

# Algoritmi e Strutture Dati

## Modulo 2

Jocelyne Elias

<https://www.unibo.it/sitoweb/jocelyne.elias/>

Moreno Marzolla

<https://www.moreno.marzolla.name/>

Dipartimento di Informatica—Scienza e Ingegneria (DISI)  
Università di Bologna



This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

# Presentiamoci

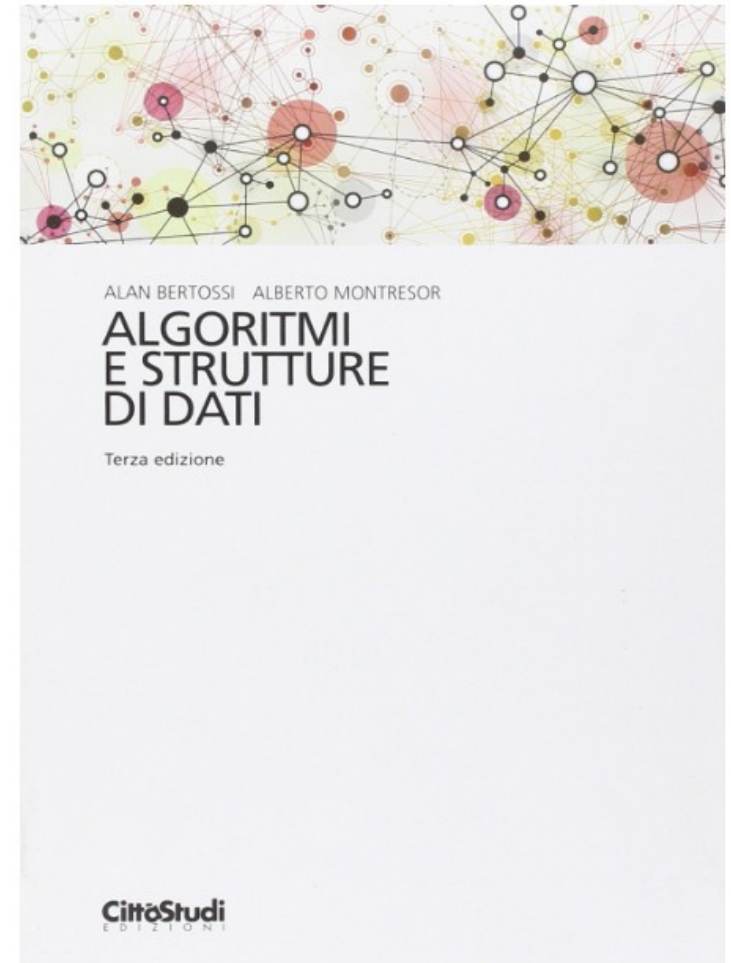
- **Modulo 2 (II ciclo)**
  - Jocelyne Elias
  - [jocelyne.elias@unibo.it](mailto:jocelyne.elias@unibo.it)
  - ***Tutor: Michele Dinelli e Martina Ianaro***
- **Orario delle lezioni**
  - Lunedì 09:00 – 12:00, Aula E2
  - Mercoledì 09:00 – 12:00, Aula E2
  - <https://www.unibo.it/it/studiare/dottorati-master-specializzazioni-e-altra-formazione/insegnamenti/insegnamento/2024/509379/orariolezioni>
- **Ricevimento**
  - Da concordare via mail

# Sito web del Modulo 2

- Piattaforma Virtuale
  - Avvisi
  - Lucidi delle lezioni
  - Dispensa di esercizi svolti

# Bibliografia

- Testo adottato
  - Alan Bertossi, Alberto Montresor, *Algoritmi e strutture di dati Terza Edizione*, 2014, Città Studi, ISBN: 9788825173956
- Testi di consultazione
  - Camil Demetrescu, Irene Finocchi, Giuseppe F. Italiano, *Algoritmi e strutture dati 2/ed*, 2008, McGraw-Hill, ISBN: 9788838664687
  - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduzione agli algoritmi e strutture dati 3/ed*, 2010, McGraw-Hill, ISBN: 9788838665158



# Programma

- Grafi/algoritmi di visita di grafi
- Alberi di copertura (*spanning trees*)
- Cammini minimi
- *Analisi ammortizzata degli algoritmi*
- Tecniche algoritmiche
  - Divide-et-impera
  - Greedy
  - Programmazione dinamica
- Macchine di Turing e teoria della calcolabilità (se avanzerà tempo)
- Classi di complessità dei problemi (se avanzerà tempo)

# Prerequisiti

- Programmazione Internet + Lab. di prog. Internet
  - Algoritmi e Strutture Dati  $\neq$  Programmazione
  - In questo corso non si impara a programmare, perché dovrete già essere in grado di farlo
- Nozioni di base di algebra e analisi matematica
  - Sommatorie, polinomi, ordini di grandezza delle funzioni, disequazioni

# Scopo del corso

- **Contenuto**

- Una panoramica di problemi noti e loro soluzioni
- Elenco di algoritmi e strutture dati standard
- Come valutare l'efficienza degli algoritmi

- **Metodo**

- Principi e tecniche per risolvere problemi algoritmici
- Come risolvere nuovi problemi, applicando soluzioni note o “inventando” varianti alle soluzioni note



# Modalità d'esame

- **Progetto** da svolgere **individualmente**
  - **4-5** esercizi/algoritmi da progettare e realizzare in Java
  - **3 set** di progetti diversi:
    - uno per la sessione estiva (giugno-luglio)
    - uno per la sessione autunnale (settembre)
    - uno per la sessione invernale (gennaio/febbraio 2026)
  - Specifiche disponibili **circa un mese prima** della consegna
  - Consegna tramite la piattaforma "Virtuale"

# Modalità d'esame cont.

- Un progetto sufficiente consente l'accesso all'**orale** per la sessione cui si riferisce il progetto
- La prova orale include
  - Discussione del progetto
  - Domande su tutti gli argomenti svolti a lezione e durante le esercitazioni

# Regole d'esame

- L'esame è un momento ufficiale e va affrontato con serietà
- Si sono verificati in passato casi di **gravi irregolarità**
  - Tali situazioni sono state (e saranno) sanzionate con la massima intransigenza
  - Esame annullato, da ripetere con nuovo progetto.
- L'esame orale è ugualmente un momento ufficiale
  - È sempre possibile rifiutare il voto e ritentare l'esame (**nuovo** progetto + **nuovo** orale)

# Il vero significato della complessità degli algoritmi

# Sottovettore di valore massimo

- Consideriamo un vettore  $V[1..n]$  di  $n \geq 1$  valori reali arbitrari
- Vogliamo individuare un sotto-vettore  $V[i..j]$  non vuoto di  $V$  la somma dei cui elementi sia massima

3	-5	10	2	-3	1	4	-8	7	-6	-1
---	----	----	---	----	---	---	----	---	----	----

- Soluzione "di forza bruta":  $O(n^3)$
- Esiste una soluzione  $O(n)$  ?
  - stay tuned...

# Soluzione di “forza bruta” $O(n^3)$

```
real SommaMax1( real V[1..n] )
  real smax ← V[1];
  for integer i ← 1 to n do
    for integer j ← i to n do
      real s ← 0;
      for integer k ← i to j do
        s ← s + V[k];
      endfor
      if (s > smax) then
        smax ← s;
      endif
    endfor
  endfor
  return smax;
```

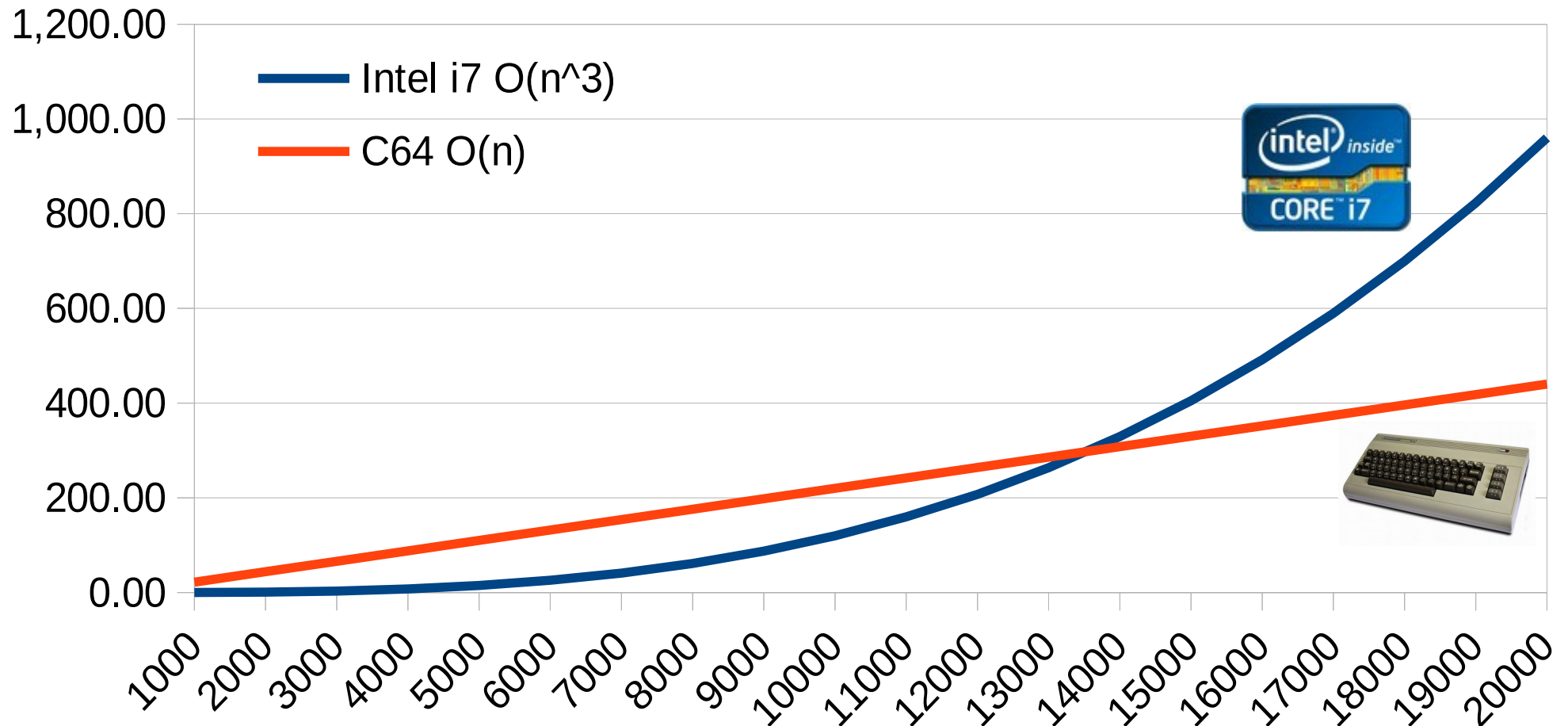
# L'efficienza conta!

- Confrontiamo la soluzione  $O(n^3)$  con una soluzione  $O(n)$  (che vedremo più avanti) su due sistemi molto diversi
- Algoritmo  $O(n^3)$ 
  - Ubuntu Linux 18.04
  - CPU: Intel i7 @ 3.6GHz
  - 16 GB RAM
  - Java (OpenJDK 11.0.10)
- Algoritmo  $O(n)$ 
  - Commodore 64 (anno 1982)
  - CPU: MOS 6502 @ 1MHz
  - 64 KB RAM
  - Commodore BASIC



# Sottovettore di somma massima

Tempi di esecuzione in secondi





# Esercizi di ripasso

# La notazione asintotica $O(f(n))$

## Definizione

Data una funzione costo  $f(n)$ , definiamo l'insieme  $O(f(n))$  come l'insieme delle funzioni  $g(n)$  per le quali esistono costanti  $c > 0$  e  $n_0 \geq 0$  per cui vale:

$$\forall n \geq n_0 : g(n) \leq cf(n)$$

In maniera più sintetica:

$$O(f(n)) = \{g(n) : \exists c > 0, n_0 \geq 0 \text{ tali che } \forall n \geq n_0 : g(n) \leq cf(n)\}$$

Nota: si utilizza la notazione (sebbene non formalmente corretta)  $g(n) = O(f(n))$  per indicare  $g(n) \in O(f(n))$ .

- **Notazione asintotica** - Capitolo: Tecniche di analisi degli algoritmi, Prof. Davide Rossi

# Vero o falso?

1.  $1325 n^2 + 12n + 1 = O(n^3)$

2.  $76 n^3 = O(n^3)$

3.  $n^2 \log n = O(n^2)$

4.  $3^n = O(2^n)$

5.  $2^n = O(2^{n/2})$

6.  $2^{n+100} = O(2^n)$

7.  $\log n = O(n)$

8.  $n = O(n \log n)$

9.  $n^2 = O(n \log n)$

10.  $\log(n^2) = O(\log n)$

11.  $n(n+1) / 2 = O(n)$

# Esercizio

- Determinare il costo asintotico dell'algoritmo seguente

```
algA( integer n ) → integer
  if ( n ≤ 1 ) then
    return 2*n;
  else
    integer a ← 2;
    for integer i ← 1 to n/2 do
      a ← a + 2 * i;
    endfor
    return algA( n/2 ) + algA( n/2 ) + a;
  endif
```

# Esercizio

- Determinare il costo asintotico dell'algoritmo seguente

```
algA( integer n ) → integer
  if ( n ≤ 1 ) then
    return 2*n;
  else
    integer a ← 2;
    for integer i ← 1 to n/2 do
      a ← a + 2 * i;
    endfor
    return algA( n/2 ) + algA( n/2 ) + a;
  endif
```

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

# Esercizio

- Determinare il costo asintotico dell'algorithm seguente

```
algB( integer n ) → integer
  integer a ← 0;
  Integer s, t;
  for s ← 1 to n do
    for t ← s to n do
      a ← a + s + t;
    endfor
  endfor
  return a;
```

# Esercizio

- Determinare il costo asintotico dell'algoritmo seguente

```
algB( integer n ) → integer
  integer a ← 0;
  Integer s, t;
  for s ← 1 to n do
    for t ← s to n do
      a ← a + s + t;
    endfor
  endfor
  return a;
```

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

# Strutture Dati

- Che differenza c'è tra `LinkedList` e `ArrayList` Java?

	LinkedList	ArrayList
Inserimento in testa		
Inserimento in coda		
Inserimento dopo un elemento di posizione/riferimento dati		
Cancellazione di un elemento di posizione/riferimento dati		
Accesso diretto al $k$ -esimo elemento		



# Strutture Dati

- Che differenza c'è tra `LinkedList` e `ArrayList` Java?

	<code>LinkedList</code>	<code>ArrayList</code>
Inserimento in testa	$O(1)$	$O(n)$
Inserimento in coda	$O(1)$	$O(1)$ amm.
Inserimento dopo un elemento di posizione/riferimento dati	$O(1)$	$O(n)$
Cancellazione di un elemento di posizione/riferimento dati	$O(1)$	$O(n)$
Accesso diretto al $k$ -esimo elemento	$O(k)$	$O(1)$

Dalla documentazione di `java.util.ArrayList`

The add operation runs in **amortized constant time ( $O(1)$  amm.)**, that is, adding  $n$  elements requires  $O(n)$  time. All of the other operations run in linear time (roughly speaking).

# Caso ottimo / caso pessimo

- Consideriamo un albero binario di ricerca **non bilanciato** con  **$n$**  nodi
- Quale è il costo asintotico dell'operazione di ricerca
  - nel caso pessimo?
  - nel caso ottimo?

# Caso ottimo / caso pessimo

- Consideriamo un albero binario di ricerca **non bilanciato** con ***n*** nodi
- Quale è il costo asintotico dell'operazione di ricerca
  - nel caso pessimo?  $O(n)$
  - nel caso ottimo?  $O(1)$