# Elementi di Teoria della Computazione

## (ETC)

## A.A. 2024/25

Classe resto 1

Docenti: Prof. Luisa Gargano

Prof. Gianluca De Marco

BENVENUTI!

# Cosa è la "Computazione"?

Il termine computazione deriva dal latino *computatio-onis* che significa contare:

- *calcolo*

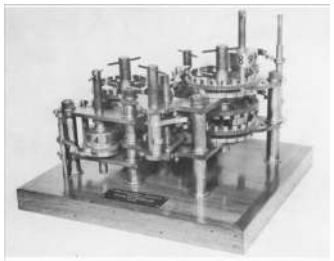
# Gli uomini hanno computato per migliaia di anni con

- Carta e Penna

```
  237
   84=
 ----
  321
```

- Abaco

- Strumenti meccanici



Calculating wheels di Charles Babbage.

- ….
- Calcolatori/Algoritmi/Programmi

# Cosa è la "Computazione"?

Il nome "algoritmo" deriva dalla traslitterazione latina di *Muhammad ibn Musa al-Khwarizmi*, uno studioso persiano del IX secolo i cui libri hanno introdotto al mondo occidentale il sistema numerale posizionale decimale, così come le soluzioni di equazioni  linearie e quadratiche

# Cosa è la "Computazione"?

Ecco come al-Khwarizmi ha descritto come risolvere un'equazione della forma $x^2$+bx=c:

*"radici e quadrati sono uguali a numeri": per esempio un quadrato e dieci radici dello stesso, ammontano a trentanove dirhems"* ($x^2$+10x=39) *cioè, quale deve essere il quadrato che, quando aumentato di dieci volte la propria radice, ammonta a trentanove? La soluzione è questa:*

- *si dimezza il numero delle radici, che in questa istanza è 5.*
- *Questo si moltiplica per se stesso; il prodotto è 25.*
- *Aggiungi questo a 39, la somma è 64.*
- *Ora prendi la radice di questo, che è 8, e sottrai metà del numero di radici, che è 5; il resto è 3.*
- *Questa è la radice del quadrato che hai cercato; il quadrato stesso è 9.*

Traduzione da "L'algebra di Ben-Musa", Fredric Rosen, 1831.

# Cosa è la "Computazione"?

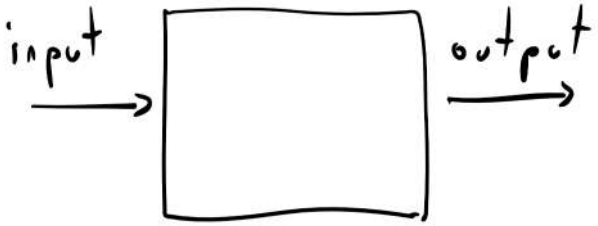
Per questo corso, serve modo molto più preciso per definire gli algoritmi.
Nel XX secolo nati formalismi esatti per descrivere algoritmi

Algoritmo di risoluzione delle equazioni quadratiche di al-Khwarizmi descritto in Python:

```
def solve_eq(b,c):
    # calcola soluzione di x² + bx = c  usando le istruzioni di  Al Khwarizmi
    val1 = b/2.0 #  divide il numero di radici
    val2 = val1*val1 #  moltiplicalo per se stesso
    val3 = val2 + c #  Aggiungi 39 (c)
    val4 = math.sqrt(val3) #  fanne la radice quadrata
    val5 = val4 - val1 # sottrai il risultato dalla metà delle radici
    return val5  # Questo è il valire cercato (x)
```

# Cosa è la "Computazione"?

La nostra nozione di base di computazione è un processo che associa un input a un output

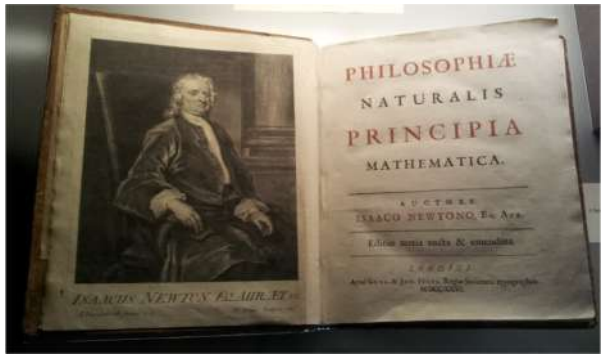



*mediante l'applicazione illimitata di un insieme finito di operazioni o regole*

Es. Linguaggio macchina, linguaggio C,.....

# ‘’Teoria della Computazione’’

Che cosa `e un computer? Che cosa `e un algoritmo / programma?
Quali sono le capacità, i limiti dei computer?




Una teoria per essere tale deve avere i seguenti requisiti:

- Precisione,
- Generalità

# "Teoria della Computazione"

- **Generalità:** è importante separare
  la domanda su **cosa** dobbiamo eseguire (cioè, la specifica)
  dalla domanda su **come** lo eseguiamo (cioè, l'implementazione).

  Dobbiamo definire la computazione
  - senza fa riferimento ad un calcolatore attuale
  - indipendentemente dai limiti odierni della scienza (ingegneria, fisica,…)

- **Precisione**
  - Dobbiamo definire formalmente un calcolatore
  - Dobbiamo dimostrare affermazioni circa ciò che può o non può essere computato

# "Teoria della Computazione"

*Nasce nel XX secolo*

- Formulare una *teoria* a partire dall'idea della *computazione*

Alan Turing

# "Teoria della Computazione"?

Avendo a disposizione risorse (memoria, tempo,…) sufficienti

- un *calcolatore* può risolvere qualsiasi problema?
- oppure esistono limiti fondamentali a ciò che si può computare?

# Computabilità:

*Quali problemi possono essere computati?*
*(con qualsiasi macchina, linguaggio, … )*

Esempi di problemi computazionali

- Problemi numerici
  - Data una stringa binaria, il numero di 1 è maggiore del numero di 0?
  - Dati due numeri x e y, calcola x+y
  - Dato un intero, risulta x primo?
- Problemi riguardanti programmi (es. in C)
  - Data una sequenza di caratteri ASCII, rispetta la sintassi del C?
  - Dato un programma in C, esiste un input che lo manda in loop?

# Computabilità:

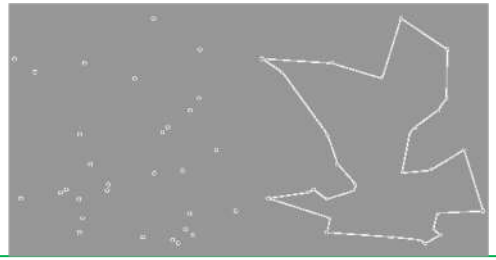
*Quali problemi possono essere computati?*
*(con qualsiasi macchina, linguaggio, … )*

**Esempi di problemi computazionali in matematica**

- **Equazioni Diofantine**: data un'equazione con una o più incognite e coefficienti interi (es $x^2+3xyz-37z^3-5=0$), essa ammette una soluzione intera?
- **Problema del commesso viaggiatore**: data una rete di città, connesse tramite delle strade, trovare il percorso di minore lunghezza che un commesso viaggiatore deve seguire per visitare

  tutte le città una e una sola volta

  per poi tornare alla città di partenza



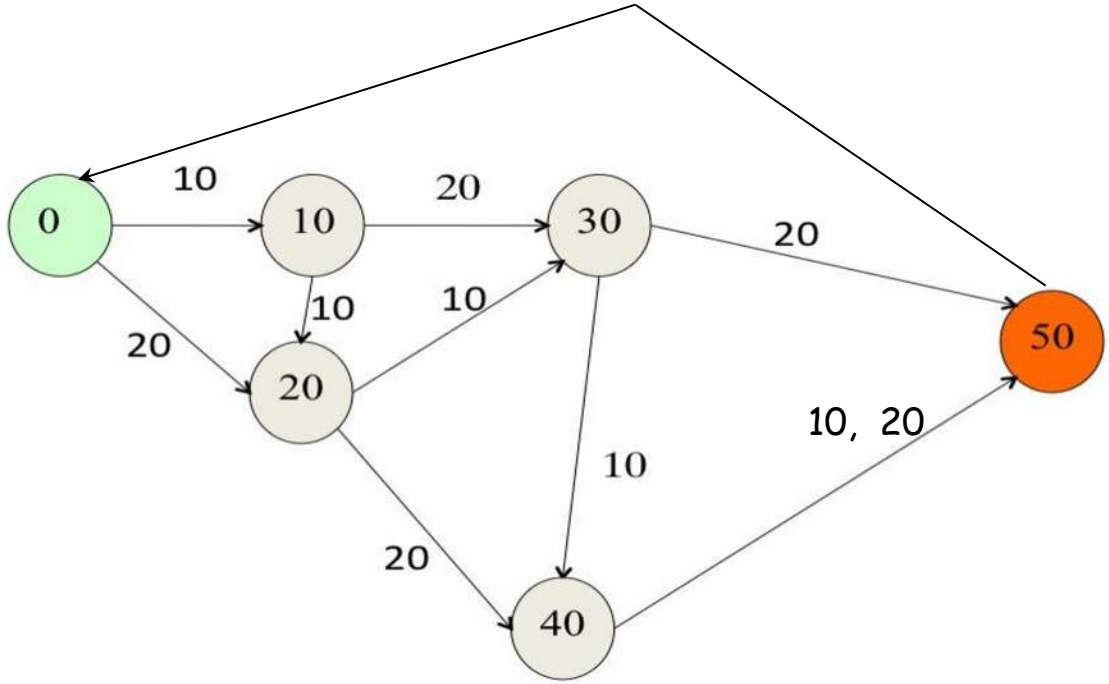
**Computabilità:** *Quali problemi possono essere computati?*
*(con qualsiasi macchina, linguaggio, … )*

**Macchine a stati finiti/Automi:**

*Quali dei problemi problemi possiamo risolvere con* ***memoria costante***


## Distributore di bibite/snack a 50c

Accetta solo monete da 10c e da 20c
non da resto, rifiuta una moneta troppo grande

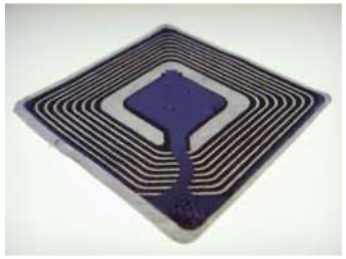



E' una (semplice) **macchina** che opera in accordo all'input (monete)

## Macchina a stati finiti o Automa finito

Astrazione ⇔ modello di calcolo più semplice

- Chip



Componente elettronico su cui è presente un circuito integrato, cioè un circuito elettronico miniaturizzato
Parte di molti apparecchi elettromeccanici

**Computabilità:** *Quali problemi possono essere computati?*
*(con qualsiasi macchina, linguaggio, … e senza limiti sulla memoria)*

Esempi di problemi computazionali

- Problemi numerici
  - Data una stringa binaria, il numero di 1 è maggiore del numero di 0?
  - Dati due numeri x e y, calcola x+y
  - Dato un intero, risulta x primo?
- Problemi riguardanti programmi (es. in C)
  - Data una sequenza di caratteri ASCII, rispetta la sintassi del C?
  - Dato un programma in C, esiste un input che lo manda in loop?

**Computabilità:** *Quali problemi possono essere computati?*
*(con qualsiasi macchina, linguaggio, …
e senza limiti sulla memoria)*

*Es. Dato il seguente programma in C, possiamo stabiliure se termina su ogni input?*

```
input n;
i=0;
while (n>0)
{
n = n-i;
}
```

**Computabilità:** *Quali problemi possono essere computati?*
*(con qualsiasi macchina, linguaggio, …
e senza limiti sulla memoria)*

*Es. Dato il seguente programma in C, possiamo stabiliure se termina su ogni input?*

```
input n;
while (n !=1)
{
if (n is even)
n = n/2;
else
n = 3*n+1;
}
```

*Per n=17: 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.*

# Computabilità:

*Quali problemi possono essere computati?*

*(con qualsiasi macchina, linguaggio, … e senza limiti sulla memoria)*

*Es. Dato il seguente programma in C, possiamo stabiliure se termina su ogni input?*

```
input n;
while (n !=1)
{
if (n is even)
n = n/2;
else
n = 3*n+1;
}
```

*Per n=17: 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.*

***Termina per ogni n>1?***

**Computabilità:** *Quali problemi possono essere computati?*

***Non tutti!!!***

Esempi di problemi computazionali

- Problemi numerici
  - Data una stringa binaria, il numero di 1 è maggiore del numero di 0?
  - Dati due numeri x e y, calcola x+y
  - Dato un intero, risulta x primo?
- Problemi riguardanti programmi (es. in C)
  - Data una sequenza di caratteri ASCII, rispetta la sintassi del C?
  - Dato un programma in C, esiste un input che lo manda in loop?

**Computabilità:** *Vogliamo un modello di computazione indipendente dalla tecnologia presente*

## Macchine di Turing

Ideate da Alan Turing nel 1936.

Modello di calcolatore più semplice:

- macchina a stati finiti, ma NON con memoria finita

- **Nastro (lettura e scrittura) ⇔ memoria, processore**

**Computabilità:** *Vogliamo un modello di computazione indipendente dalla tecnologia presente*

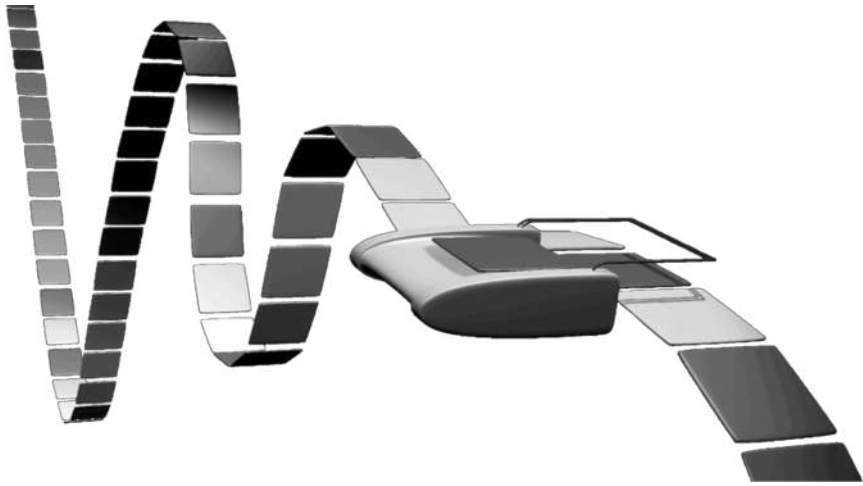
## Macchine di Turing

Ideate da Alan Turing nel 1936.

Modello di calcolatore più semplice:

- macchina a stati finiti, ma NON con memoria finita
- **Nastro (lettura e scrittura) ⇔ memoria, processore**

*Vedremo che una macchina di Turing può fare tutto ciò che può fare un computer reale.*

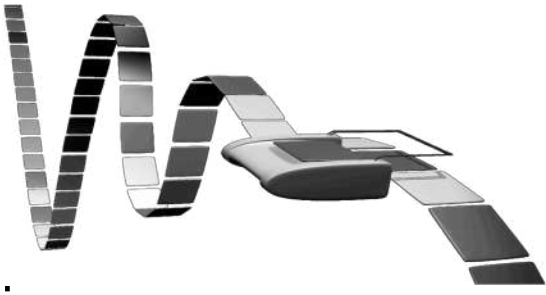


- **Macchine di Turing:**

  Concetto di Computabilità indipendente dalla tecnologia

- **Tesi di Church-Turing:**

  Equivalenza tra programmi  e Macchine di Turing



- **Macchine di Turing:**

  Concetto di Computabilità indipendente dalla tecnologia

- **Tesi di Church-Turing:**

  Equivalenza tra programmi  e Macchine di Turing

- **Limiti delle macchine di Turing (e della computazione):**

  **Esistenza di  problemi non computabili:**

  problemi che non possono essere risolti da alcun programma (indipendentemente dal linguaggio di programmazione usato e dalla macchina su cui lo si esegue).

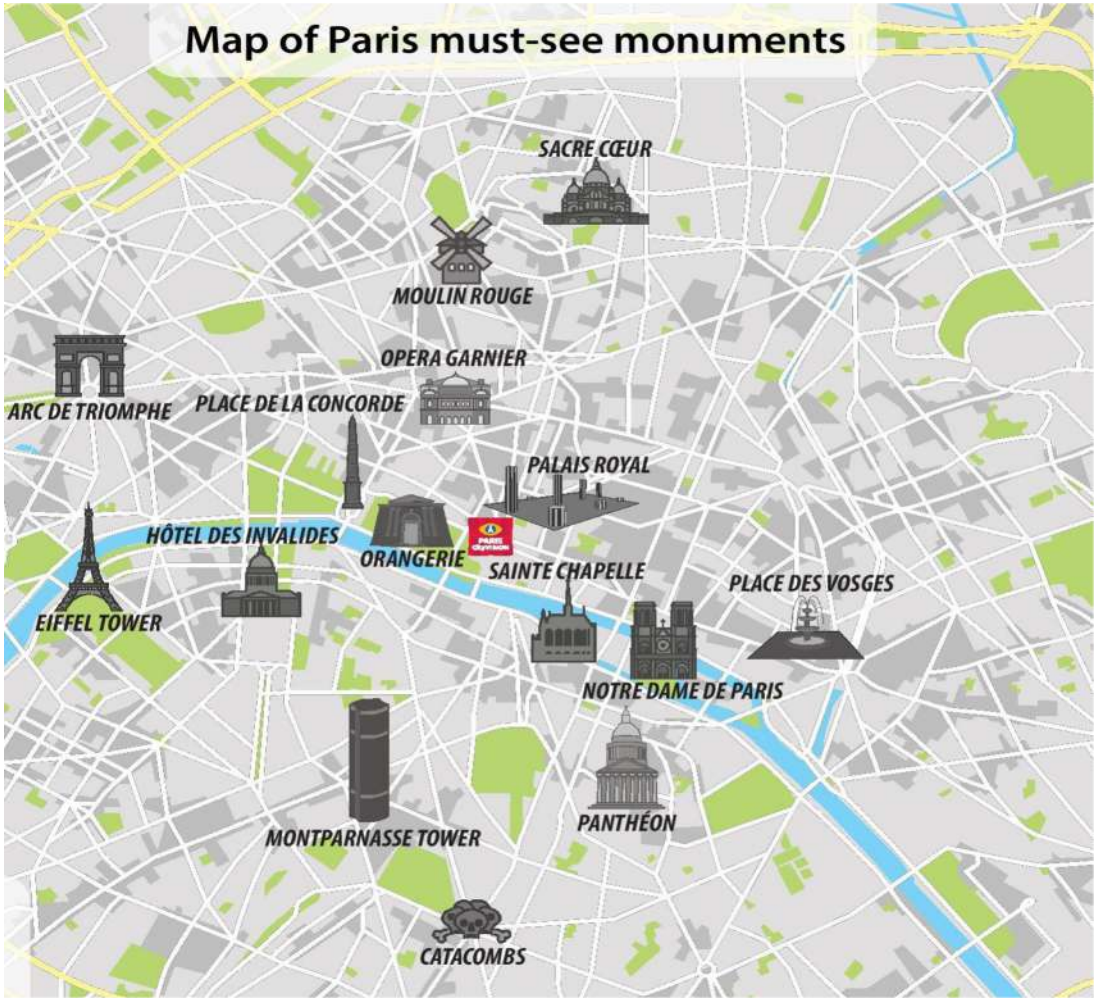
**Computabilità:** *Cosa può essere computato?*
*Cosa non può esserlo?*
*(con qualsiasi macchina, linguaggio, … )*
*Dove si trova il confine?*

**Complessità:** *Quali sono le le risorse minime necessarie*
*(es. tempo di calcolo e memoria)*
*per la risoluzione di un problema?*

*Lista di luoghi con associato l'interesse a visitare il luogo (es. voto da 0 a 100)*

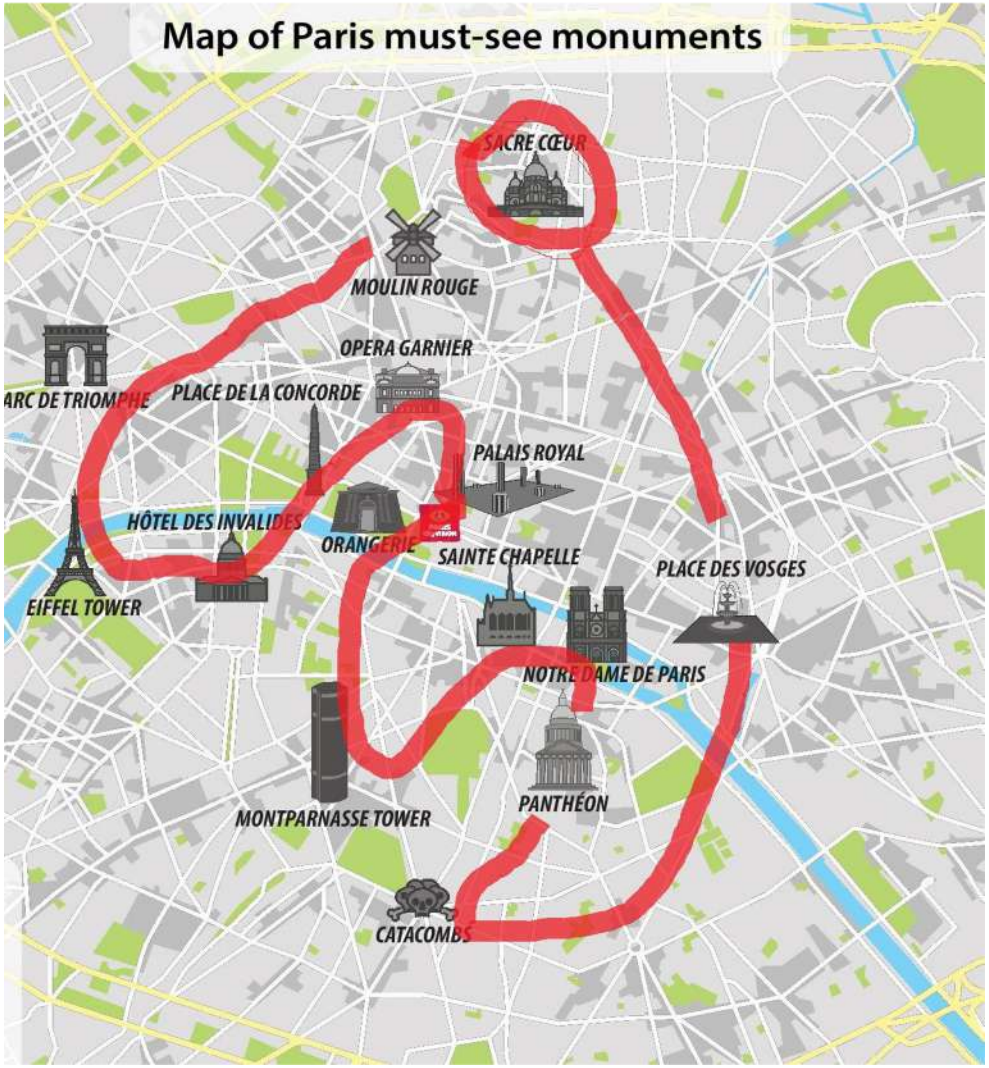
*Vogliamo ordinare i luoghi in ordine di interesse*

*Facile!*



**Vogliamo pianificare un giro turistico che visita tutti i posti di interesse esattamente 1 volta**

*Usando i collegamenti esistenti (metro, bus). Facile?*



# Algoritmo semplice (Backtrack)

# Quanto tempo occorre?

| Siti | Passi |
|---|---|
| N | N! |
| 5 | 120 |
| 15 | 1307674368000 |
| 100 | $\approx 9 \cdot 10^{157}$ |

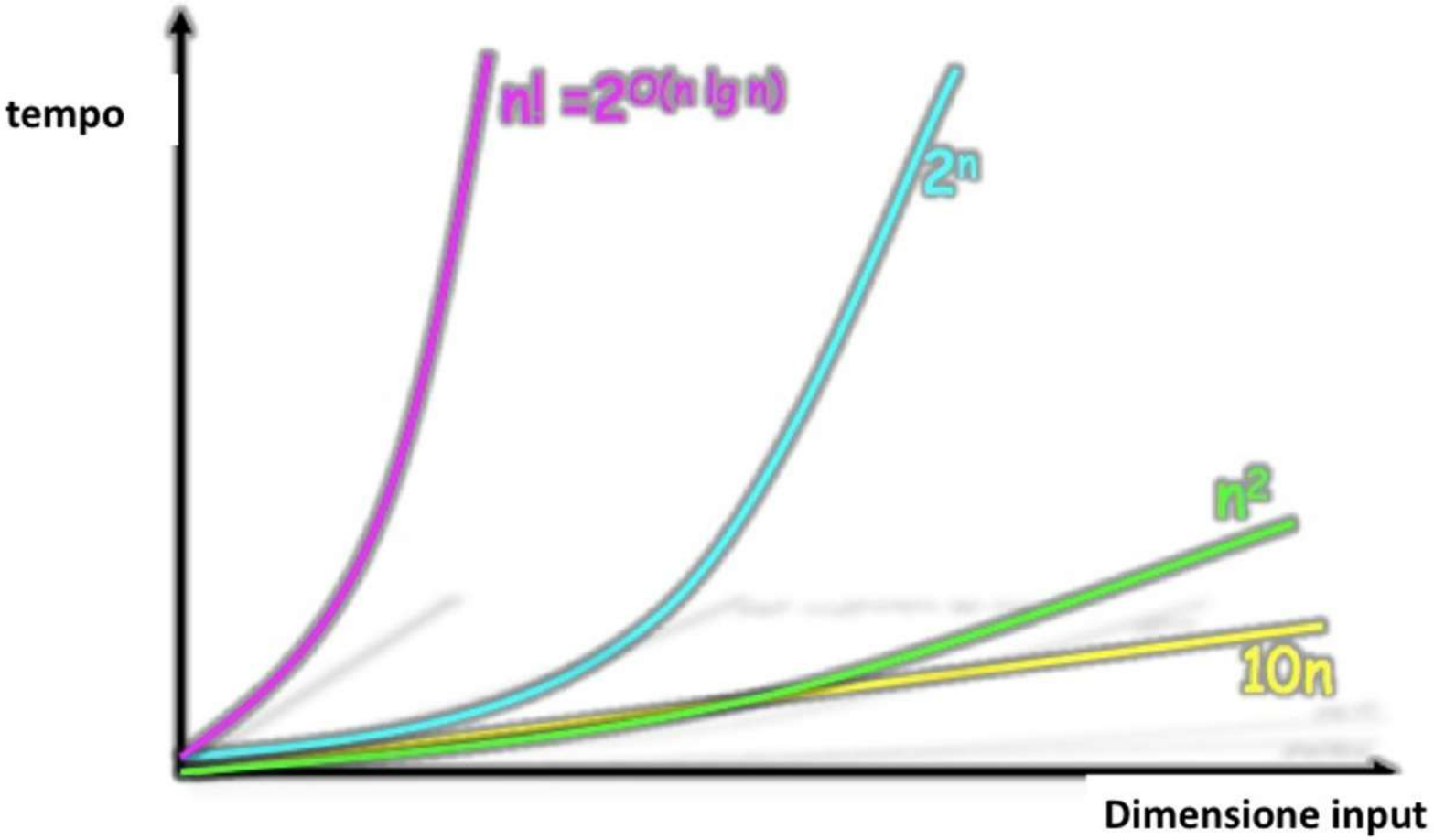
*4 anni su computer che può valutare 10.000 opzioni al secondo*

# Quanto tempo occorre?

| Siti | Passi |
|---|---|
| N | N! |
| 5 | 120 |
| 15 | 1307674368000 |
| 100 | $\approx 9 \cdot 10^{157}$ !!!! |

Quasi 6 mesi su calcolatore 100 volte più veloce (1.000.000 di operazioni al sec.)

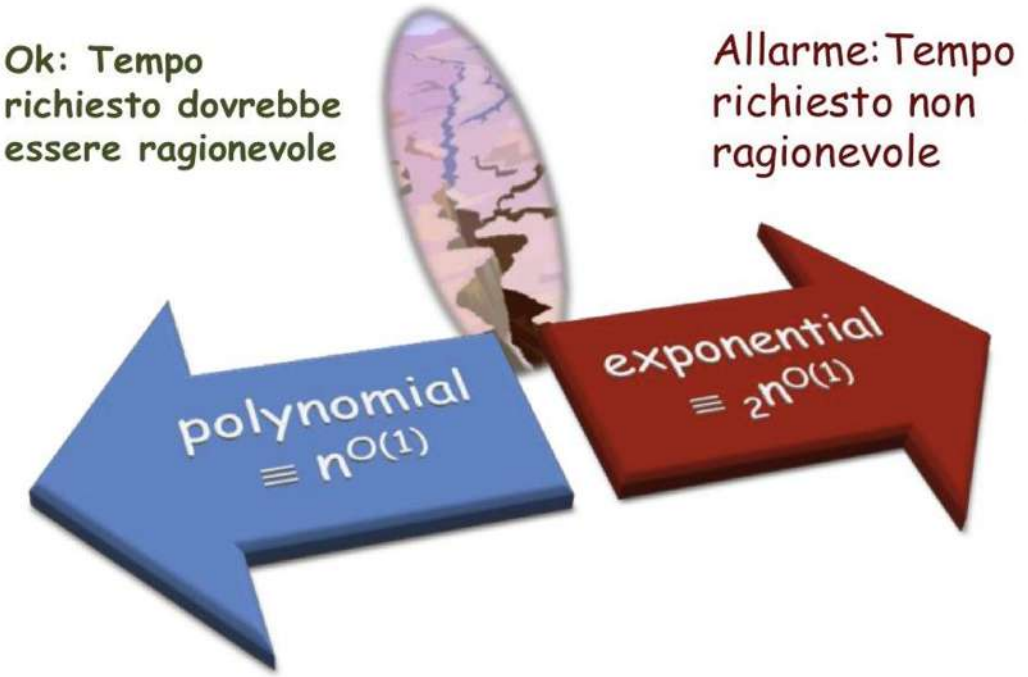
# Tasso di crescita: una prima classificazione



# Algoritmi utili in pratica

**Algoritmi efficienti**: Utilizzano un tempo **polinomiale** su **tutti** gli input

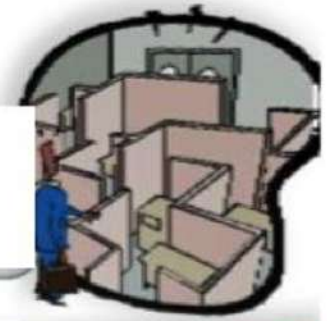
**Algoritmi inefficienti**: Utilizzano un tempo **esponenziale** su **alcuni** input




Nota. Definizione indipendente da sviluppo tecnologico

# Il problema è trattabile o meno?


SI, ecco un algoritmo che lo risolve!




No, ed è anche possibile dimostrarlo



Nessuna delle due.

# Esempi del terzo tipo

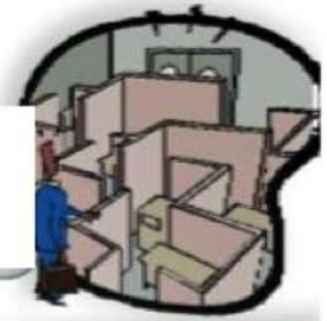
- **Problema del commesso viaggiatore**

- **Programmazione lineare intera**: chiede se esiste una soluzione intera di un sistema di equazioni del tipo

$$
\begin{aligned}
-x + y &\leq 1 \\
3x + 2y &\leq 12 \\
2x + 3y &\leq 12 \\
x, y &\geq 0 \\
x, y &\in \mathbb{Z}
\end{aligned}
$$

# Il problema è trattabile o meno?


SI, ecco un algoritmo che lo risolve!




No, ed è anche possibile dimostrarlo



Nessuna delle due.

Cosa rende un problema "facile" o meno?

# La classe P

**Definizione (informale) della classe P:**

insieme di problemi risolubili in tempo polinomiale da una macchina di Turing

Tesi Church-Turing

insieme di problemi che ammettono un'algoritmo efficiente

# La classe NP

**Definizione (*informale*) della classe NP:**

insieme di problemi per cui

anche se non si conosce un algoritmo efficiente,

si conosce un' algoritmo efficiente di *verifica di una soluzione fornita*

# Problemi NP-completi

- *Travelling Salesman Problem,*
- *3-coloring di grafi,*
- *Scheduling Multiprocessore,*
- *Folding Proteine,*
- *Programmazione lineare intera:*

  *esiste soluzione intera per un sistema del tipo*

$$\begin{aligned} x_1 - 4x_2 + x_3 &= 0 \\ x_1 + x_2 + x_3 &\leq 0 \\ x_1 + 7x_3 &\geq 0 \end{aligned}$$

**Tutti risolvibili efficientemente o nessuno!**

## Argomenti di massima:

- Nozioni preliminari
- Automi Finiti
- Macchine di Turing
- Limiti delle macchine di Turing
- La tesi di Church-Turing
- Le classi P e NP

## *Suggerimenti*

## *(per superare facilmente l'esame)*

- **Seguire il corso**

  È più difficile imparare da soli dal libro di testo (ancora di più dalle slide!)

- Studiare lezione per lezione

  I concetti del corso richiedono un pò di tempo per essere assorbiti,

  non rimanete indietro!

- **Studiare dal libro di testo**
- **Fare gli esercizi**

  Provate sempre a pensare come risolvere un problema

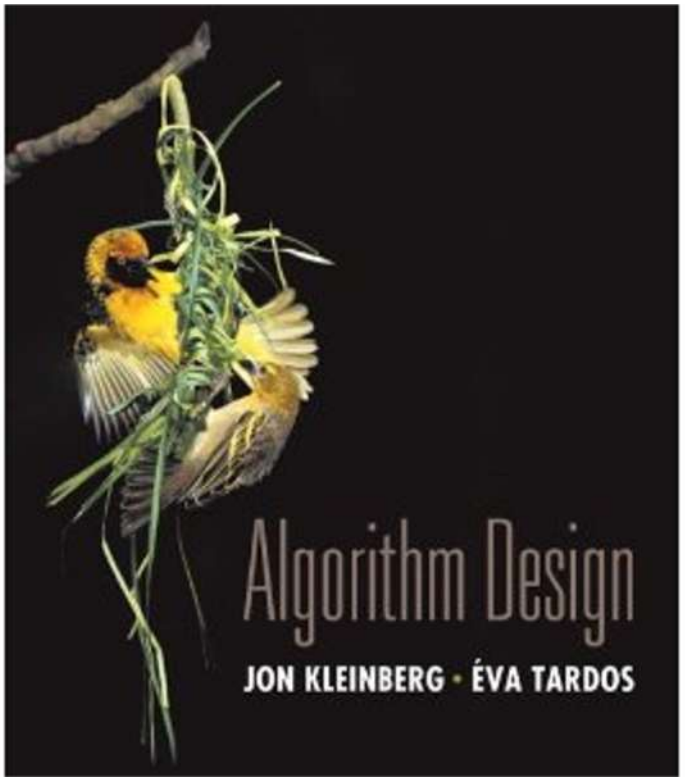
  prima di vedere la risposta

# Testo
# Automi e Computabilità

Michael Sipser, **INTRODUZIONE ALLA TEORIA DELLA COMPUTAZIONE**, MAGGIOLI, 2016

# Testo
# Complessità

- Jon Kleinberg, Eva Tardos, **Algorithm Design,** Pearson (solo **Capitolo 8**)



## *Prove di Esame*

- **Prova scritta con esercizi e teoria**

  **(nessun materiale ammesso)**

- **Eventuale prova orale**

- **Requisito minimo: 48% del totale**

## Prove in Itinere

- 2 prove (Aprile e Preappello)
- stesse modalità delle prove d'esame