Introduzione agli Algoritmi e al Coding

Scuola di Pensiero Computazionale per docenti STEM

Docente: LINDA PAGLI, linda.pagli@unipi.it
Dipartimento Informatica, Università di Pisa

Cos’è un algoritmo?

Un algoritmo è una sequenza finita di istruzioni ben definite per risolvere un problema di natura arbitraria

“istruzioni ben definite ” significa non ambigue e comprensibili dall’esecutore dell’alg., sia esso umano o una macchina.

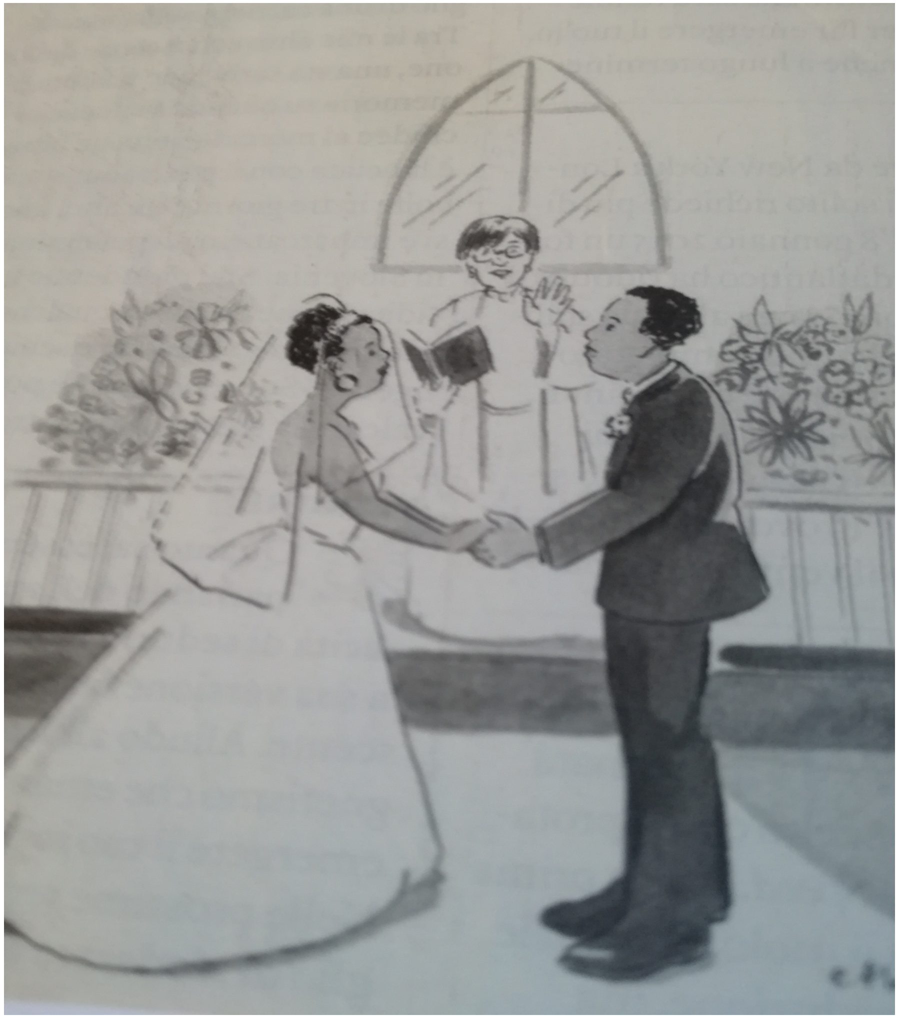
Perché si chiama algoritmo?

Matematico Uzbeko

Abu Ja'far Muhammad ibn Musa al-Khwaritzmi (VIII secolo d. C.)

Scrisse un trattato di algebra (al-jabr in arabo) dove veniva descritto sistematicamente come risolvere le equazioni lineari e quadratiche, cioè gli algoritmi di risoluzione delle equazioni.

Ora, dal New Yorker



…..if someone suspects that the algorithm that put them together doesn't work well, speak now….

Primo esempio di algoritmo

Papiro di Ahmes 1650 B.C.

Algoritmo per la moltiplicazione

```
Molt_Egizia(A, B):                A e B numeri binari, A≥B
P = 0;
    while (A != 0)
        if (A è dispari)    P = P + B;
        A = A/2;
        B =  B*2;
    end
return P;                              alla fine P = AXB.
```

Algoritmo per la moltiplicazione

```
Molt_Egizia (A, B):
P = 0;
    while (A != 0)
        if (A è dispari)    P = P + B;
        A = A/2;
        B =  B*2;
    end
return P;
```

A and B numeri binari, A≥B of n bit

alla fine P = AXB.

A	B	P	
25	17	0	inizio
25	17	17	
12	34	17	
6	68	17	
3	136	17+136=153	
1	272	153+272= 425	
0	544	425	

Algoritmo per la moltiplicazione

A e B numberi binari, A≥B of n bit P = 0;
alla fine P = AXB.

```
Molt_Egizia (A, B):
P=0;
    while (A != 0)
        if (A è dispari)    P = P + B;
        A = A/2;
        B =  B*2;
    end
return P
```

25X
17 =
175 +
25____
425

A	B	P	
25	17	0	inizio
25	17	17	
12	34	17	
6	68	17	
3	136	17+136=153	
1	272	153+272= 425	
0	544	425	

Contiamo il numero di operazioni

Funzione di n = numero di cifre di A e B

Quante volte viene eseguito il ciclo while ?

A ogni iterazione A perde un cifra (divisione intera per 2), quindi diventa 0, dopo n iterazioni

Ad ogni iterazione inoltre:

- una somma
- una divisione intera per 2
- una moltiplicazione per 2.

Contiamo il numero di operazioni

Funzione di n = numero di cifre di A e B
Quante volte viene eseguito il ciclo while ?
A ogni iterazione A perde un cifra (divisione intera per 2), quindi diventa 0, dopo n iterazioni

Ad ogni iterazione inoltre:

- una somma	$O(n)$ operazioni
- una divisione intera per 2	1 operazione
- una moltiplicazione per 2.	1 operazione

In totale: $O(n^2)$ operazioni cioè tempo $O(n^2)$

Perché l'algoritmo funziona?

Multiplicazione Egizia esegue:

- Somme
- Moltiplicazioni per 2
- Divisioni intere per 2

Correttezza:

Idea:

$$A \times B = \begin{cases} A/2 \times 2B & \text{se } A \text{ è pari} \\ A/2 \times 2B + B & \text{se } A \text{ è dispari} \end{cases}$$

Se A è dispari, un unità di A viene persa e quindi al risultato va sommato B una volta in più. Il procedimento si ripete ricorsivamente

Qual è la squadra campione in un torneo di calcio?

- Qual’è l’algoritmo migliore?

Qual è la squadra campione in un torneo di calcio?

Campionato:

T1: ogni squadra incontra tutte le altre una volta in casa e una volta fuori casa.

Campionati mondiali torneo di eliminazione

T2: ogni squadra incontra tutte le altre una sola volta in campo neutrale.

Qual è la squadra campione in un torneo di calcio?

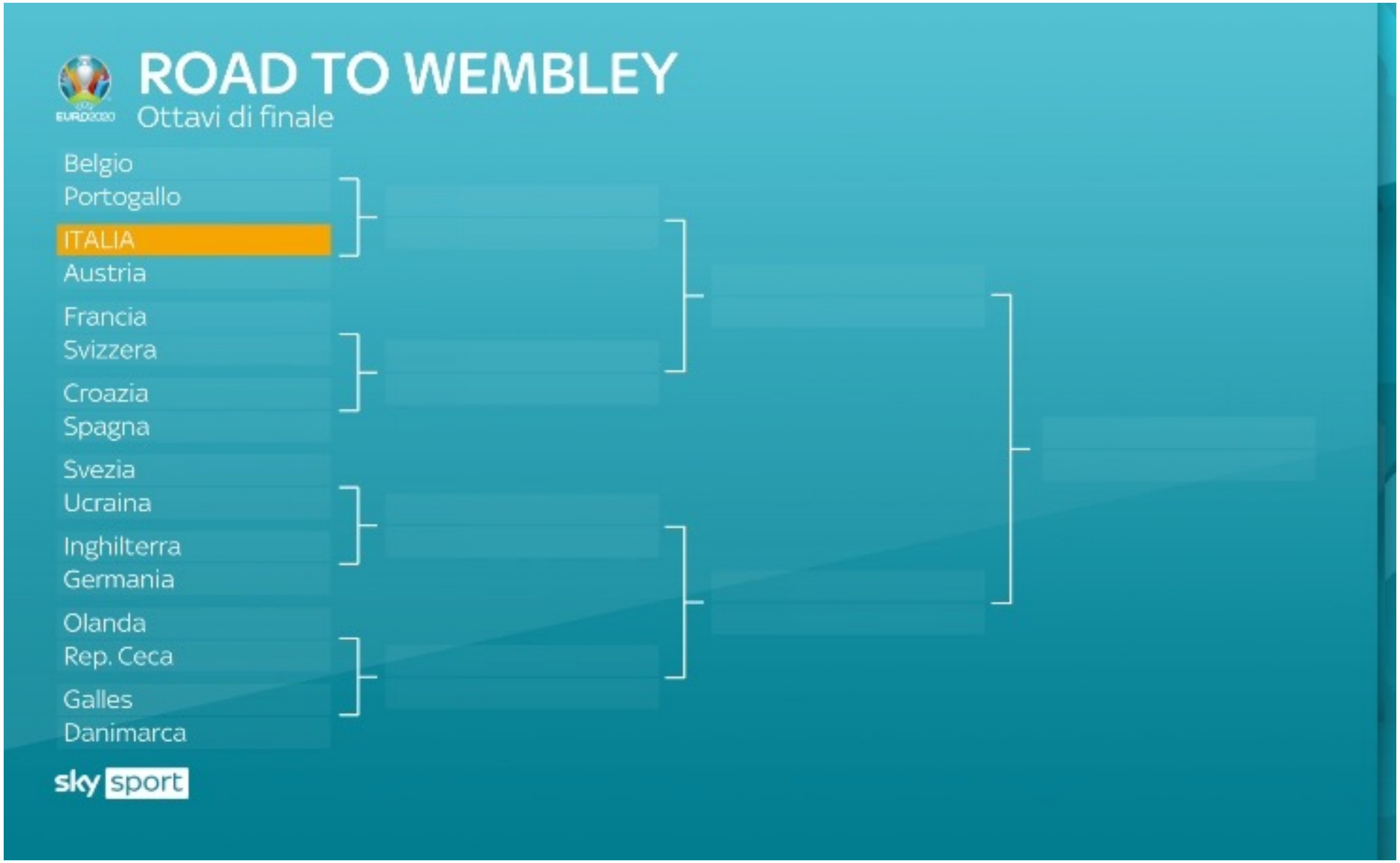
Eliminazione diretta

T3: coppie di squadre s'incontrano. La vincitrice di una partita incontra la vincitrice di un'altra partita, la perdente è eliminata.

Eliminazione diretta in sequenza

T4: Partendo dalla prima partita, la vincitrice incontra la squadra seguente, la perdente è eliminata.

Qual è la squadra campione in un torneo di calcio? T3



Qual è la squadra campione in un torneo di calcio?

Tornei T3 e T4

- Ipotesi

  Per T3 and T4 vale la proprietà transitiva

Contiamo il numero di partite con un torneo di 8 squadre

- T1: 56 (ogni squadra incontra tutte le restanti 8x7) partite.
- T2: 28 partite
- T3: 7 partite (in ogni partita una squadra diversa è eliminata)
- T4: 7 partite

Qual è la squadra campione in un torneo di calcio?

- Qualè il torneo migliore?
- Migliore rispetto a che cosa?

T3 determina la squadra campione e la seconda classificata

T3 minimizza il numero di partire e, se si possono disputare partite contemporanee, minimizza anche il tempo.

T1 massimizza il numero di partite e anche il tempo (campionato: girone di andata e ritorno)

Qual è la squadra campione in un torneo di calcio?

- Qualè il torneo migliore?
- Migliore rispetto a che cosa?

T3 determina la squadra campione e la seconda classificata.

In T3 è dichiarata seconda la perdente della partita finale. E` giusto? Supponiamo che la vera seconda in bravura incontri subito la squadra campione…

I tornei T1 e T2 ci danno la classifica completa dalla squadra campione fino all'ultima.

Trovare la squadra campione è come determinare il massimo elemento di un insieme

- Input: {15, 3, 18, 7, 9, 21, 12, 4}   n=8
- Output: 21

Algoritmo (corrisponde a T4) : confronta ogni elemento col massimo corrente (max).

All'inizio: max=15.

Partita di calcio= confronto tra interi.

Julia code: algoritmo che determina il massimo

- function findmax(s)
-     max = s[1]
-         for i = 2:length(s)
-         if s[i]>max
-             max =s[i]
-         end
-     end
-     return max
- end

- # Inizialize a sequence of integer numbers.
- s = [15, 3, 18, 7, 9, 21, 12, 4] current element s[i], 1 ≤i ≤ length(s) = 8
- # If you want to print the maximum
- println(”the maximum is: ",findmax(s))

Trova il massimo in un insieme di n elementi

Algoritmo: Come nel torneoT4 il massimo corrente sopravvive finché non è battuto.

Operazione: **confronto** tra elementi

Analisi: Quanti confronti per n elementi?

Risposta: **$n-1$.** **$O(n)$, $\Theta(n)$**

Trova il massimo in un insieme di n elementi

Il numero di **confronti** è **n-1.**

Il problema può essere risolto con un numero minore di confronti?

Trova il massimo in un insieme di n elementi

No!!

Occorrono n-1 confronti per trovare il massimo el. (o la squadra campione).

n-1 è un limite inferiore (lower bound) al numero di confronti necessari a risolvere il problema

Idea: Per dire che una squadra non è campione, bisogna che perda almeno una volta. I non campioni sono n-1, quindi occorrono almeno n-1 partite.

Findmax è un algoritmo ottimo! (anche T3 e T4)

Concetti importanti

- Problema
- Algoritmo (molti per un solo problema)
- Confronto tra algoritmi
- Valutazione
- Limite inferiore
- Algoritmo ottimo
- Istanza del problema
- Dimensione del problema

Concetti importanti

- Istanza del problema: il problema considerato su input specifici,
- Es: Findmax, n=8,
- Moltiplicazione, A=28 e B=17

- Dimensione dei dati:
  Es: Moltiplicazione, il numero n di cifre di A and B.
  Trova il max: Il numero di elementi, n.

Complessità

Vogliamo valutare l'efficienza di un alg. come funzione della dimensione dei dati.

- Complessità di spazio: spazio di memoria addizionale per risolvere il problema.
- Complessità di tempo: proporzionale al numero di operazioni.

Il nostro scopo non è solo progettare algoritmi, ma anche di confrontarli per trovare quello migliore.

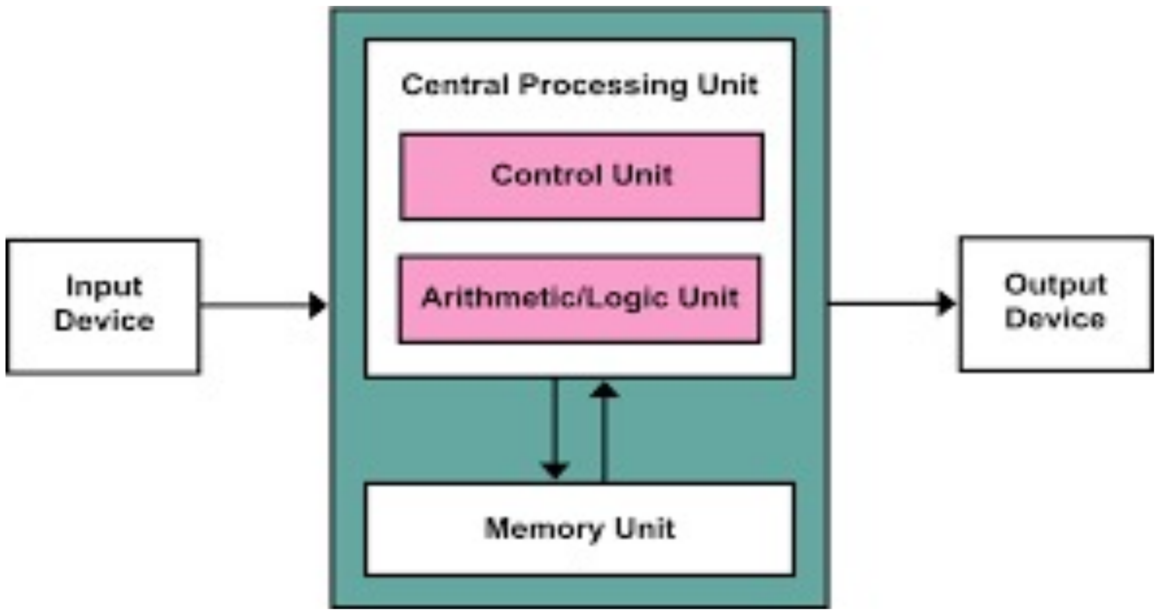
Complessità

- Abbiamo bisogno di un modello di riferimento, non un computer reale ma astratto.

- Si usa il modello RAM (Random Access Machine) chiamato anche

- Il modello di Von Neumann dal nome del suo inventore.

Il modello RAM



Il modello RAM

- Semplificazione del comportamento del computer:

  - Operazioni aritmetiche : +,-,*,/ tempo costante
  - Operazioni di confronto: $<$, $>$, $\leq$, $\geq$, $=$,$\neq$ tempo costante
  - Operazioni logiche : and, or, not, xor, etc. tempo costante
  - Operazioni di controllo: Salti nel programma, tempo costante

La memoria ha dimensione infinita. L'accesso (lettura e scrittura) prende tempo costante.

Il modello RAM

- Stabiliremo la complessità di un alg sulla base del modello RAM, assumendo che tutte le operazioni elementari prendano O(1), cioè tempo costante. (word model).

- Più raro il bit model, dove le operazioni sono funzione della lunghezza delle variabili. Si usa quando le quantità in gioco possono crescere arbitrariamente.

Julia code: algoritmo to determinare il massimo elemento

- function findmax(s)
- max = s[1] tempo costante
- for i = 2:length(s) tempo costante
- if s[i]>max tempo costante
- max =s[i] tempo costante
- end
- end
- return max tempo costante
- end

n-1 volte

- # Inizialize a sequence of integer numbers.
- s = [15, 3, 18, 7, 9, 21, 12, 4]
- # If you want to print the maximum
- println(”the maximum is: ",findmax(s)). O(n)

Il problema dell'ordinamento

Una delle operazioni più frequenti sul web con quella di ricerca (Google)

Ordinare aiuta a cercare.

Tantissimi algoritmi!

Il problema dell’ordinamento

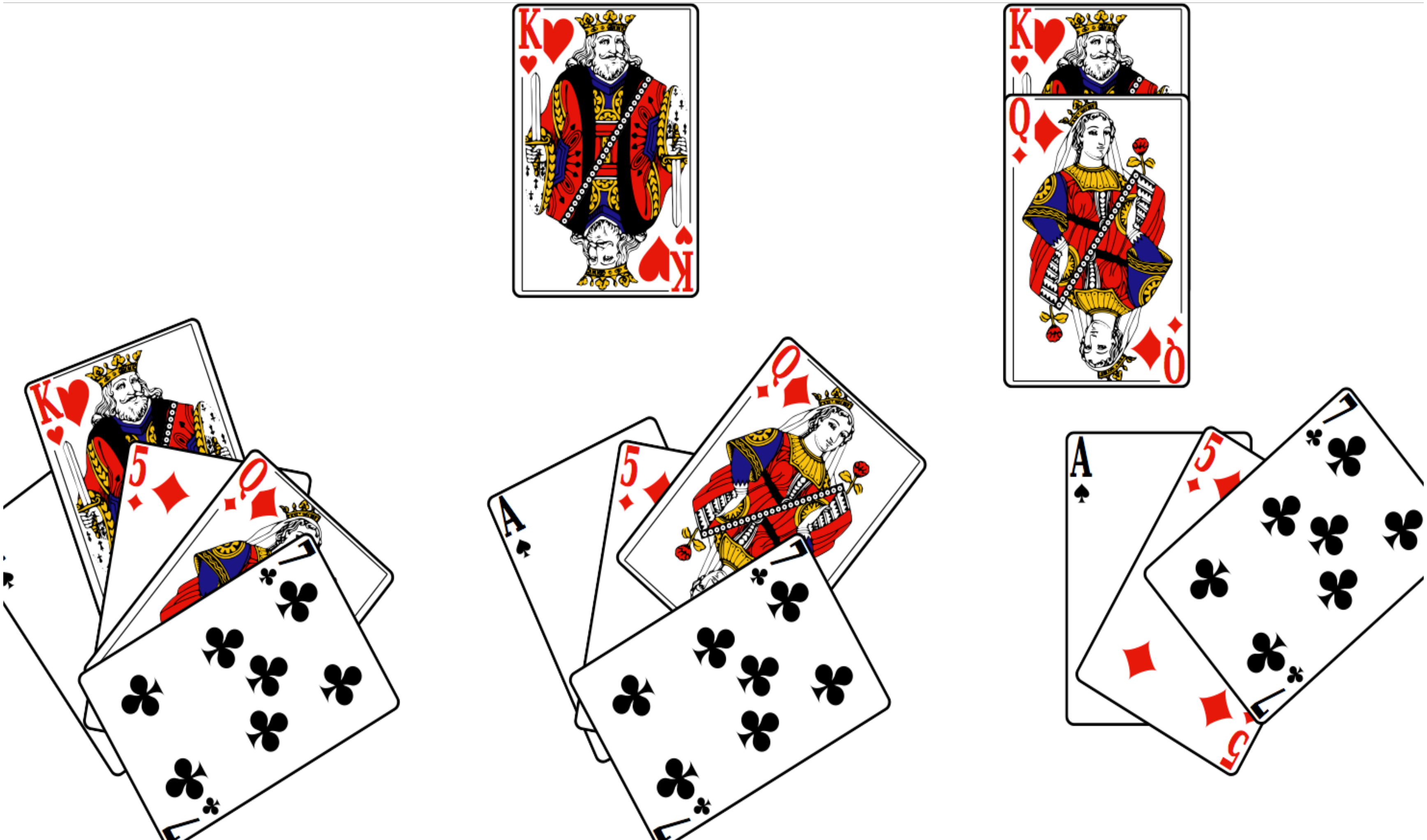
SelectionSort

(esempio con le carte da gioco)

Algoritmo

- Seleziona la carta col valore più alto e mettila in fondo al mazzo;
- Continua così finché ci sono carte.

Selection Sort



Il problema dell'ordinamento

Selection Sort utilizza la function **findmax:**

Alg: Ad ogni passo i, $0 \leq i < n-2$

considera i primi n-i elementi

findmax

metti max in ultima posizione

Step 0: n carte, n-1 confronti

Step 1: n-1 carte, n-2 confronti

....

Step n-2: 2 carte, 1 confronti

Totale: $(n-1)+ (n-2) + \ldots + 2+ 1 = (n-1)/2 \cdot n$

$= n(n-1)/2 = n^2/2-n/2$, that is $\Theta(n^2)$ tempo

Il problema dell'ordinamento

n =40 : 780 operazioni, **molto veloce**

Se dobbiamo ordinare un milione di pagine web, il numero di passi di SelectionSort $\approx 10^{12}$, che significa 2 minuti su di un computer super veloce:

Troppo!

Esistono algoritmi migliori:

MergeSort QuickSort $\Theta(n\log n)$

Notazioni Asintotiche

Utili a esprimere la complessità di tempo o di spazio di un algoritmo al crescere della dimensione del problema.

The complessità è espressa in genere non in maniera esatta, ma in ordine di grandezza con n che va all'infinito.

Notazioni:	O	big O	limiti superiori
	Θ	Theta	limiti esatti
	Ω	Omega	limiti inferiori

Donald Knuth introdusse queste notazioni in un articolo degli anni '70.

Notazioni Asintotiche

A algoritmo
$f(n)$ complessità di tempo di *A*

$f(n) = \frac{1}{2}n^2 - 3n$ $\qquad$ $f(n) = \Theta(n^2)$

Si tralasciano le costanti moltiplicative e i termini di grado inferiore a quello di grado più alto.
$f(n)$ è anche $O(n^2)$ e $\Omega(n^2)$

$f(n) = 3n^2\log n + 2n^4$ $\qquad$ $f(n) = \Theta(n^4)$
$f(n) = 3n^3\log n^2 + 4n\log^3 n$ $\qquad$ $f(n) = \Theta(n^3\log n^2)$
$f(n) = 4n^{10} + 5n^2 + 6n - 2$ $\qquad$ $f(n) = \Theta(n^{10})$

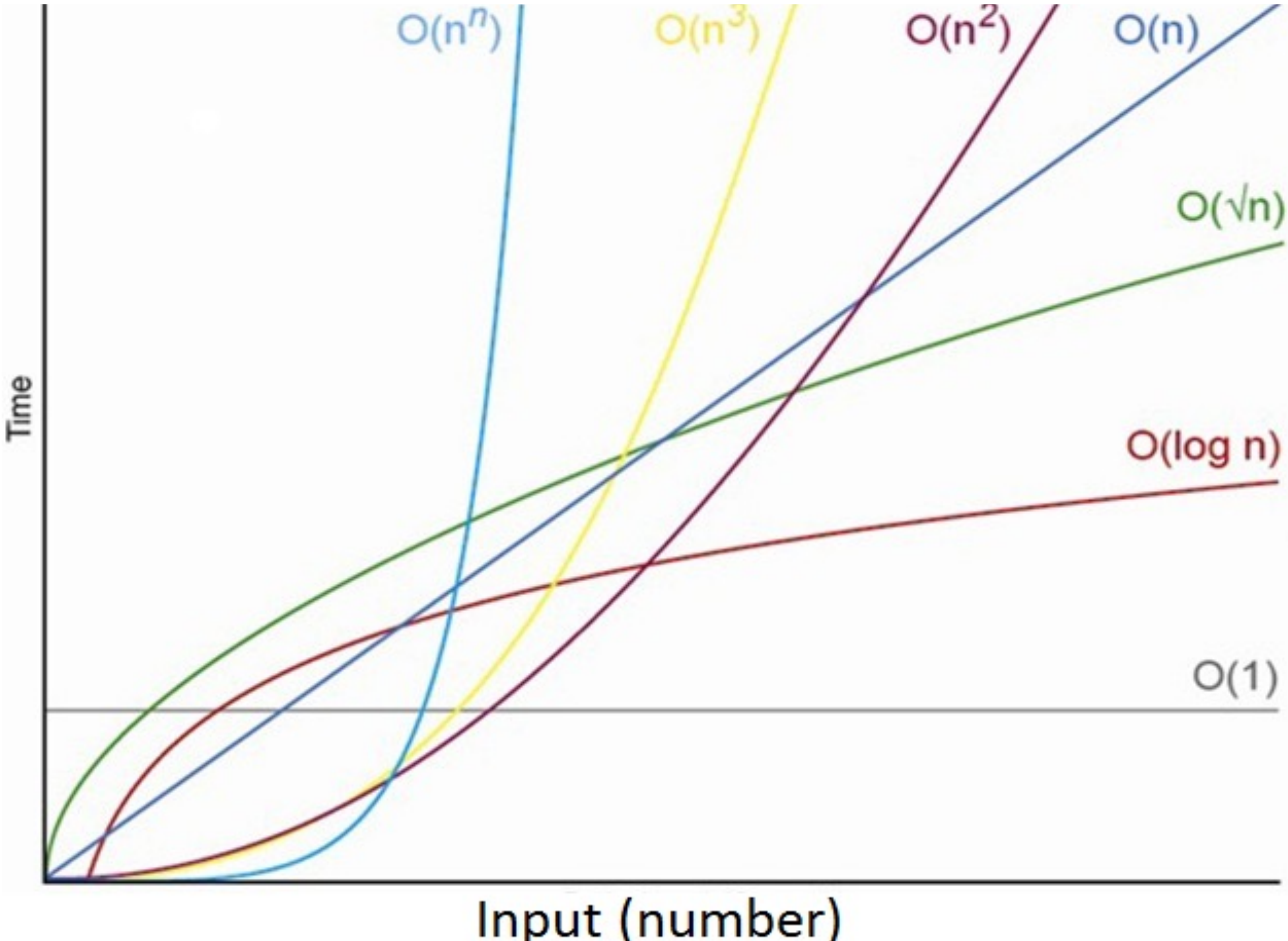
Notazioni Asintotiche

P problema
A un algoritmo che risolve P

Upper Bound $O(t(n))$ per P (caso pessimo):
esiste A per P che richiede al più tempo $t(n)$ per ogni instanza di P di dimensione n.
Lower Bound $\Omega(f(n))$ per P (caso pessimo):
ogni A per P di dimensione n richiede almeno tempo $f(n)$.

If $t(n) = f(n)$ A è ottimo per P

Crescita di funzioni



	n			
	10	**50**	**100**	**1,000**
lg n	0.0003 sec	0.0006 sec	0.0007 sec	0.0010 sec
$n^{1/2}$	0.0003 sec	0.0007 sec	0.0010 sec	0.0032 sec
n	0.0010 sec	0.0050 sec	0.0100 sec	0.1000 sec
n lg n	0.0033 sec	0.0282 sec	0.0664 sec	0.9966 sec
n^2	0.0100 sec	0.2500 sec	1.0000 sec	100.00 sec
n^3	0.1000 sec	12.500 sec	100.00 sec	1.1574 day
n^4	1.0000 sec	10.427 min	2.7778 hrs	3.1710 yrs
n^6	1.6667 min	18.102 day	3.1710 yrs	3171.0 cen
2^n	0.1024 sec	35.702 cen	4×10^{16} cen	1×10^{186} cen
$n!$	362.88 sec	1×10^{51} cen	3×10^{144} cen	1×10^{2554} cen

Table 1: Time required to process n items at a speed of

Algoritmi polinomiali e esponenziali

- $O(n^K)$ o $\Theta(n^K)$ complessità di tempo polinomiale, con k costante, es: Moltiplicazione, SelectionSort, etc.
- $\Theta(k^n)$ o $\Theta(n^n)$ complessità di tempo esponenziale, es: Partizione
- n è la dimensione del problema

Un problema difficile

- Input: un insieme A di n interi positivi.
- Domanda: È possibile suddividere A in 2 sottoinsiemi A' e A'' di ugual somma?

A = {6, 7, 5, 4, 1, 3, 7, 2, 4, 1}
A' = {6, 7, 2, 4, 1} , somma=20
A'' = {5, 4, 1, 3, 7} , somma=20

- Output: Yes

Siamo in grado di risolvere Partizione per alcune istanze del problema solo in tempo esponenziale
Partizione è $O(2^n)$ nel caso pessimo

Ricerca Esaustiva (o forza bruta)

La questione di P e NP

Possiamo risolvere Partizione e tutti i problemi che si sanno risolvere facendo la ricerca esaustiva senza tale ricerca?

Non lo sappiamo

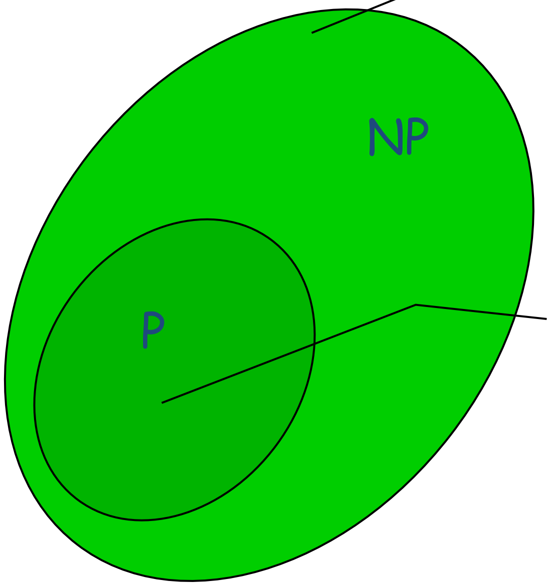
P e NP

- P “tempo Polinomiale”
  Problemi risolubili velocemente

- NP “tempo non deterministico Polinomiale”
  Problemi verificabili velocemente

  include Partizione e tantissimi altri

Le classi P e NP



Problemi facilmente verificabili

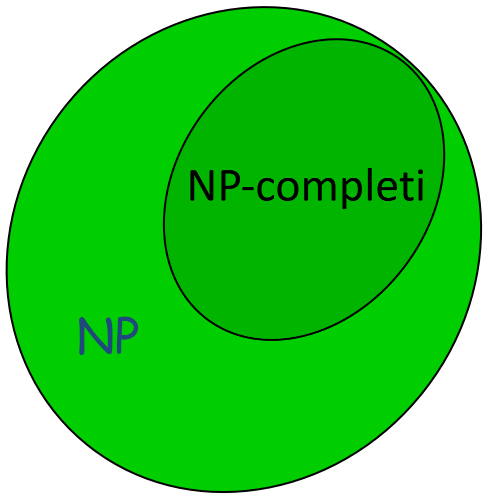
- partizione
- Clique

Problemi facilmente risolvibili

- Moltiplicazione
- Ordinamento

$P = NP$
oppure
$P \neq NP$

Problemi NP-completi



Problemi NP-completi :
Se uno è facile allora sono tutti facili!
Se uno è difficile allora sono tutti difficili!

Se un problema NP-completo è in P allora P = NP

NP-completezza

Se un problema è NP-completo la speranza di trovare un algoritmo efficiente è molto bassa.

Se lo trovassimo non solo avremmo risolto il problema ma anche tutti gli altri problemi NP-completi con le riduzioni.

Tantissimi problemi NP-completi si trovano in matematica, biologia, fisica, economia...