# Algoritmi e Strutture Dati

Luciano Gualà
guala@mat.uniroma2.it
www.mat.uniroma2.it/~guala

# Informazioni utili

- **Orario lezioni**
  - Lunedì: 11,00 – 13,00
  - mercoledì: 9,00 – 11,00
- **Orario ricevimento**
  - lunedì: 14,45 – 16,15 (o su appuntamento)
  - Ufficio: dip. di matematica, piano 0, corridoio B0, stanza 206

# Struttura del corso

- Corso strutturato in due moduli
  - Modulo I (vecchio Elementi di Algoritmi e Strutture Dati)
    - 6 CFU
    - Ottobre – Gennaio
  - Modulo II (vecchio Algoritmi e Strutture dati con Laboratorio)
    - 6 CFU
    - Marzo – Giugno

# Prerequisiti del corso

## Cosa è necessario sapere…

- programmazione di base
- strutture dati elementari
- concetto di ricorsione
- dimostrazione per induzione e calcolo infinitesimale

## Propedeuticità

- programmazione
- analisi matematica
- matematica discreta

# Slide e materiale didattico

http://www.mat.uniroma2.it/~guala/

# Libri di testo

C. Demetrescu, I. Finocchi, G. Italiano
Algoritmi e Strutture dati (sec. ed.)
McGraw-Hill

P. Crescenzi, G. Gambosi, R. Grossi, G. Rossi
Strutture di dati e algoritmi
Pearson

S. Dasgupta, C. Papadimitriou, U. Vazirani
Algorithms , McGraw-Hill

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein
Introduzione agli algortimi e strutture dati
McGraw-Hill

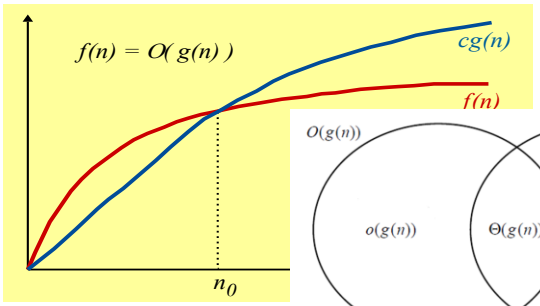
A. Bertossi, A. Montresor
Algoritmi e strutture di dati
Città Studi

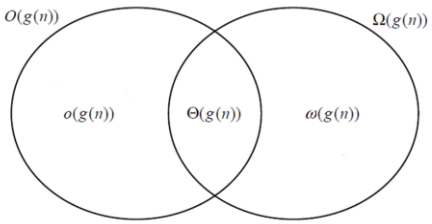
J. Kleinberg, E. Tardos
Algorithm Design
Addison Wesley

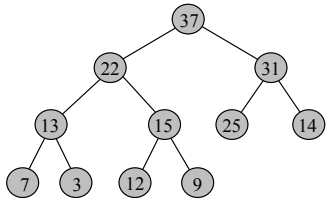
# Modalità d'esame

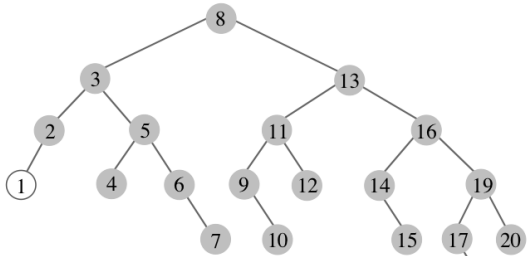
- L'esame consiste in una prova scritta e una prova orale (per ogni modulo)
- Quattro appelli
  - 2 giugno/luglio
  - 1 settembre
  - 1 gennaio/febbraio
- Prova parziale a febbraio
- Per sostenere l'esame è **obbligatorio** prenotarsi online (una settimana prima) su delphi.uniroma2.it

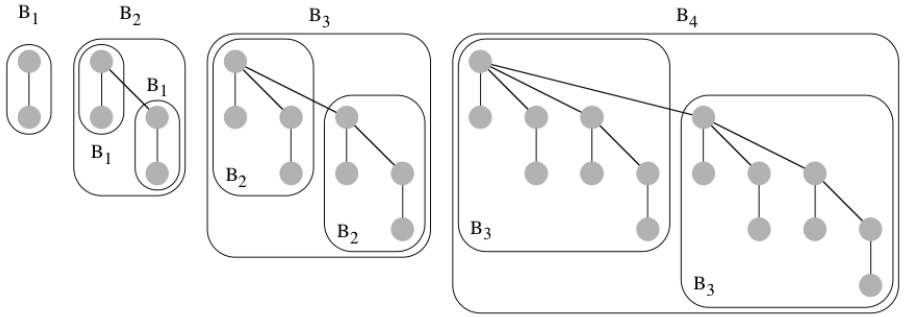
# Teoria degli algoritmi piena di idee bellissime

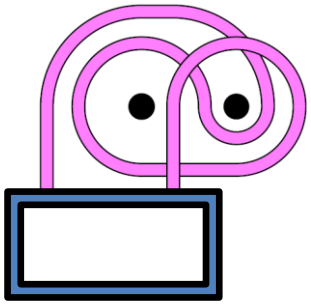


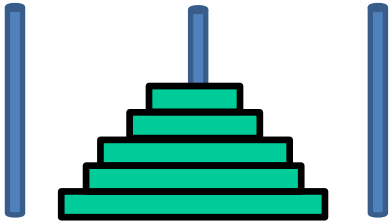



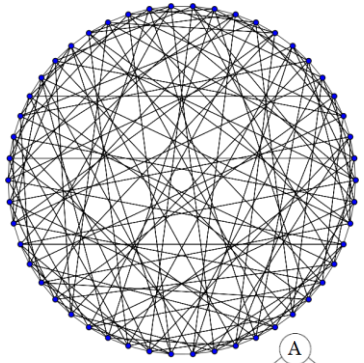



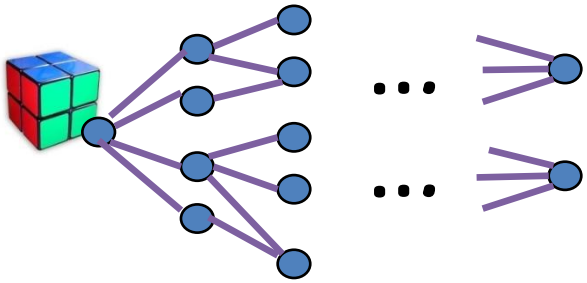



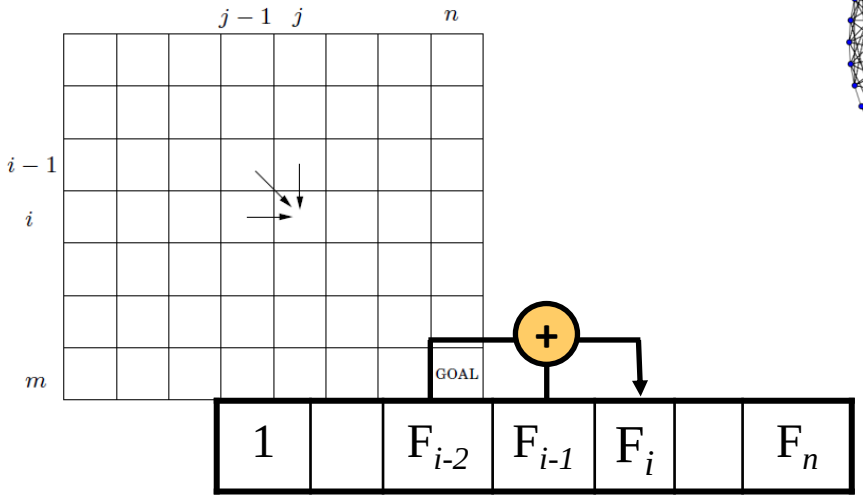



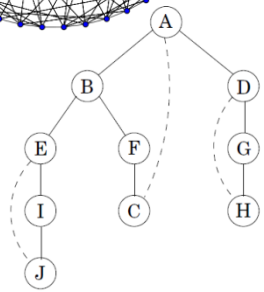



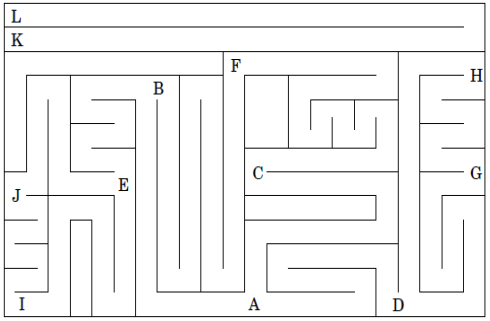








# Qualche consiglio:

- Studiare giorno per giorno


- Lavorare sui problemi assegnati in gruppo

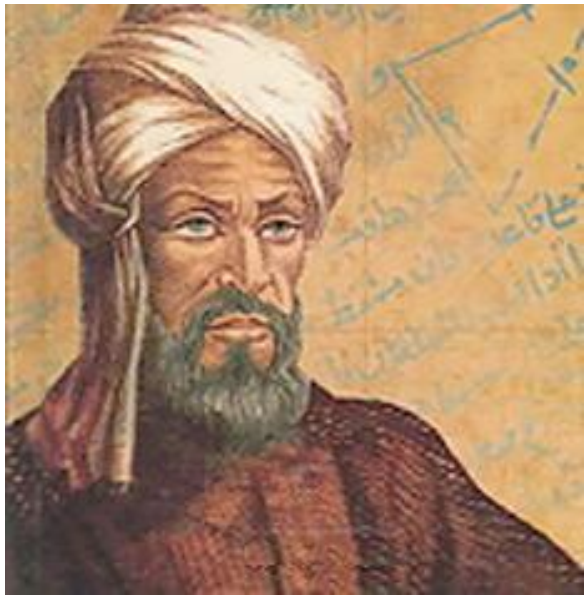
- Scrivere/formalizzare la soluzione individualmente

- Cercate di divertivi!

# Algoritmo

Procedimento che descrive una sequenza di passi ben definiti finalizzato a risolvere un dato problema (computazionale).

# etimologia



Il termine *Algoritmo* deriva da *Algorismus*, traslitterazione latina del nome di un matematico persiano del IX secolo, Muhammad al-Khwarizmi, che descrisse delle procedure per i calcoli matematici

# Algoritmi e programmi

- Un algoritmo può essere visto come l'essenza computazionale di un programma, nel senso che fornisce il procedimento per giungere alla soluzione di un dato problema di calcolo

- Algoritmo diverso da programma
  - programma è la codifica (in un linguaggio di programmazione) di un algoritmo
  - un algoritmo può essere visto come un programma distillato da dettagli riguardanti il linguaggio di programmazione, ambiente di sviluppo, sistema operativo
  - Algoritmo è un concetto autonomo da quello di programma

# Cosa studieremo?

…ad analizzare e progettare "buoni" algoritmi

…che intendiamo per "buoni"?

- Corretti: producono correttamente il risultato desiderato
- Efficienti: usano poche risorse di calcolo, come tempo e memoria.

algoritmi veloci!

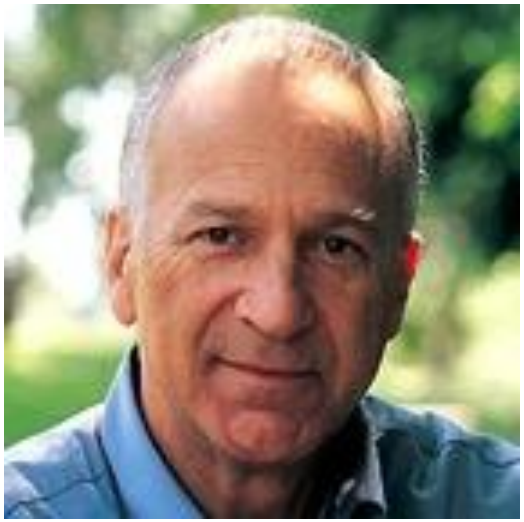
# Cosa è (più) importante oltre l'efficienza?

- Correttezza
- Semplicità
- Mantenibilità
- Stabilità
- Modularità
- Sicurezza
- User-friendliness
- …

Allora perché tanta enfasi sull'efficienza?

- Veloce è bello
- A volte: o veloce o non funzionale
- Legato alla User-friendliness
- Efficienza può essere usata per "pagare" altre caratteristiche

# Altri motivi per studiare gli algoritmi



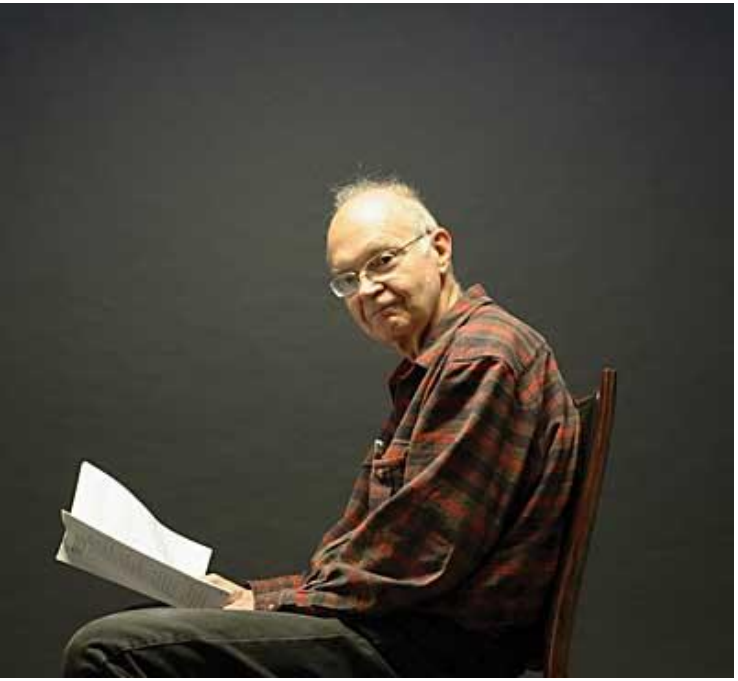
"L'algoritmica è l'anima dell'informatica."

David Harel

Le idee algoritmiche non solo trovano soluzioni a problemi ben posti, quanto costituiscono il linguaggio che porta ad esprimere chiaramente il problema soggiacente

# Altri motivi per studiare gli algoritmi

## importanza teorica



"Se è vero che un problema non si capisce a fondo finché non lo si deve insegnare a qualcuno altro, a maggior ragione nulla è compreso in modo più approfondito di ciò che si deve insegnare ad una macchina, ovvero di ciò che va espresso tramite un algoritmo."

Donald Knuth

In ogni algoritmo è possibile individuare due componenti fondamentali:

- l'identificazione della appropriata tecnica di progetto algoritmico (basato sulla struttura del problema);
- la chiara individuazione del nucleo matematico del problema stesso.

# Altri motivi per studiare gli algoritmi

## importanza pratica

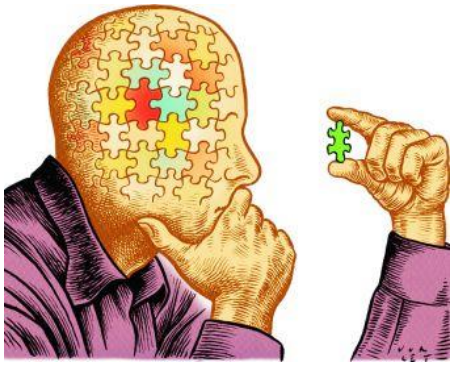
"There is a saying: If you want to be a good programmer, you just program every day for two years, you will be an excellent programmer. If you want to be a world-class programmer, you can program every day for ten years. Or you can program every day for two years and take an algorithms class."

Charles E. Leiserson

# Altri motivi per studiare gli algoritmi

Potenzia le capacità di:

- ***Critical Thinking***:
  - un modo di decidere se un certo enunciato è sempre vero, vero a volte, parzialmente vero, o falso



- ***Problem Solving***:
  - insieme dei processi atti ad analizzare, affrontare e risolvere positivamente problemi

complessità
temporale

# alcuni concetti di cui non è sempre facile parlare

algoritmo

istanza

efficienza

problema

modello di
calcolo

dimensione
dell'istanza

caso
peggiore

correttezza

# un puzzle può aiutare; eccone uno famoso

n monete tutte identiche d'aspetto
una delle monete è falsa e pesa
leggermente più delle altre

ho a disposizione solo una
bilancia a due piatti

obiettivo: individuare la moneta falsa
(facendo poche pesate)

# tornando ai concetti fondamentali

**problema:** individuare una moneta falsa fra n monete

**istanza:** n specifiche monete; quella falsa è una di queste; può essere la "prima", la "seconda", ecc.

**dimensione dell'istanza:** il valore n

**modello di calcolo:** bilancia a due piatti. specifica quello che si può fare

**algoritmo:** strategia di pesatura. La descrizione deve essere "comprensibile" e "compatta". Deve descrivere la sequenza di operazioni sul modello di calcolo eseguite per una generica istanza

**correttezza dell'algoritmo:** la strategia di pesatura deve funzionare (individuare la moneta falsa) per una generica istanza, ovvero indipendentemente da quante monete sono, e se la moneta falsa è la "prima", la "seconda", ecc.

# tornando ai concetti fondamentali

**complessità temporale (dell'algoritmo):** # di pesate che esegue prima di individuare la moneta falsa. Dipende dalla dimensione dell'istanza e dall'istanza stessa.

**complessità temporale nel caso peggiore:** # massimo di pesate che esegue su una istanza di una certa dimensione. E' una delimitazione superiore a quanto mi "costa" risolvere una generica istanza. Espressa come funzione della dimensione dell'istanza.

**efficienza (dell'algoritmo):** l'algoritmo deve fare poche pesate, deve essere cioè veloce. Ma veloce rispetto a che? quando si può dire che un algoritmo è veloce?

## Alg1

uso la prima moneta e la confronto con le altre

n=7

1 2 3 4 5 6 7

# Alg1

uso la prima moneta e la confronto con le altre

n=7

1 2 3 4 5 6 7

## Alg1

uso la prima moneta e la confronto con le altre

n=7

1 2 3 4 5 6 7

## Alg1

uso la prima moneta e la confronto con le altre

n=7

1 2 3 4 5 6 7

## Alg1

uso la prima moneta e la confronto con le altre

n=7

1 2 3 4 5 6 7

## Alg1

uso la prima moneta e la confronto con le altre


n=7

1 2 3 4 5 6 7

## Alg1

uso la prima moneta e la confronto con le altre

n=7

1 2 3 4 5 6 7

# Alg1

uso la prima moneta e la confronto con le altre


n=7

1 2 3 4 5 6 7

due parole sui costrutti:
sequenziamento, condizionale, ciclo

Alg1 ($X=\{x_1, x_2, \ldots, x_n\}$)

1. **for** i=2 **to** n **do**
2. **if** peso($x_1$) > peso($x_i$) **then return** $x_1$
3. **if** peso($x_1$) < peso($x_i$) **then return** $x_i$

Corretto? sì!  # pesate? dipende!

nel caso peggiore? n-1

efficiente? …boh?!

posso fare meglio?

Oss: l'ultima pesata non serve ➡ n-2 pesate

…mah!

# Alg2

peso le monete a coppie

n=7

1 2 3 4 5 6 7

# Alg2

peso le monete a coppie

n=7

1 2 3 4 5 6 7

# Alg2

peso le monete a coppie

n=7

1 2 3 4 5 6 7

# Alg2

peso le monete a coppie


n=7

1 2 3 4 5 6 7

# Alg2

peso le monete a coppie

n=7

1 2 3 4 5 6 7

# Alg2

peso le monete a coppie

n=7

1 2 3 4 5 6 7

# Alg2

peso le monete a coppie

n=7

1 2 3 4 5 6 7

# Alg2

peso le monete a coppie


n=7

1 2 3 4 5 6 7

Alg2 ($X=\{x_1, x_2, \ldots, x_n\}$)

1. $k=\lfloor n/2 \rfloor$
2. **for** i=1 **to** k **do**
3. **if** peso($x_{2i-1}$) > peso($x_{2i}$) **then return** $x_{2i-1}$
4. **if** peso($x_{2i-1}$) < peso($x_{2i}$) **then return** $x_{2i}$
5. *//ancora non ho trovato la moneta falsa; n è dispari*
   *//e manca una moneta*
   **return** $x_n$

Corretto? sì! # pesate? dipende!

nel caso peggiore? $\lfloor n/2 \rfloor$

efficiente? ...boh?!

però meglio di *Alg1* ☺

posso fare meglio?

# Alg3

peso le monete
dividendole ogni volte in
due gruppi

n=7

1 2 3 4 5 6 7

# Alg3

peso le monete
dividendole ogni volte in
due gruppi

n=7

1 2 3 4 5 6 7

# Alg3

peso le monete
dividendole ogni volte in
due gruppi


n=7

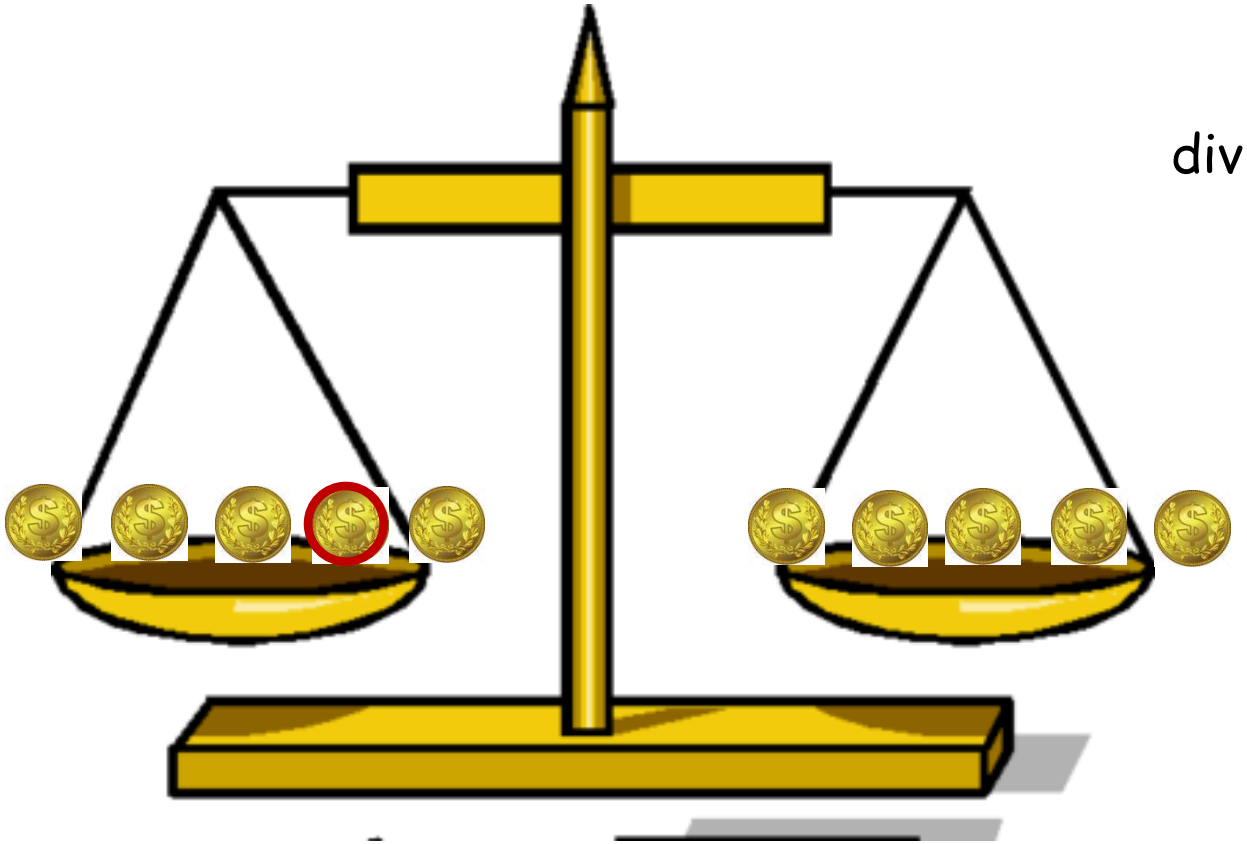
1 2 3 4 5 6 7

# Alg3

peso le monete
dividendole ogni volte in
due gruppi

n=10

1 2 3 4 5 6 7 8 9 10



# Alg3

peso le monete
dividendole ogni volte in
due gruppi

n=10

1 2 3 4 5 6 7 8 9 10

# Alg3

peso le monete
dividendole ogni volte in
due gruppi

n=10

1 2 3 4 5 6 7 8 9 10

# Alg3

peso le monete
dividendole ogni volte in
due gruppi


n=10

1 2 3 4 5 6 7 8 9 10

Alg3(X)

1. **if** (|X|=1) **then** return unica moneta in X
2. dividi X in due gruppi $X_1$ e $X_2$ di (uguale) dimensione $k = \lfloor |X|/2 \rfloor$ e se |X| è dispari una ulteriore moneta y
3. **if** peso($X_1$) = peso($X_2$) **then return** y
4. **if** peso($X_1$) > peso($X_2$) **then return** Alg3($X_1$)
   **else return** Alg3($X_2$)

Corretto? sì!

# pesate nel caso peggiore? $\lfloor \log_2 n \rfloor$ (da argomentare)

efficiente? …boh?!

però meglio di *Alg2*

☺

# Alg3: analisi della complessità

P(n): # pesate che Alg3 esegue nel caso peggiore su un'istanza di dimensione n

$P(n) = P(\lfloor n/2 \rfloor) + 1$ $\qquad$ $P(1)=0$

Oss.: P(x) è una funzione non decrescente in x

$$
\begin{aligned}
P(n) &= P(\lfloor n/2 \rfloor) + 1 \\
&= P(\lfloor (1/2)\lfloor n/2 \rfloor \rfloor) + 2 \\
&\leq P(\lfloor n/4 \rfloor) + 2 \\
&\leq P(\lfloor n/8 \rfloor) + 3 \\
&\leq P(\lfloor n/2^i \rfloor) + i \qquad \text{per } i = \lfloor \log_2 n \rfloor \\
&\leq P(1) + \lfloor \log_2 n \rfloor = \lfloor \log_2 n \rfloor
\end{aligned}
$$

Oss.: vale
$\lfloor (1/2)\lfloor n/2 \rfloor \rfloor \leq \lfloor (1/2)\, n/2 \rfloor \leq \lfloor n/4 \rfloor$

quando $\lfloor n/2^i \rfloor = 1$?

Una domanda: quanto è più veloce Alg3 rispetto agli altri?

assunzione: ogni pesata richiede un minuto

TABELLA

| n | 10 | 100 | 1.000 | 10.000 | 100.000 |
|---|---|---|---|---|---|
| Alg1 | 9m | ~ 1h, 39m | ~16 h | ~7gg | ~69gg |
| Alg2 | 5 m | ~ 50 min | ~8 h | ~3,5gg | ~35gg |
| Alg3 | 3 m | 6m | 9m | 13m | 16m |

posso fare meglio di Alg3?

## Alg4

posso dividere in tre gruppi invece di due

n=7

1 2 3 4 5 6 7

# Alg4

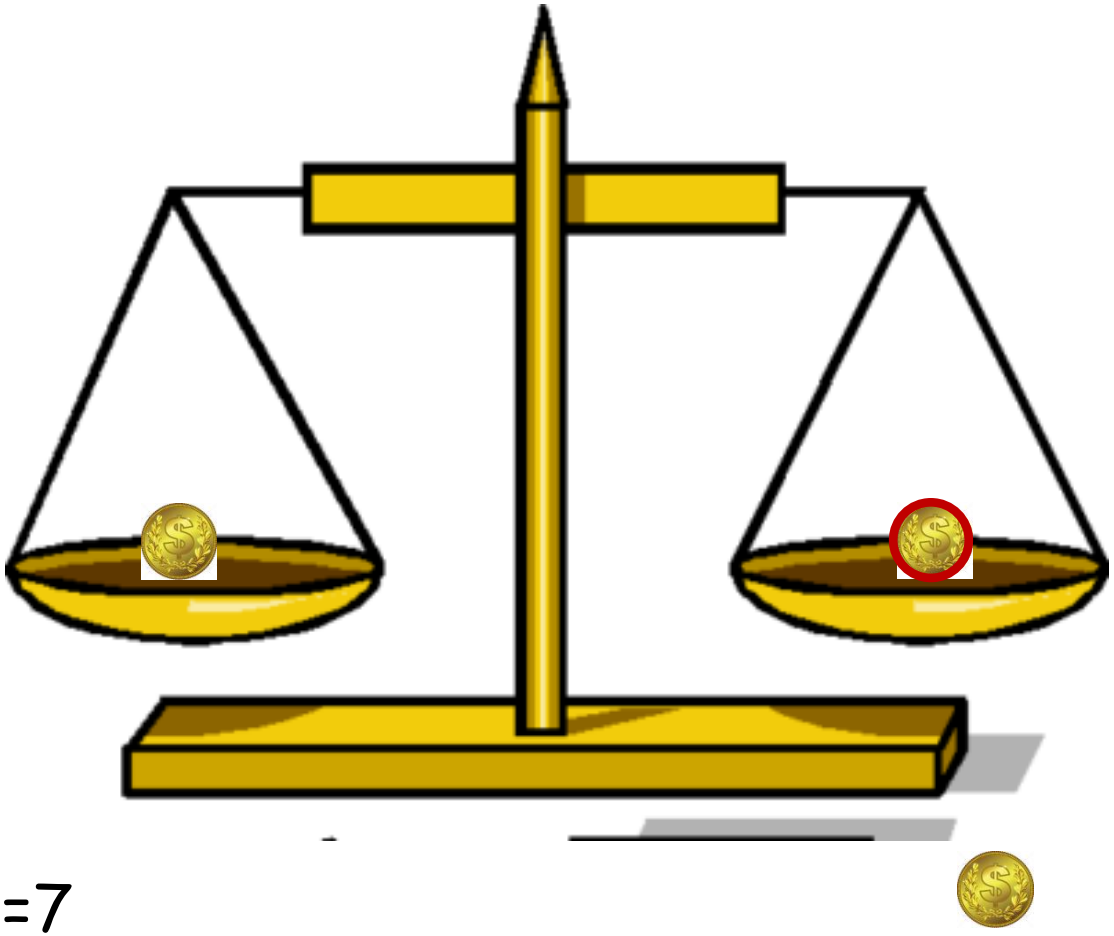
posso dividere in tre gruppi invece di due

n=7

1 2 3 4 5 6 7

# Alg4

posso dividere in tre gruppi invece di due




n=7

1 2 3 4 5 6 7

# Alg4

posso dividere in tre gruppi invece di due

n=10

1 2 3 4 5 6 7 8 9 10

# Alg4

posso dividere in tre gruppi invece di due

n=10

1 2 3 4 5 6 7 8 9 10

# Alg4

posso dividere in tre gruppi invece di due


n=10

1 2 3 4 5 6 7 8 9 10

Alg4 (X)

1. **if** (|X|=1) **then** return unica moneta in X
2. dividi X in tre gruppi $X_1$, $X_2$, $X_3$ di dimensione bilanciata siano $X_1$ e $X_2$ i gruppi che hanno la stessa dimensione (ci sono sempre)
3. **if** peso($X_1$) = peso($X_2$) **then return** Alg4($X_3$)
4. **if** peso($X_1$) > peso($X_2$) **then return** Alg4($X_1$)
   **else return** Alg4($X_2$)

Corretto? sì!

# pesate nel caso peggiore? $\lceil \log_3 n \rceil$ (da argomentare)

efficiente? …boh?!

però meglio di *Alg3*

☺

# Alg4: analisi della complessità

P(n): # pesate che Alg4 esegue nel caso peggiore su un'istanza di dimensione n

$P(n) = P(\lceil n/3 \rceil) + 1$ $\qquad$ $P(1)=0$

Oss.: P(x) è una funzione non decrescente in x

sia k il più piccolo intero tale che $3^k \geq n$ $\qquad$ $n'=3^k$

$\Rightarrow \quad k \geq \log_3 n \quad \xrightarrow{k \text{ intero}} \quad k = \lceil \log_3 n \rceil$

$P(n) \leq P(n') = k = \lceil \log_3 n \rceil$

$P(n') = P(n'/3) + 1$

$= P(n'/9) + 2$

$= P(n'/3^i) + i$

$= P(1) + k = k$ $\qquad$ per i=k

…torniamo alla tabella: quanto è più veloce Alg4 rispetto agli altri?

assunzione: ogni pesata richiede un minuto

TABELLA

| n | 10 | 100 | 1.000 | 10.000 | 100.000 |
|---|---|---|---|---|---|
| Alg1 | 9m | ~ 1h, 39m | ~16 h | ~6gg | ~69gg |
| Alg2 | 5 m | ~ 50 m | ~8 h | ~3,5gg | ~35gg |
| Alg3 | 3 m | 6m | 9m | 13m | 16m |
| Alg4 | 3 m | 5m | 7m | 9m | 11m |

posso fare meglio di Alg4?

# Sui limiti della velocità: una delimitazione inferiore (lower bound) alla complessità del problema

## Teorema

Un qualsiasi algoritmo che correttamente individua la moneta falsa fra $n$ monete deve effettuare nel caso peggiore almeno $\lceil \log_3 n \rceil$ pesate.

la dimostrazione usa argomentazioni matematiche per mostrare che un generico algoritmo se è corretto deve avere almeno una certa complessità temporale nel caso peggiore.

dimostrazione elegante e non banale che usa la tecnica dell'albero di decisione di un problema (che vedremo durante il corso)

## Corollario

Alg4 è un algoritmo ottimo per il problema.

# Esercizio

Si devono cuocere n frittelle. Si ha a disposizione una padella che riesce a contenere due frittelle alla volta. Ogni frittella va cotta su tutte e due i lati e ogni lato richiede un minuto.
Progettare un algoritmo che frigge le frittelle nel minor tempo possibile. Si argomenti, se possibile, sulla ottimalità dell'algoritmo proposto.


Buon inizio anno!