Titolo unità didattica: Efficienza degli algoritmi [10]

Titolo modulo : Complessità asintotica [02-T]

Complessità asintotica: notazioni formali e classificazione degli algoritmi

Argomenti trattati:

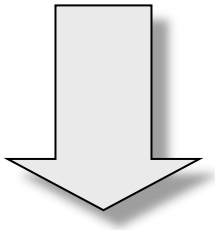
- ✓ complessità asintotica di un algoritmo
- ✓ notazione “O”
- ✓ classificazione degli algoritmi in base alla complessità asintotica

Prerequisiti richiesti: P1-10-01-T

complessità asintotica

caratterizzare l'andamento **asintotico** di $T(n)$ e $S(n)$

comportamento della complessità computazionale al **crescere** della dimensione computazionale n



determinare solo i **termini dominanti** per n che tende a infinito, e non l'espressione analitica completa di $T(n)$ e $S(n)$

siano $f(n)$ e $g(n)$ due funzioni **non negative** e **non decrescenti**;
si dice che

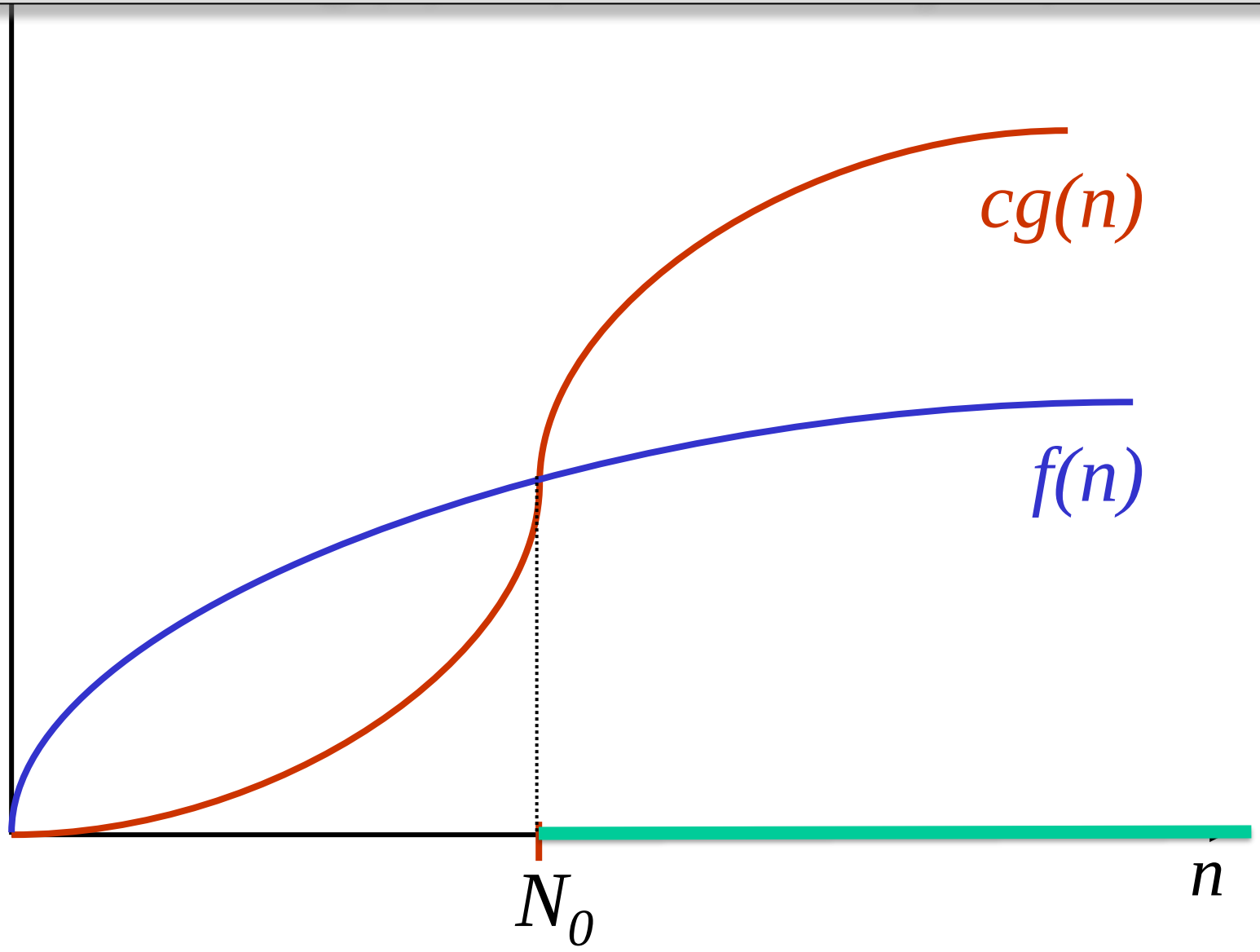
$$f(n) = O(g(n))$$

f è dell'ordine di g

se esistono due **costanti positive** c e N_0
tali che:

$$f(n) \leq cg(n) \qquad n \geq N_0$$

il grafico $f(n)$ sta **sotto** (o coincide con) il grafico di $cg(n)$, a partire da N_0 in poi



$$f(n) = O(g(n))$$

$$f(n) \leq cg(n)$$

$$n \geq N_0$$

notazione *O grande* (big O), notazione di Landau, notazione asintotica

oppure

$$f(n) \in O(g(n))$$

f appartiene all'insieme delle funzioni maggiorate asintoticamente da una opportuna scalatura della funzione g

Esempio: se $f(n) = \sum_{i=0}^{p} a_i n^i$

cioè $f(n)$ è un polinomio di grado p allora:

$$f(n) = O\left(n^{p}\right)$$

Esempio: se $f(n) = 2n^2 + 3n + 5$ allora:

$$f(n) = O\left(n^{2}\right)$$

Esempio: trovare una coppia di valori per c e N_0

$$f(n) = 2n^2 + 3n + 5 \quad \Rightarrow \quad g(n) = n^2$$

infatti, basta considerare, per esempio

$$c = 2 + 1 = 3$$

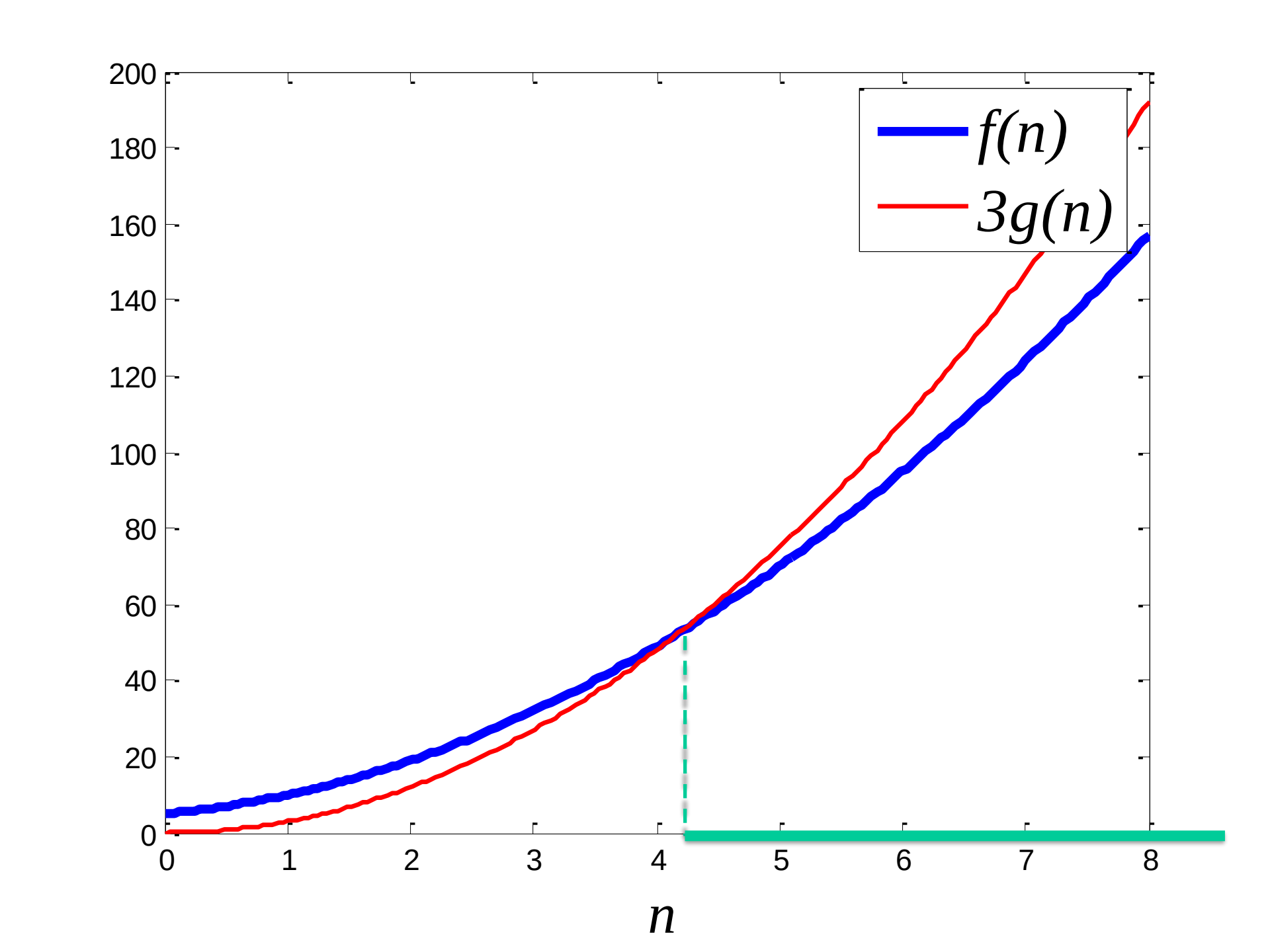
e porre N_0 uguale al più piccolo intero maggiore della più piccola radice positiva dell'equazione

$$2n^2 + 3n + 5 = 3n^2$$

$$n^2 - 3n - 5 = 0$$

soluzioni:
4.1926, -1.1926

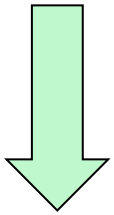
ovvero $N_0 = 5$


200
180
160
140
120
100
80
60
40
20
0
0
1
2
3
4
5
6
7
8
n
f(n)
3g(n)

$$T(n) = O(g(n))$$

$$T(n) \in O(g(n))$$

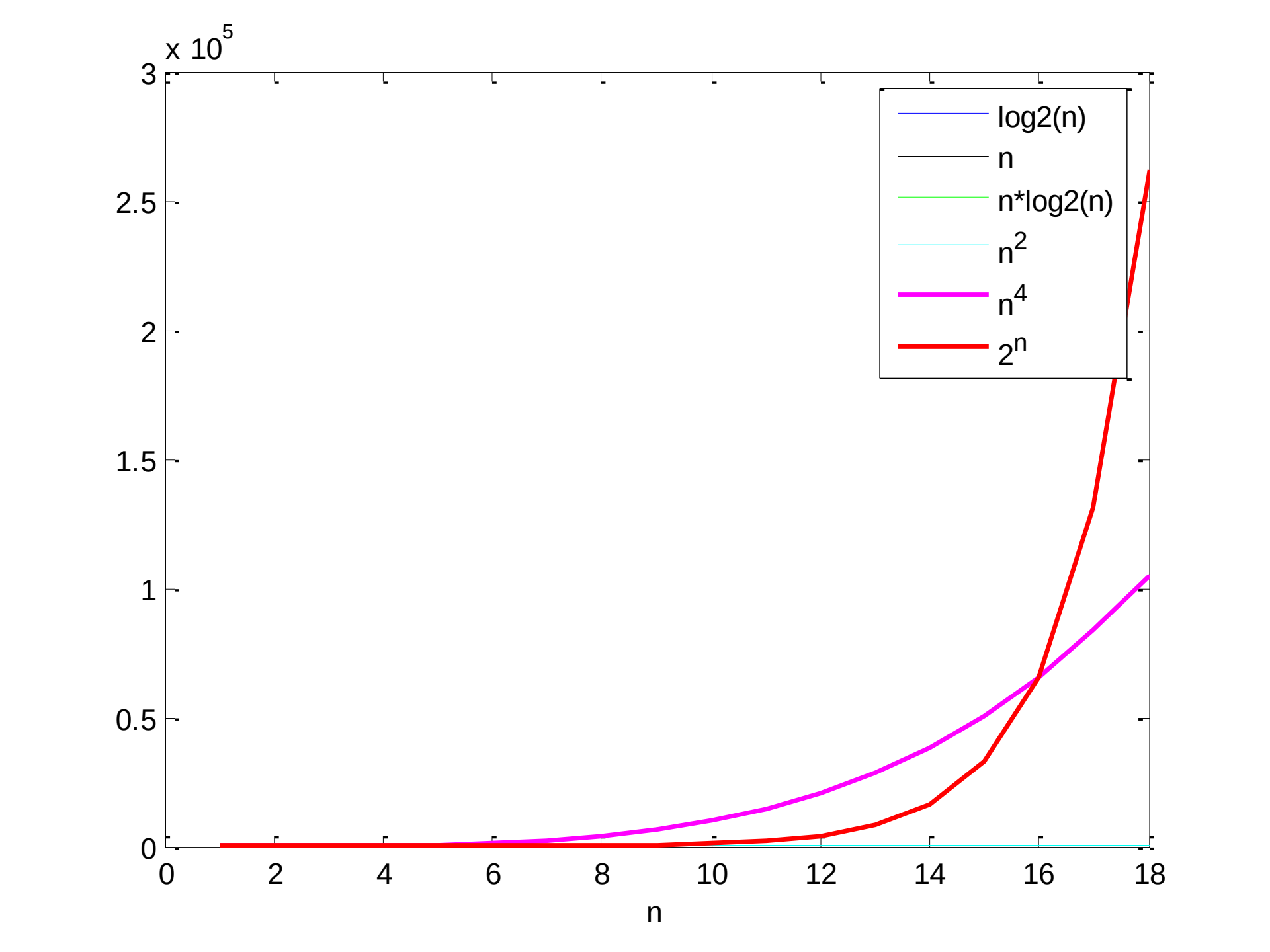
la complessità di tempo dell'algoritmo è minore di (o al più uguale a) $cg(n)$, per n sufficientemente grande

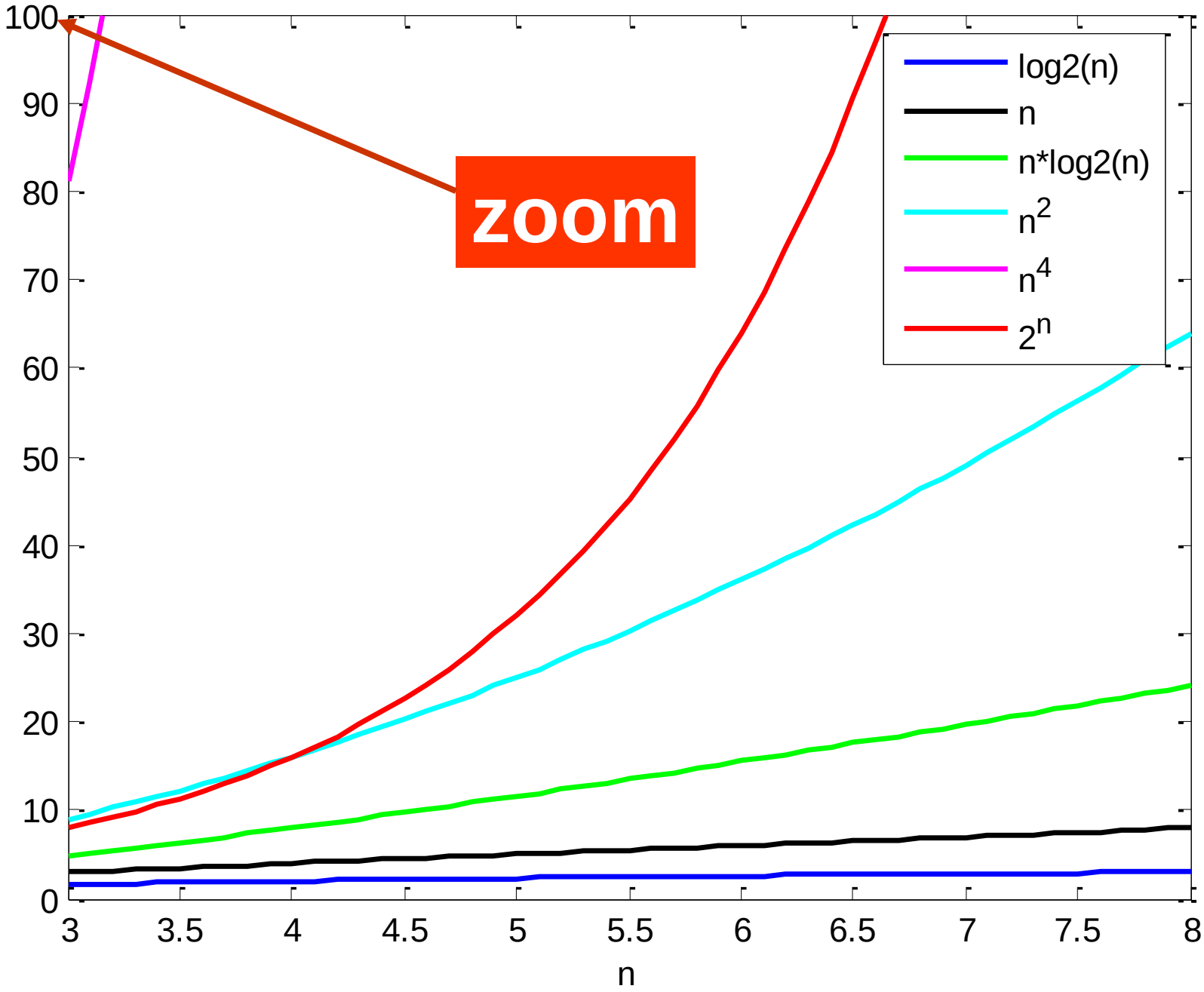


classi di complessità

classi di complessità

T(n) =	$O(1)$	costante
T(n) =	$O(\log_2 n)$	logaritmica
T(n) =	$O(n)$	lineare
T(n) =	$O(n \log_2 n)$	lin-log
T(n) =	$O(n^2)$	quadratica
T(n) =	$O(n^k)$	polinomiale di grado k
T(n) =	$O(2^n)$	esponenziale
T(n) =	$O(n!)$	fattoriale


x 10^5
3
2.5
2
1.5
1
0.5
0
0
2
4
6
8
10
12
14
16
18
n
log2(n)
n
n*log2(n)
n^2
n^4
2^n


100
90
80
70
60
50
40
30
20
10
0
3
3.5
4
4.5
5
5.5
6
6.5
7
7.5
8
n
zoom
log2(n)
n
n*log2(n)
n^2
n^4
2^n

dimensioni dei problemi risolvibili, con un computer di potenza 100 Mops/sec

complessità	1 sec	1 minuto	1 ora
$O(n)$	10^8	$6 \cdot 10^9$	$3.6 \cdot 10^{11}$
$O(n \log_2 n)$	$\approx 4 \cdot 10^6$	$\approx 2 \cdot 10^8$	$\approx 1 \cdot 10^{10}$
$O(n^2)$	10^4	77459	$6 \cdot 10^5$
$O(2^n)$	26	32	38
$O(n!)$	11	12	14

in WolframAlpha: **`solve 10^(-8)*x^2 = 60`**

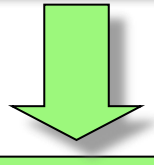
- `10^(-8)`: tempo per 1 operazione
- `x^2`: numero operazioni
- `60`: 1 min

dimensioni dei problemi risolvibili, con un computer di potenza 1000000 Mops/sec

complessità	1 sec	1 minuto	1 ora
$O(n)$	10^{12}	$6 \cdot 10^{13}$	$3.6 \cdot 10^{15}$
$O(n \log_2 n)$	$2.5 \cdot 10^{10}$	$\approx 1.4 \cdot 10^{12}$	$\approx 7.5 \cdot 10^{13}$
$O(n^2)$	10^6	$\approx 7.7 \cdot 10^6$	$6 \cdot 10^7$
$O(2^n)$	35	45	51
$O(n!)$	14	16	17

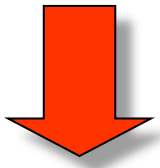
in WolframAlpha: **`solve 10^(-12)*2^x=60`**

algoritmi utilizzabili per la risoluzione effettiva di problemi



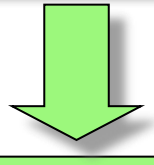
algoritmi a complessità di tempo **polinomiale**

algoritmo a complessità di tempo **fattoriale**: calcolatore che esegue una operazione in 10^{-12} sec, problema di dimensione computazionale 100



tempo di esecuzione: 10^{138} anni

algoritmi utilizzabili per la risoluzione effettiva di problemi



algoritmi a complessità di tempo **polinomiale**

algoritmo a complessità di tempo **fattoriale**: calcolatore che esegue una operazione in 10^{-12} sec, problema di dimensione computazionale 100

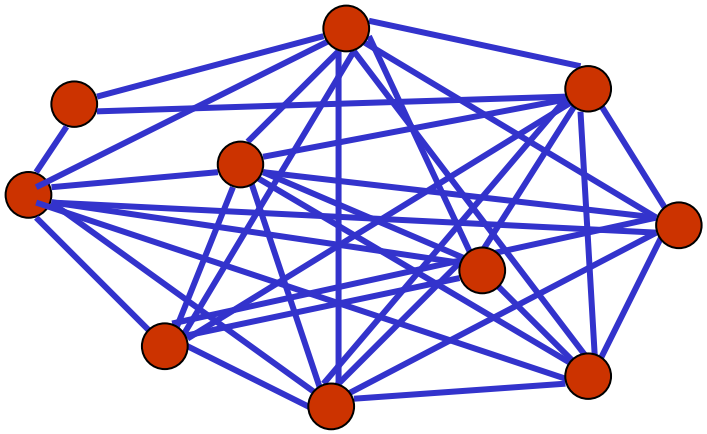
in WolframAlpha:

```
10^(-12)*factorial(100)/(86400*365)
```

tempo di esecuzione: 10^{138} anni

c’è necessità di fare ricorso a un algoritmo di complessità di tempo esponenziale o fattoriale**?**

problema del commesso viaggiatore
(TSP, *travelling salesman problem*)



città

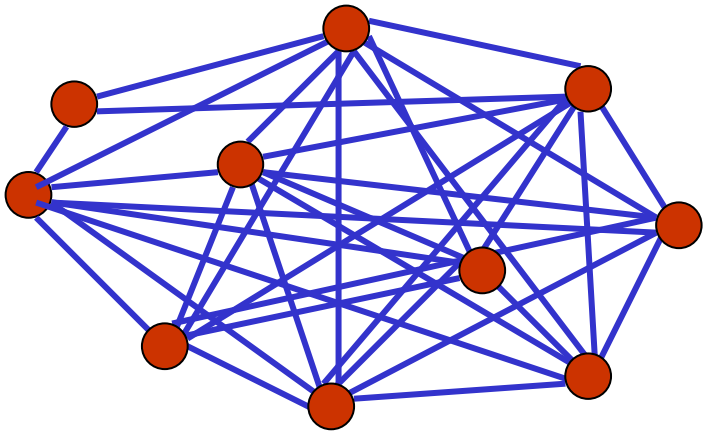
percorso

$$T(n) = O(n!)$$

dato un insieme di città e di costi di viaggio, da una qualunque città a un’altra qualunque città, determinare il **cammino più economico** che permetta di visitare **ogni** città esattamente **una volta sola** e quindi di ritornare alla città di partenza

c’è necessità di fare ricorso a un algoritmo di complessità di tempo esponenziale o fattoriale**?**

problema del commesso viaggiatore
(TSP, *travelling salesman problem*)



città

percorso

$$T(n) = O(n!)$$

l'unico algoritmo conosciuto che risolve questo problema è un algoritmo di forza bruta che calcola tutti i percorsi possibili e determina quello più economico

il numero di percorsi possibili è pari al numero delle permutazioni di n città

problema di dimensione computazionale $n=10^7$

algoritmo a complessità quadratica	algoritmo a complessità lin-log
10000 Mops/sec:	100 Mops/sec
$\frac{(10^7)^2 \text{ operazioni}}{10^{10} \text{ operazioni/sec}}$	$\frac{10^7 \cdot \log_2 10^7}{10^8 \text{ operazioni/sec}}$
$=10^4$ sec	≈ 2 sec

algoritmo = tecnologia