

Algoritmi e Strutture Dati

Introduzione al corso

Luciano Bononi
Dip. di Scienze dell'Informazione
Università di Bologna

bononi@cs.unibo.it
<http://www.cs.unibo.it/~bononi/>

Copyright © 2011, Luciano Bononi and Moreno Marzolla,
Università di Bologna, Italy



This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Presentiamoci

- Luciano Bononi
`bononi@cs.unibo.it`
`http://www.cs.unibo.it/~bononi/`
- Lezioni
 - Martedì ore 13:30—15:30, aula Ercolani 1
 - Venerdì ore 11:30—13:30, aula Ercolani 1
- Orario di ricevimento
 - Giovedì ore 14.30-16.30 (da concordare sempre via email!)
 - Mura Anteo Zamboni 7, Ufficio T08

Informazioni generali sul corso

Sito web del corso

- <http://www.cs.unibo.it/~bononi/>
 - > Courses > Algoritmi e Strutture Dati (11929)
- Trovate:
 - **Avvisi!**
 - Lucidi delle lezioni
 - Esercizi
 - Link a ulteriori approfondimenti
 - Più avanti... Testi degli esami precedenti
- Controllate anche il feed degli avvisi sul sito UniBO:
 - <http://www.unibo.it/SitoWebDocente/default.htm?upn=luciano.bononi%40unibo.it>
 - <http://www.unibo.it/SitoWebDocente/default.htm?upn=luciano.bononi%40unibo.it&TabControl1=TabAvvisi>

Testo adottato

- Camil Demetrescu, Irene Finocchi, Giuseppe F. Italiano, *Algoritmi e strutture dati* 2/ed, McGraw-Hill ISBN: 9788838664687, Giugno 2008



Testi consigliati



- Alan Bertossi, Alberto Montresor, *Algoritmi e strutture di dati 2/ed*, Città Studi, ISBN: 9788825173567
 - I contenuti sono pressoché equivalenti a quelli del testo adottato



- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduzione agli algoritmi e strutture dati 2/ed*, McGraw-Hill, ISBN: 9788838662515, Maggio 2005
 - Per chi vuole approfondire

Propedeuticità

- Per sostenere l'esame è necessario aver superato l'esame di **Programmazione Internet+Lab.**
Programmazione Internet

Programma del corso

- Complessità asintotica degli algoritmi
- Strutture dati elementari (Liste, Pile, Code, Alberi...)
- Algoritmi di ordinamento e ricerca
- Alberi di ricerca
- Tabelle Hash
- Tecniche Algoritmiche (divide et impera, algoritmi *greedy*, programmazione dinamica)
- Algoritmi su grafi (spanning tree, cammini minimi, problemi di flusso)

Modalità d'esame

- Esame scritto **obbligatorio**
 - L'esame può essere superato sostenendo due prove parziali
 - Sarà possibile recuperare uno dei due parziali
- Mini-progetti **facoltativi**
 - Da svolgere **singolarmente**, in linguaggio Java
 - Ne verranno assegnati 2 o 3 durante il corso
 - Un progetto consegnato e valutato positivamente vale 1 punto; un progetto non consegnato o non sufficiente vale 0 punti.
 - I punti guadagnati con i progetti verranno **sommati** al voto dello scritto

Un suggerimento

- Nei lucidi troverete una serie di **domande** che vengono lasciate a voi per esercizio.
 - In alcuni casi la risposta si trova nel libro di testo; in generale dovrete essere in grado di rispondere da soli.
 - Provate a rispondere: **potreste ritrovarle come domande d'esame**.
- TUTTO il materiale viene inserito nella pagina del corso:

<http://www.cs.unibo.it/~bononi/>

Un (altro) suggerimento

- **La pratica è fondamentale**: lo studio degli algoritmi non sono uno sport da seguire come “spettatori”
 - Scrivete molti algoritmi
 - Dimostrate la correttezza
 - Analizzare la loro efficienza
- Descriveremo gli algoritmi mediante pseudocodice
 - Per poterne dare una versione compatta senza perdersi in dettagli implementativi
- In alcuni casi verranno mostrati esempio di codice Java

FAQ

- Questo è un esame facile?
 - No.
- È sufficiente studiare sui lucidi?
 - I lucidi sono fatti per integrare lo studio individuale, e soprattutto lo studio sul libro di testo.
- Posso rifare lo scritto?
 - Sì. La consegna dello scritto annulla automaticamente l'eventuale voto precedente.
- Posso rifare uno o più progetti?
 - No. Ogni mini-progetto può essere consegnato **una sola volta**. Se il progetto è insufficiente (o non viene consegnato entro la scadenza) non può essere riconsegnato

FAQ

- **Si puo' fare l'orale?**
 - No (dipende).
 - In casi particolari il docente si riserva la facoltà di richiedere chiarimenti orali sui progetti e/o sulle prove scritte, ad esempio in casi di sospetta “copiatura”.
- **Queste slide non ci sono sul sito del corso /
Queste slide sono diverse sul sito del corso**
 - Cerco di mettere a disposizione le slide aggiornate prima della lezione, ma non sempre ci riesco
 - In ogni caso metto sempre i lucidi online dopo la lezione
 - Talvolta trovo (o mi vengono segnalati) errori in lucidi di lezioni passate. Controllate spesso la pagina del corso per aggiornamenti

Algoritmi e strutture dati

Cos'è un algoritmo?

- Un algoritmo è un procedimento per risolvere un problema mediante una sequenza finita di passi elementari
- Il procedimento deve essere descritto in modo preciso allo scopo di poterne automatizzare l'esecuzione
- Il termine deriva dal nome del matematico persiano **Abu Ja'far Muhammad ibn Musa Khwarizmi**
 - Autore di un primo fondamentale trattato di algebra
 - Un cratere lunare porta il suo nome



Algoritmo vs Programma

- Un **algoritmo** descrive (ad alto livello) una procedura di calcolo che, se seguita, consente di ottenere un certo risultato
- Un **programma** è l'implementazione di un algoritmo mediante un opportuno linguaggio di programmazione.
 - Un programma può essere direttamente eseguito da un calcolatore (processo in esecuzione); un algoritmo solitamente no.

Gli algoritmi sono ovunque!

- **Internet**. Web search, packet routing, distributed file sharing.
- **Biology**. Human genome project, protein folding.
- **Computers**. Circuit layout, file system, compilers.
- **Computer graphics**. Movies, video games, virtual reality.
- **Security**. Cell phones, e-commerce, voting machines.
- **Multimedia**. CD player, DVD, MP3, JPG, DivX, HDTV.
- **Transportation**. Airline crew scheduling, map routing.
- **Physics**. N-body simulation, particle collision simulation.
- ...

Perché studiare gli algoritmi?

[Web](#) [Images](#) [Videos](#) [Maps](#) [News](#) [Shopping](#) [Gmail](#) [more](#) ▼

[Sign in](#) | [Help](#)

Google maps

Search Maps

[Show search options](#)

Find businesses, addresses and places of interest. [Learn more](#)

Get Directions [My Maps](#)

A milano, italy

B napoli, italy

[Add Destination - Show options](#)

By car

Get Directions

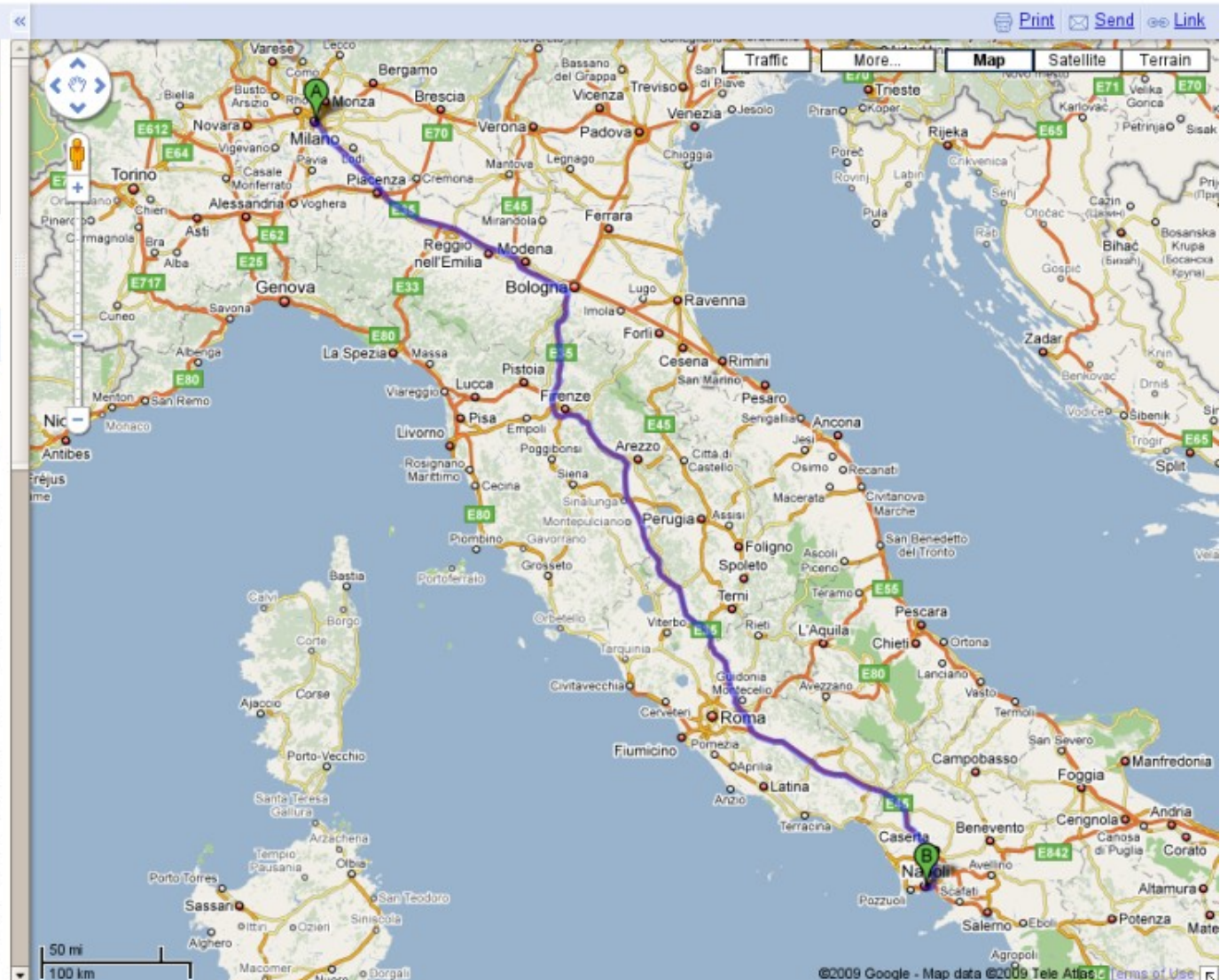
Driving directions to Naples, Italy

Suggested routes

| | |
|------------|------------------------|
| A1 | 7 hours 9 mins |
| 774 km | |
| A14 | 9 hours 36 mins |
| 936 km | |

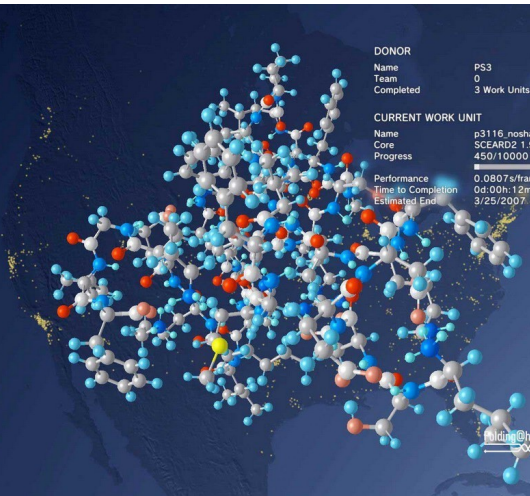
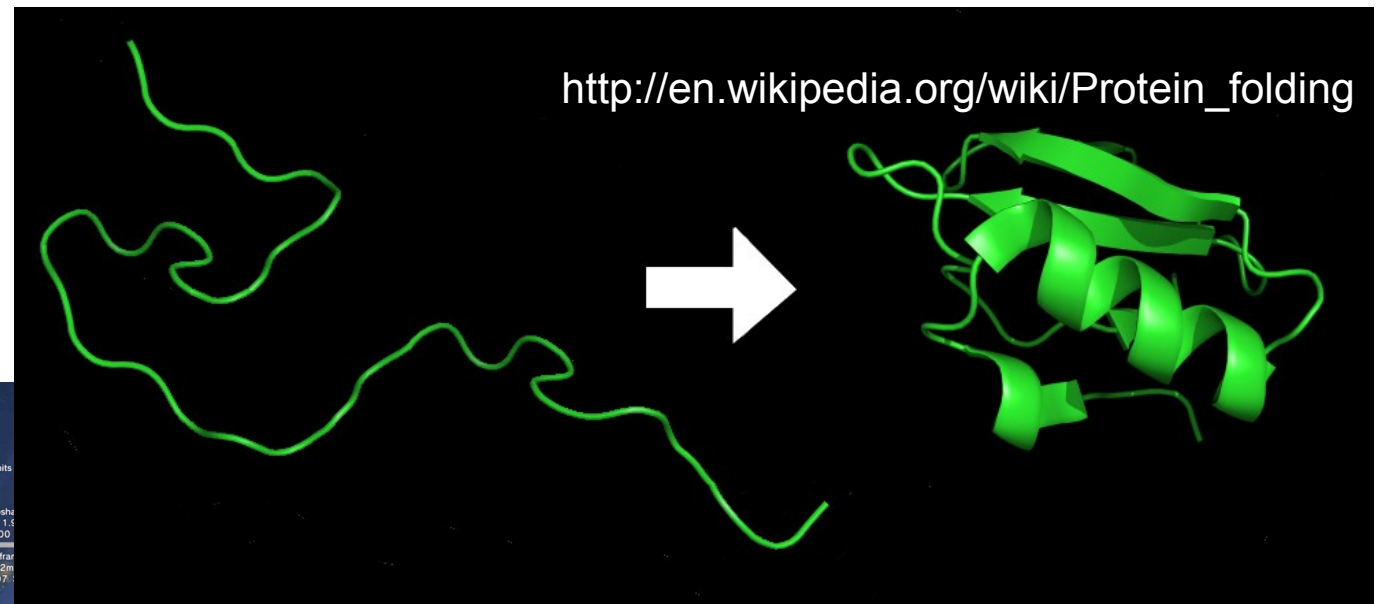
Milan Italy

1. Head **southwest** on **Via Torino** toward **Via Spadari** 0.6 km
2. Continue on **Carrobbio** 66 m
3. Continue on **Via Cesare Correnti** 0.3 km
4. Turn **left** at **Via dei Fabbri** 0.1 km
5. Take the **1st right** onto **Via Edmondo De Amicis** 26 m
6. Take the **1st left** to stay on **Via Edmondo De Amicis** 0.1 km
7. Continue on **Via Molino delle Armi** 0.5 km
8. Continue on **Via Santa Sofia** 0.5 km
9. Turn **right** at **Corso di Porta Romana** 0.3 km
10. Continue straight onto **Largo della Crocetta** 10 m
11. Slight **right** at **Corso di Porta Romana** 0.6 km
12. Continue on **Piazza Medaglie d'Oro** 93 m
Leaving toll zone



Perché studiare gli algoritmi?

- Le proteine assumono una ben precisa struttura tridimensionale a causa dell'interazione degli aminoacidi che le compongono
- Si ritiene che certe malattie neurodegenerative siano causate dall'accumulo di proteine che si ripiegano in maniera “scorretta”
- Folding@Home



Perché studiare gli algoritmi?

- Eliminazione superfici nascoste, strutture dati per codificare l'ambiente di gioco, simulazioni fisiche (collisioni, movimento dei tessuti, sistemi di particelle—fuoco, nebbia, acqua...)



Perché studiare gli algoritmi?

- Per divertimento
- Per profitto
 - Un algoritmo efficiente può fare la differenza tra il poter risolvere un problema e non poterlo risolvere
- Perché alcuni degli algoritmi che studieremo furono inventati da studenti

Cosa imparerete in questo corso?

- Quali sono gli algoritmi “classici” per risolvere problemi ricorrenti
 - Ordinamento, ricerca, problemi su grafi...
- Come valutare l'efficienza di un algoritmo
- Come sviluppare nuovi algoritmi per risolvere problemi che si presentano

Esempio “di riscaldamento”: i numeri di Fibonacci

- La successione di Fibonacci
 $F_1, F_2, \dots, F_n, \dots$ è definita come:

$$F_1 = 1$$

$$F_2 = 1$$

$$F_n = F_{n-1} + F_{n-2}, n > 2$$



Leonardo Fibonacci
(Pisa, 1170—Pisa, 1250)

http://it.wikipedia.org/wiki/Leonardo_Fibonacci

Formula chiusa

- **Buona notizia:** esiste una semplice formula chiusa per il valore di F_n

$$F_n = \frac{1}{\sqrt{5}} (\phi^n - \hat{\phi}^n)$$

ove

$$\phi = \frac{1 + \sqrt{5}}{2} \approx 1.618 \quad \hat{\phi} = \frac{1 - \sqrt{5}}{2} \approx -0.618$$

- **Cattiva notizia:** la valutazione di tale formula introduce errori numerici dovuti alla necessità di fare calcoli in virgola mobile

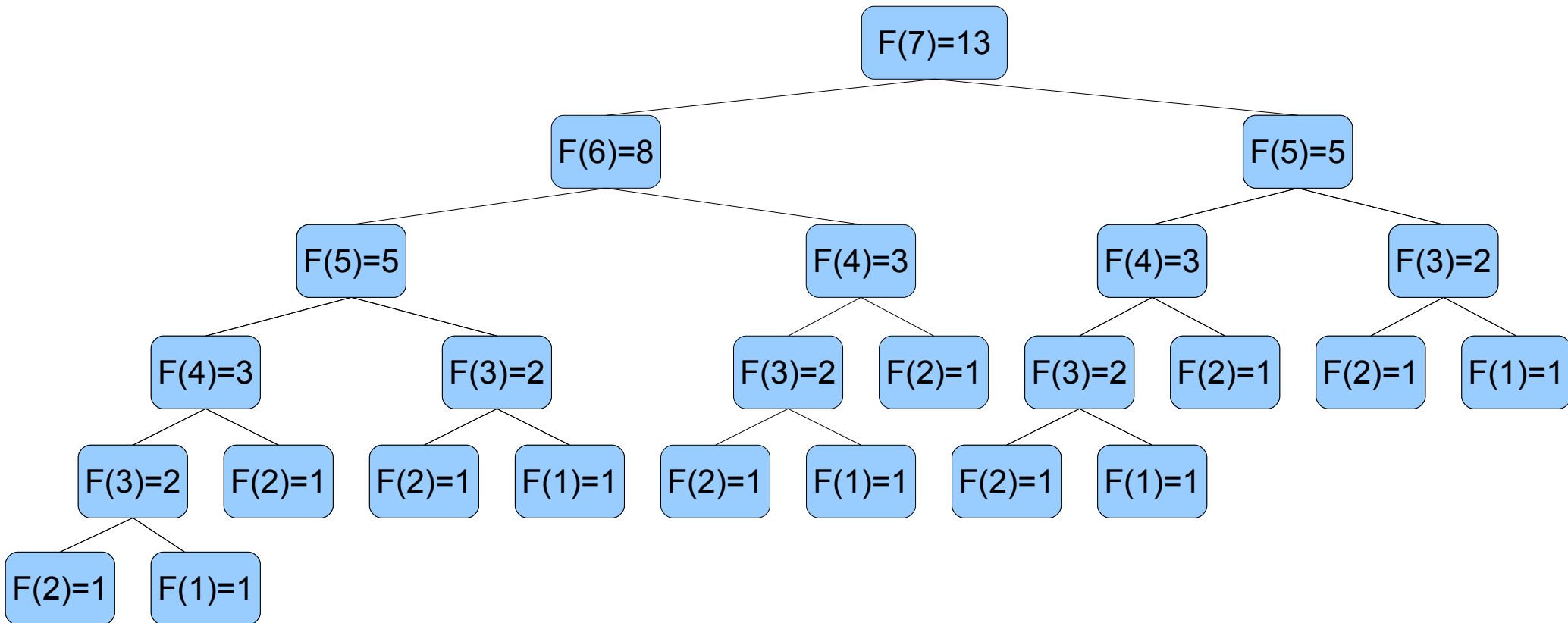
Algoritmo banale

- Definiamo un algoritmo per il calcolo dell'n-esimo numero di Fibonacci utilizzando la definizione ricorsiva

```
algoritmo Fibonacci2(int n) → int
  if ( n==1 || n==2 ) then
    return 1;
  else
    return Fibonacci2(n-1)+Fibonacci2(n-2);
  endif
```

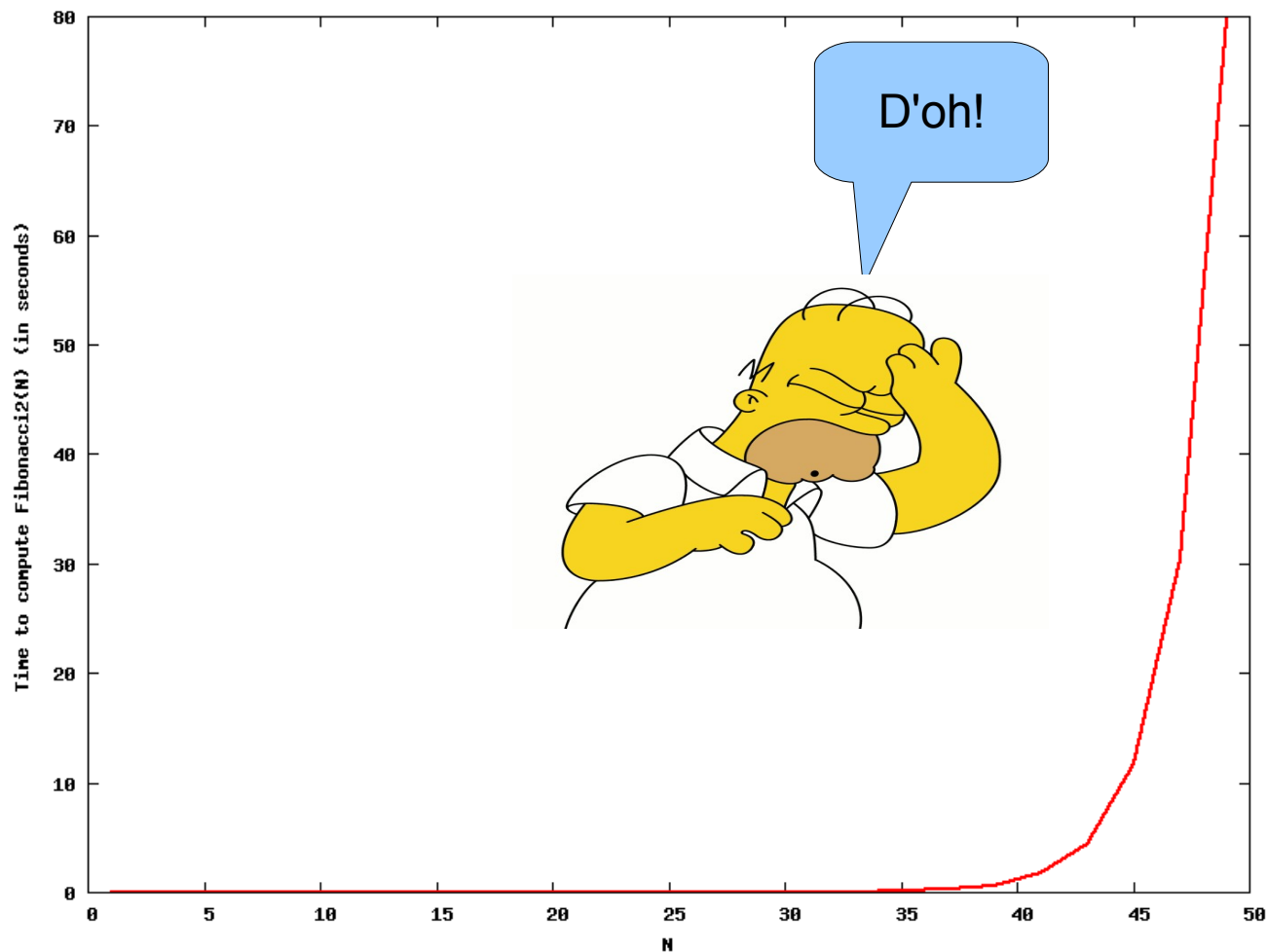
- L'algoritmo è descritto mediante pseudocodice; la sua traduzione in Java (o qualsiasi altro linguaggio) è immediata

Albero di ricorsione



C'è un “piccolo” problema...

- Il tempo necessario per calcolare F_n cresce in maniera “preoccupante” all'aumentare di n

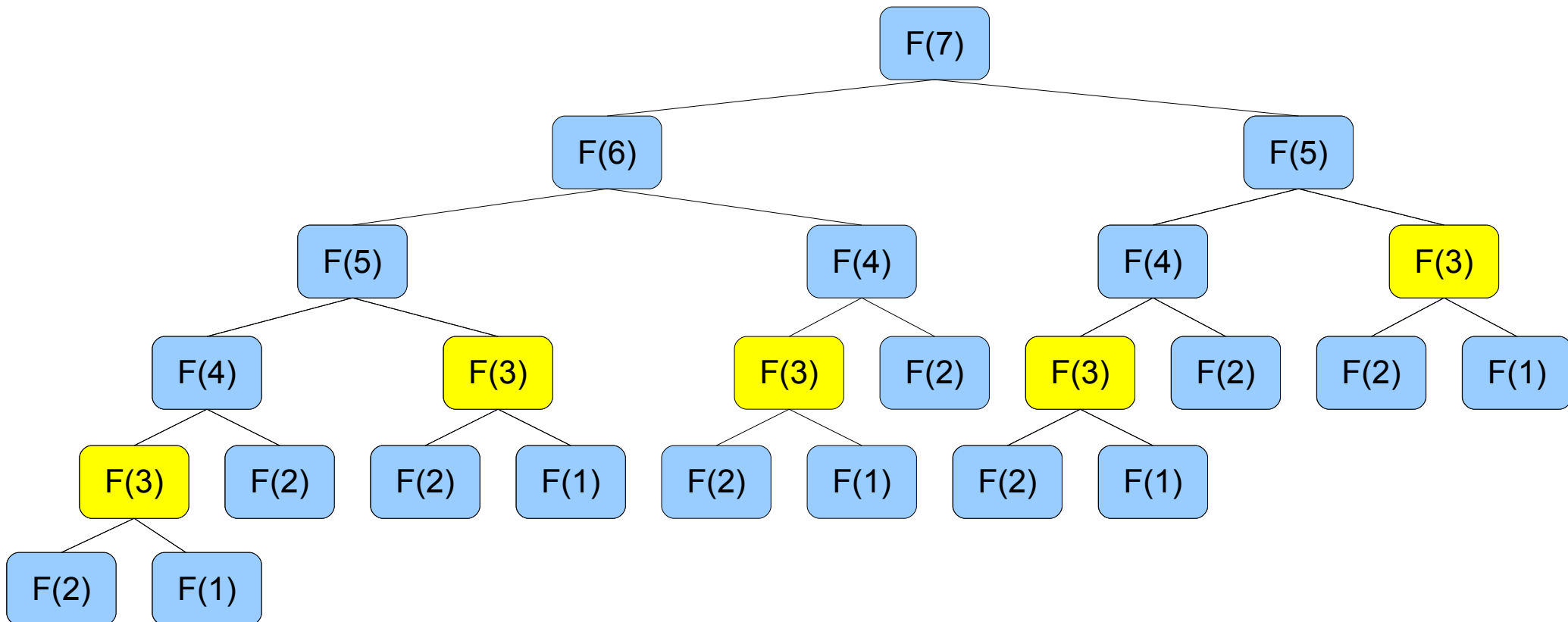


Stima del tempo di esecuzione

- Misurare il tempo di esecuzione in secondi?
 - Dipende dalla macchina su cui si fanno le misure
- Misurare il numero di istruzioni in linguaggio macchina eseguite?
 - Anche questo dipende dalla macchina su cui si fanno le misure, e in più è una quantità difficile da desumere a partire dallo pseudocodice
- **Stimiamo il tempo di esecuzione calcolando il numero di operazioni elementari eseguite nello pseudocodice**

Identifichiamo il problema

- I valori intermedi vengono ricalcolati più volte



Stima del tempo di esecuzione

- Sia $T(n)$ il tempo necessario a calcolare l' n -esimo numero di Fibonacci.
- Possiamo stimare $T(n)$ come il numero di nodi dell'albero di ricorsione per F_n
 - **Domanda**: dare una espressione ricorsiva del numero $T(n)$ di nodi dell'albero di ricorsione per il calcolo di F_n

Stima del tempo di esecuzione

- È possibile dimostrare (per induzione) che

$$T(n) = 2F_n - 1$$

- **Domanda:** dimostrare la relazione sopra
- Ricordando la formula chiusa per F_n possiamo subito concludere che $T(n)$ cresce in modo esponenziale
- Possiamo anche calcolare direttamente un limite inferiore a $T(n)$
 - Vedi pagina successiva

Stima del tempo di esecuzione

```
algoritmo Fibonacci2(int n) → int
  if ( n==1 || n==2 ) then
    return 1;
  else
    return Fibonacci2(n-1)+Fibonacci2(n-2);
  endif
```

- Sia $T(n)$ il numero di nodi dell'albero di ricorsione per il calcolo di F_n
 - $T(1) = T(2) = 1$;
 - $T(n) = T(n-1) + T(n-2) + 1$ (se $n > 2$)
 - È simile alla relazione di ricorrenza che definisce F_n

Limite inferiore al tempo di esecuzione

$$T(n) = T(n-1) + T(n-2) + 1$$

$$\geq 2T(n-2) + 1$$

$$\geq 4T(n-4) + 2 + 1$$

$$\geq 8T(n-6) + 2^2 + 2 + 1$$

$$\geq \dots$$

$$\geq 2^k T(n-2k) + \sum_{i=0}^{k-1} 2^i$$

$$\geq \dots$$

$$\geq 2^{\lfloor n/2 \rfloor} + \frac{2^{\lfloor n/2 \rfloor} - 1}{2 - 1}$$

$$\geq 2^{\lfloor n/2 \rfloor}$$

Sfruttiamo il fatto che $T(n)$ è monotona crescente

La ricorsione termina quando $k = n/2$

Possiamo fare di meglio?

- Utilizziamo un vettore di lunghezza n per calcolare e memorizzare i valori di F_1, F_2, \dots, F_n

```
algoritmo Fibonacci3(int n) → int
  Sia Fib[1..n] un array di n interi
  Fib[1] := 1;
  Fib[2] := 1;
  for i:=3 to n do
    Fib[i] := Fib[i-1] + Fib[i-2];
  endfor
  return Fib[n];
```

Quanto costa?

- Stiamiamo il “costo” dell'algorithmo Fibonacci3 contando le righe di (pseudo-)codice eseguite

```
algoritmo Fibonacci3(int n) → int
  Sia Fib[1..n] un array di n interi
  Fib[1] := 1; // ..... 1 volta
  Fib[2] := 1; // ..... 1 volta
  for i:=3 to n do // ..... (n-1) volte
    Fib[i] := Fib[i-1] + Fib[i-2]; // (n-2) volte
  endfor
  return Fib[n]; // ..... 1 volta
  // Totale..... 2n
```

- Tempo proporzionale a n
- Spazio proporzionale a n

Possiamo fare di meglio?

- L'occupazione di memoria di Fibonacci3 è proporzionale a n . Possiamo ridurla?
- Sì, se osserviamo che per calcolare F_n basta ricordare i due valori precedenti F_{n-1} e F_{n-2}

```
algoritmo Fibonacci4(int n) → int
if ( n==1 || n==2 ) then
    return 1;
else
    F_nm1 := 1;
    F_nm2 := 1;
    for i:=3 to n do
        F_n := F_nm1 + F_nm2;
        F_nm2 := F_nm1;
        F_nm1 := F_n;
    endfor
    return F_n;
endif
```

Quanto costa?

- Anche qui, contiamo il numero di righe di codice eseguite

```
algoritmo Fibonacci4(int n) → int
if ( n==1 || n==2 ) then
    return 1;
else
    F_nm1 := 1; // ..... 1 volta
    F_nm2 := 1; // ..... 1 volta
    for i:=3 to n do // ..... (n-1) volte
        F_n := F_nm1 + F_nm2; // . (n-2) volte
        F_nm2 := F_nm1; // ..... (n-2) volte
        F_nm1 := F_n; // ..... (n-2) volte
    endfor
    return F_n; // ..... 1 volta
endif

// Totale..... 4n-4
```

Ora siamo a posto... o no?

- Consideriamo la matrice A definita come segue:

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

- Teorema: Per ogni $n \geq 2$, si ha:

$$A^{n-1} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} = \begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix}$$

(Si dimostra per induzione)

Algoritmo Fibonacci6

- Sfruttiamo il teorema precedente per descrivere l'algoritmo Fibonacci6 come segue

```
algoritmo Fibonacci6(int n) : int  
     $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$   
    M = MatPow( A, n-1 );  
    return M[1][1];
```

M[1][1] è il primo
elemento della
prima riga

Algoritmo MatPow

- Per calcolare la potenza di una matrice, sfruttiamo il fatto che, se k è pari, $A^k = (A^{k/2})^2$

```
algoritmo MatPow(Matrice A, int k) → Matrice
if ( k==0 ) then
    
$$M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

else
    if ( k è pari ) then
        tmp := MatPow(A, k/2)
        M := tmp '*' tmp;
    else
        tmp := MatPow(A, (k-1)/2);
        M := tmp '*' tmp '*' A;
    endif
endif
return M;
```

L'operatore '*'
effettua il prodotto
di matrici

Riassunto

Tempo
Esponenziale

Tempo
Logaritmico

| Algoritmo | Tempo | Memoria |
|------------|-------------------|-------------|
| Fibonacci2 | $\Omega(2^{n/2})$ | $O(n)$ |
| Fibonacci3 | $O(n)$ | $O(n)$ |
| Fibonacci4 | $O(n)$ | $O(1)$ |
| Fibonacci6 | $O(\log n)$ | $O(\log n)$ |

Cosa abbiamo imparato?

- Per lo stesso problema, siamo partiti da un algoritmo inefficiente (costo esponenziale) per arrivare ad un algoritmo molto efficiente (costo logaritmico)
- La scelta dell'algoritmo più appropriato fa la differenza tra poter risolvere un problema e **NON** poterlo risolvere

Esercizio “di riscaldamento”

- È dato un array $A[1..n-1]$ contenente una permutazione degli interi da 1 a n (estremi inclusi) a cui è stato tolto un elemento; i valori in A possono comparire in un ordine qualsiasi
 - Es: $A = [1, 3, 4, 5]$ è una permutazione di $1..5$ a cui è stato tolto il numero 2
 - Es: $A = [7, 1, 3, 5, 4, 2]$ è una permutazione di $1..7$ a cui è stato tolto il numero 6
- Scrivere un elemento che dato l'array $A[1..n-1]$, individua il valore nell'intervallo $1..n$ che non compare in A .