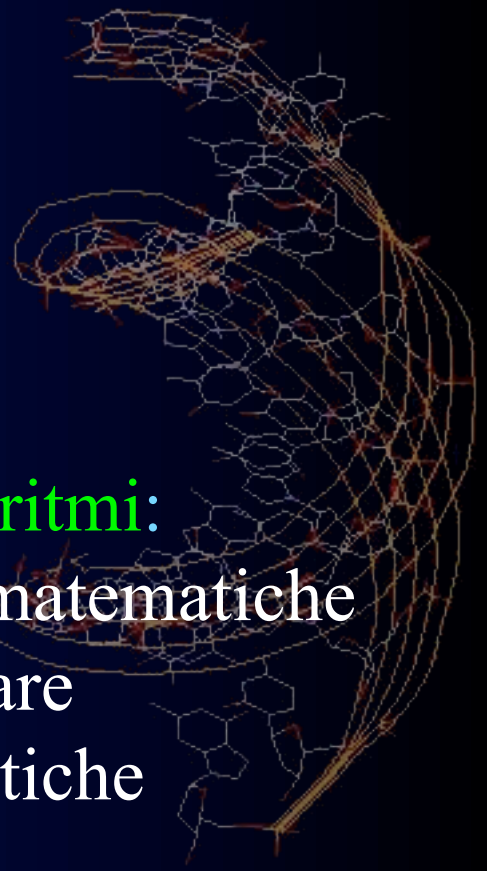
Algoritmi e loro proprietà

- Proprietà formali degli Algoritmi
- Efficienza rispetto al tempo
- Efficienza rispetto allo spazio

Cos'è l'informatica ?

L'informatica è la scienza della rappresentazione e dell'elaborazione dell'informazione



L'informatica è lo studio degli algoritmi:

- delle loro proprietà formali e matematiche
- delle loro realizzazioni hardware
- delle loro realizzazioni linguistiche
- delle loro applicazioni

Che cos'è un algoritmo ?

Un insieme ben *ordinato* e *finito*
di operazioni *non ambigue*
ed *effettivamente calcolabili* che,
applicate ad un insieme di condizioni iniziali,
produce un *risultato* e
termina in una *quantità di tempo finita*.

Un esempio di algoritmo

La somma dei primi 100 numeri interi (da 0 a 99!):

1. Poni *somma* = 0
2. Poni *indice* = 1
3. Finché *indice* è minore di 100 ripeti i passi 4. e 5.
    4. Aggiungi *indice* a *somma*
    5. Incrementa *indice* di 1
6. Stampa *somma*
7. Fermati

Proprietà formali e matematiche degli algoritmi

La prima proprietà è ovviamente la *correttezza*

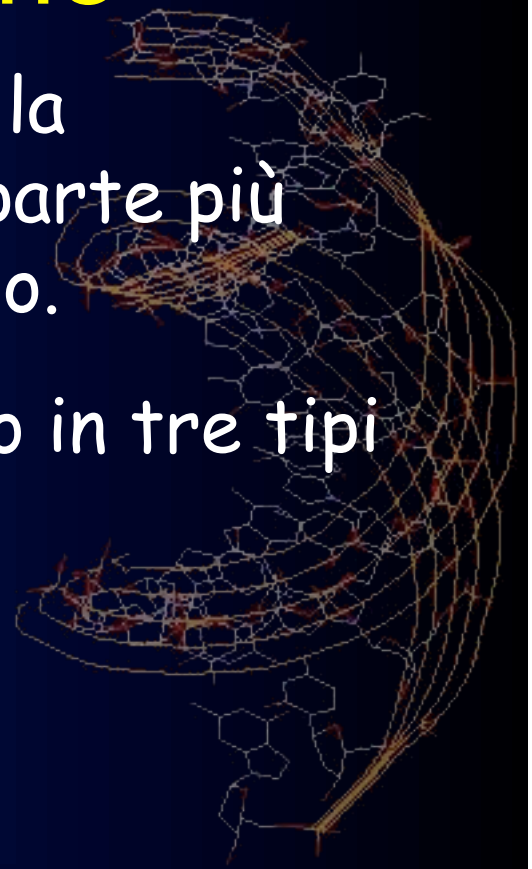
Ma ha una fondamentale importanza anche *l'efficienza*, che si misura rispetto alla risorsa *spazio* e rispetto alla risorsa *tempo*

Scrivere un algoritmo

La ricerca del giusto algoritmo per la soluzione di un dato problema è la parte più creativa del lavoro di un informatico.

Ogni algoritmo può essere scomposto in tre tipi fondamentali di operazioni:

- Operazioni sequenziali
- Operazioni condizionali
- Operazioni iterative



Operazioni *sequenziali*

- Istruzioni di elaborazione
  - Esempi:

    Poni *indice* = 1

    Aggiungi *indice* a *somma*

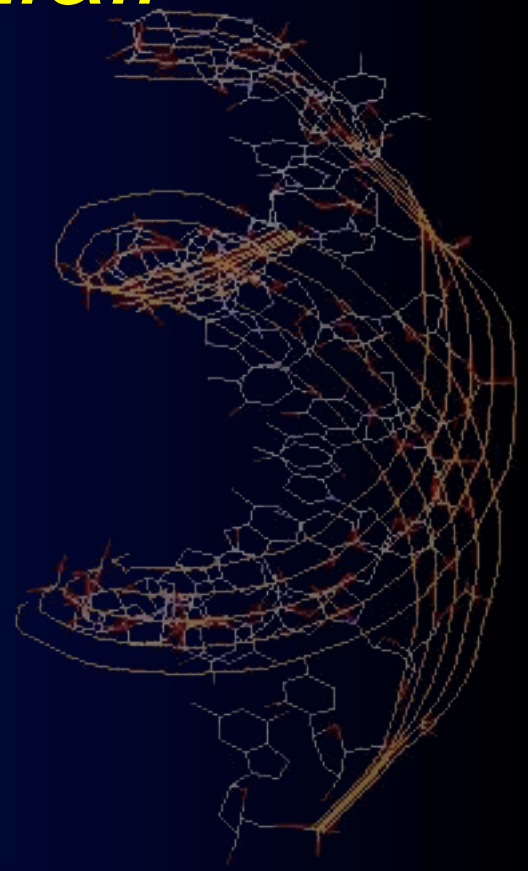
    Incrementa *indice* di 1

- Istruzioni di Input/Output (I/O)
  - Esempi:

    Acquisisci *misura*

    Acquisisci *errore assoluto*

    Stampa *errore relativo*



Operazioni *condizionali*

Esempio:

1. Se *misura* ≠ 0 allora
   2. Esegui algoritmo trova errore relativo
3. Altrimenti
   4. Stampa il messaggio "*errore relativo non definito*"
5. . . . . .

Operazioni *iterative*

Ciclo "do While":

Inizio ciclo: esegui

. . . . .

Fine del ciclo

Fino a che (condizione) rimane vera ripeti ciclo

eseguito almeno 1 volta

Ciclo While

Mentre (condizione) rimane vera ripeti:

. . . . . .

Fine del ciclo

eseguito 0 o più volte

Pseudocodice

Gli algoritmi presentati sono stati scritti secondo uno schema strutturato chiamato pseudocodice

Notate che:

- tutte le linee contengono un verbo specifico (azione)
- i passi sono numerati in modo tale che per ogni riga viene eseguita una sola azione
- se la linea richiede due passi viene suddivisa in linee successive indentate

Correttezza di un algoritmo

Determinare la correttezza dell'algoritmo elaborato per un dato problema può essere arduo…le condizioni per essere ragionevolmente sicuri di averlo trovato sono:

- Comprensione effettiva del problema
- Validità della soluzione indipendentemente
  da condizioni particolari o
  dal valore dei dati in ingresso
- Livello di approssimazione del risultato sufficiente agli scopi di progetto

Comprensione del problema

La condizione necessaria per trovare un corretto algoritmo a un problema è ovviamente la corretta formulazione e comprensione del problema

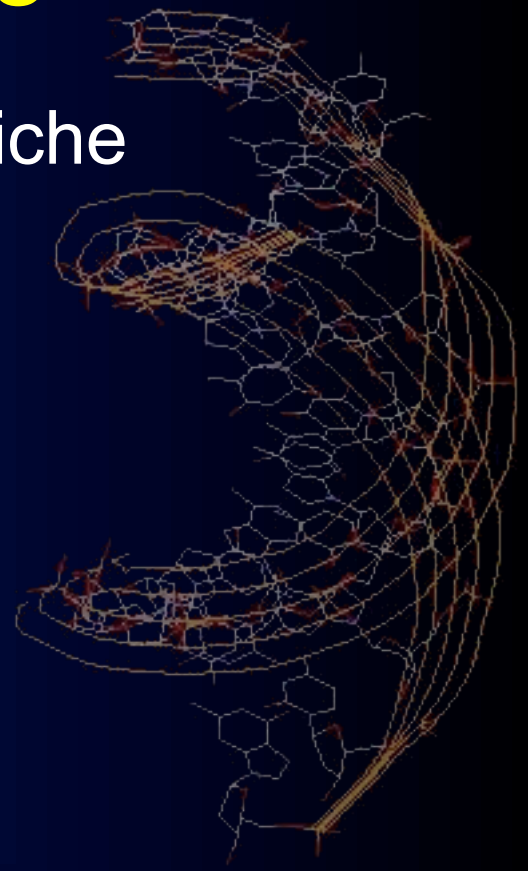
Esempio: Qual è la via più breve fra le città A e B ?

Algo 1: Esamina tutte le strade che congiungono A a B e scegli quella di lunghezza inferiore

Algo 2: Esamina tutte le strade che congiungono A a B e scegli quella il cui tempo di percorrenza medio è minimo

Dipendenza dai dati in ingresso

Algoritmo per risolvere equazioni algebriche di secondo grado:

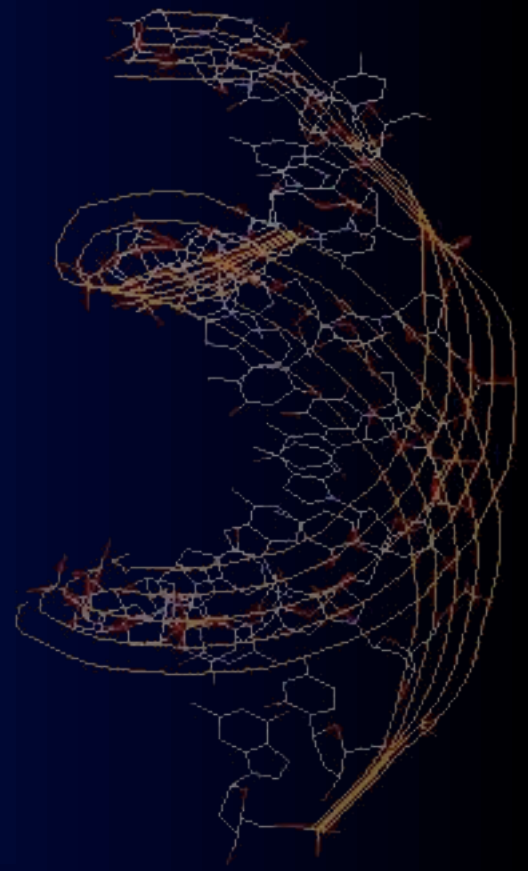


1. Acquisisci *a, b, c*
2. Poni $x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$

   $x_{21} = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$
3. Stampa x_1 , x_2
4. Fermati

Dipendenza dalle condizioni del problema

Esempio: gittata di un proiettile:



1. Acquisisci v, θ
2. Poni

$$G = \frac{2v^2 sin\theta \cos\theta}{g}$$

3. Stampa G
4. Fermati

…valido solo se il cannone è al suolo...

Livello di approssimazione richiesto

Spesso il risultato di un algoritmo è solo un'approssimazione, più o meno buona della risposta corretta.

Ad esempio se il risultato è un numero reale, un calcolatore potrà fornire solo un numero finito di cifre significative: occorrerà stabilire in base all'uso che della soluzione si deve fare quale approssimazione è accettabile come corretta.

Efficienza degli algoritmi

Una volta determinato un algoritmo corretto per la soluzione di un problema, occorre preoccuparsi della sua efficienza, ovvero di come esso gestisca le risorse

tempo (numero di operazioni necessarie alla soluzione) e

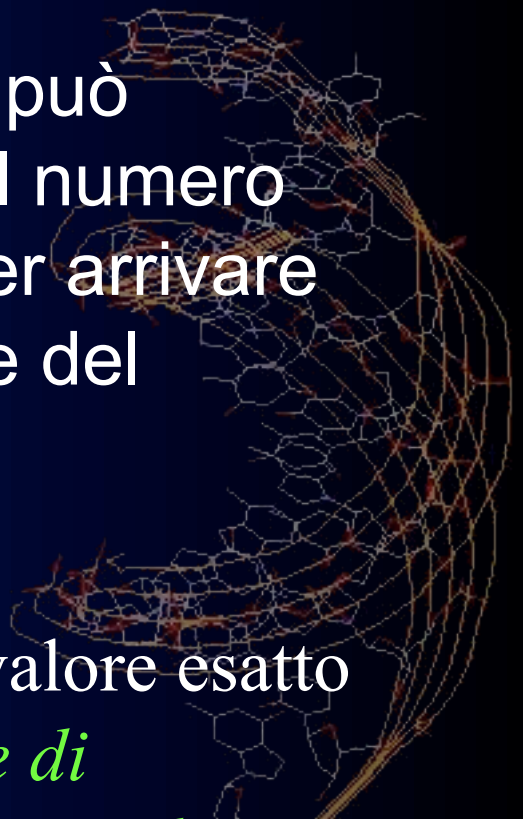
spazio (memoria occorrente per trovare la soluzione)

che saranno in quantità finita per qualunque elaboratore reale.

. . . . proseguendo

Un criterio per la valutazione dell'efficienza di un algoritmo e' di grande utilita' in quanto dobbiamo assicurarci la possibilita' di operare confronti tra diversi algoritmi, *indipendentemente* dal *linguaggio* con cui saranno implementati e dalla "*potenza*" della macchina su cui saranno eseguiti.

Valutare l'efficienza rispetto al tempo



L'efficienza rispetto alla risorsa tempo può essere valutata in generale contando il numero di operazioni richieste dall'algoritmo per arrivare alla soluzione del problema in funzione del numero *n* dei dati in ingresso.

In questa valutazione non è importante il valore esatto quanto la *dipendenza funzionale* e *l'ordine di grandezza*. Si parlerà allora di algoritmi o *complessita'* $O(n)$, $O(n^2)$, $O(\log n)$, $O(2^n)$ etc a seconda degli andamenti.

L’algoritmo di ricerca sequenziale...

Supponiamo di voler cercare un nome in una rubrica telefonica contenente 100 nomi.

Il primo algoritmo che viene in mente potrebbe essere:

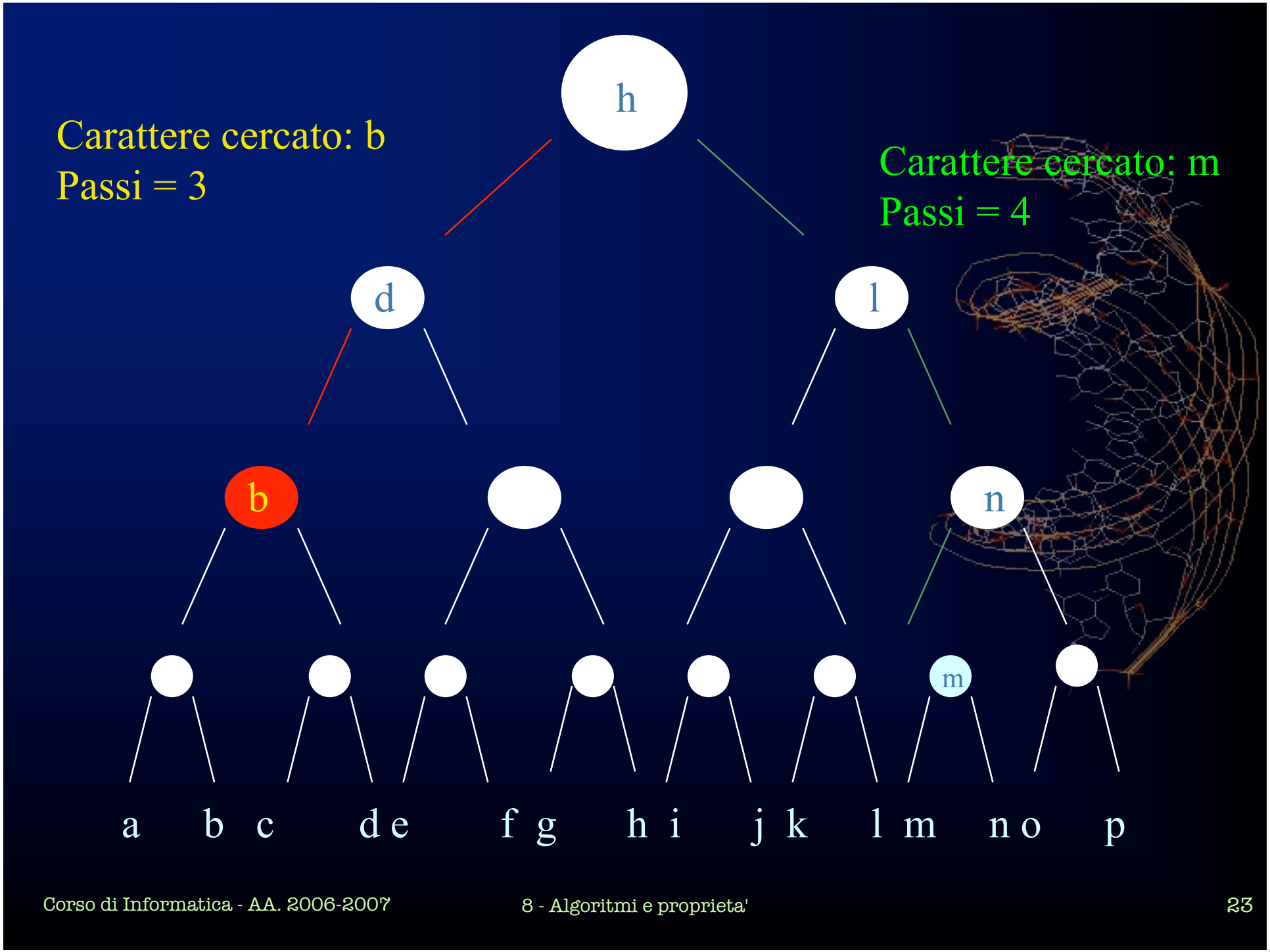
1. Acquisisci $nome_1 \ldots .. nome_{100}$
2. Acquisisci $tel_1 \ldots\ldots tel_{100}$
3. Acquisisci *nome cercato*
4. Poni *trovato* = falso
5. Poni $i = 1$
6. Ripeti finche *trovato* diventa vero o $i > 100$
    7. Se $nome_i$ = *nome cercato* allora
        8. Stampa tel_i
        9. Poni *trovato* = vero
        Altrimenti
        10. Incrementa *i* di 1
11. Fine del ciclo
12. Se *trovato* = falso allora
    13. Stampa messaggio 'Nome non in elenco'
14. Fermati

…e quello di ricerca binaria

L’algoritmo di ricerca sequenziale è semplice e corretto... ma non sfrutta il fatto che la rubrica è ordinata !

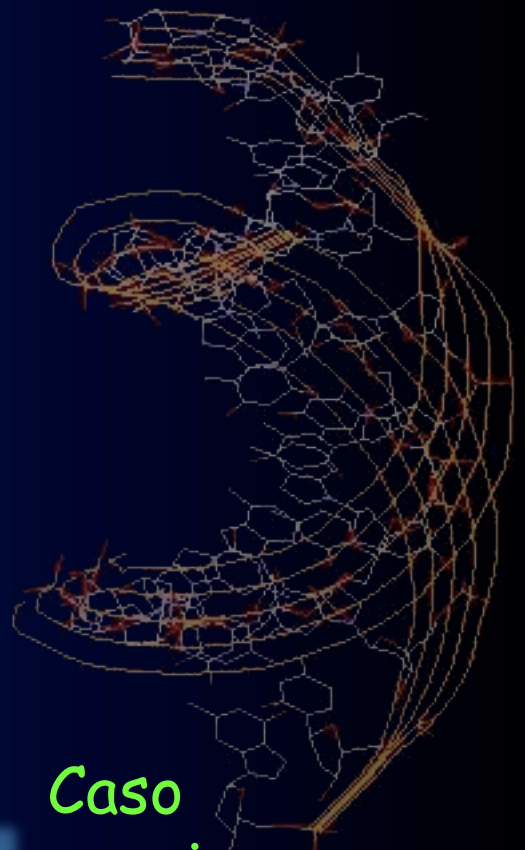
“Riesco a trovare i nomi molto più velocemente sul dizionario, da quando ho scoperto che sono in ordine alfabetico ” (Groucho Marx)

1. Acquisisci $nome_1 \ldots .nome_{100}$
2. Acquisisci $tel_1 \ldots \ldots tel_{100}$
3. Acquisisci *nome cercato*
4. Poni *trovato* = *falso*
5. Poni *inizio* = 1 e *fine* = 100
6. Ripeti finché *trovato* diventa vero o *fine*<*inizio*
    7. Poni *i* a (*inizio*+fine)/2
    8. Se $nome_i$ = *nome cercato* allora
        9. Stampa tel_i
        10. Poni *trovato* = *vero*
    Altrimenti
        Se *nome cercato* precede alfabeticamente $nome_i$
            11. Poni *fine* = *i -1*
        altrimenti (*nome cercato* segue $nome_i$)
            12. Poni *inizio* = *i +1*
13. Fine del ciclo
14. Se trovato = falso allora
    15.Stampa messaggio ‘ Nome non in elenco’
16. Fermati


Carattere cercato: b
Passi = 3
Carattere cercato: m
Passi = 4
h
d
l
b
n
m
a b c d e f g h i j k l m n o p

Ricerca Binaria: Complessita'

Passi (confronti)	Numero di dati per passo
1	n
2	n /2
3	$n/2/2 = n/2^2$
4	$n/2^3$
.....	.....
k	$n/2^{(k-1)}$
.....	.....
s	$1 = n/2^{(s-1)}$



Caso peggiore

$2^{(s-1)} = n \; ; \; s-1 = \log_2 n \; ; \; s = \log_2 n + 1 \Rightarrow O(\log_2 n)$

Confronto fra ricerca sequenziale e ricerca binaria

	CRAY T3E-900 1360 processori in parallelo	Pentium Pro 200
Costo	30 Milioni €	~ 50 €
Operazione/sec	$7*10^{11}$	$7*10^{7}$
Algoritmo	Ricerca sequenziale	ricerca binaria

Elenco di Napoli (10^6 abitanti): **Pentium: 7 volte più veloce**

Elenco di New York ($2*10^7$ abitanti): **Pentium: 120 volte più veloce**

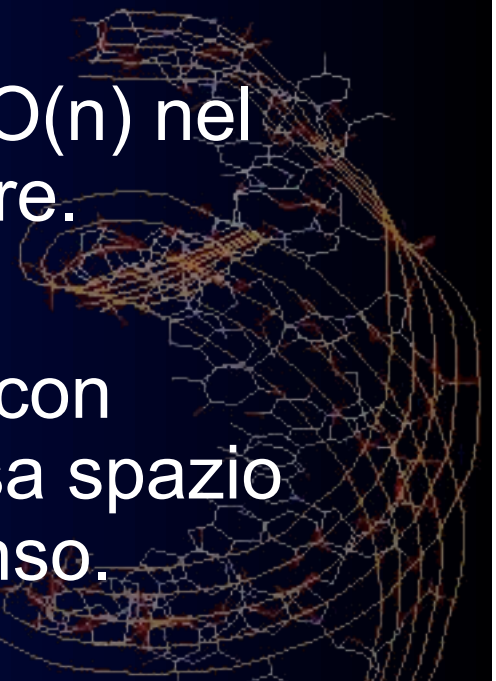
Ordini di grandezza

- gli algoritmi di complessita' di tipo polinomiale $O(n^k)$ producono una soluzione *trattabile* ad un problema;
- gli algoritmi di complessita' di tipo esponenziale $O(k^n)$ producono una soluzione *intrattabile*

$7 \cdot 10^7$ op. al secondo

	10	20	50	60
n	10^{-7} s	$2 \cdot 10^{-7}$ s	$5 \cdot 10^{-7}$ s	$6 \cdot 10^{-7}$ s
n^2	10^{-6} s	$4 \cdot 10^{-6}$ s	$25 \cdot 10^{-6}$ s	$36 \cdot 10^{-6}$ s
2^n	10^{-5} s	$1.6 \cdot 10^{-2}$ s	$1.6 \cdot 10^{7}$ s	$1.6 \cdot 10^{10}$ s

Efficienza rispetto allo spazio



- Tutti gli algoritmi mostrati finora sono O(n) nel numero di celle di memoria da utilizzare.
- In alcuni casi (ordinamento di elenchi con ricopiature e spostamenti etc) la risorsa spazio può essere utilizzata in modo più intenso.
- In generale esiste un trade-off tempo - spazio : la soluzione più efficiente per la risorsa tempo tende a utilizzare maggiormente la risorsa spazio e viceversa.

Riassumendo...

- Abbiamo definito gli algoritmi e visto alcuni esempi. Ci siamo preoccupati delle proprietà formali degli algoritmi.
- Abbiamo visto che gli algoritmi possono sempre essere ricondotti ad operazioni sequenziali-condizionali-iterative
- Abbiamo trovato dei criteri per studiare la correttezza e l'efficienza degli algoritmi.
- Abbiamo imparato che algoritmi di tipo esponenziale sono di fatto inutilizzabili.