo ha un tempo di esecuzione **lineare**,  $T(n) = O(n)$ , essendo  $T(n) \leq An + B$

# 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

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

Definizione per «informatici» la prossima volta!



# Obiettivi formativi...

- Estrapolare il problema computazionale
- Riconoscere (analogie e variazioni con) problemi noti
- Scegliere tecnica/he più appropriata/e
- Applicarla/e correttamente
- Valutare l'efficienza

## ESERCIZI

Nei compiti che seguono, potete provare a **formalizzare il problema computazionale** alla base del problema «reale» descritto.

# Problemi...

Un algoritmo risolve un problema, ma...

... cos'è un problema?

Corrispondenza fra spazio delle **istanze** (dati in ingresso) e delle **soluzioni**



# Appello 25 gennaio 2021

SVOLTO A LEZIONE

## Quesito 3 (20 punti)

I signori Braccino hanno comprato una **bicicletta elettrica** per i loro 6 figli. Siccome è il loro **unico** mezzo di locomozione, ogni mattina i ragazzi si ritrovano, ognuno esprime la propria richiesta per la bicicletta e cercano la soluzione che accontenti il maggior numero di loro. Oggi Ada ne avrebbe bisogno dalle 10 alle 14, Bea dalle 11 alle 12, Camillo dalle 15 alle 18, Dante dalle 12 alle 16, Elena dalle 13 alle 15 e Fabio dalle 15 alle 17.

**Quanti** ragazzi al massimo potranno usufruire della bicicletta? **Giustificate** la risposta, indicando quale algoritmo, fra quelli studiati, avete utilizzato.

# 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.

# 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. Avete trovato una soluzione, ma è molto lenta. Vi ricordate allora che al corso di Algoritmi vi avevano tempestato con le varie tecniche di progettazione di algoritmi: potranno esservi utili (almeno una volta nella vita)?

Formalizzate il problema reale in un problema computazionale e risolverlo in maniera che sia il più efficiente possibile. Giustificate le risposte.

# 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.