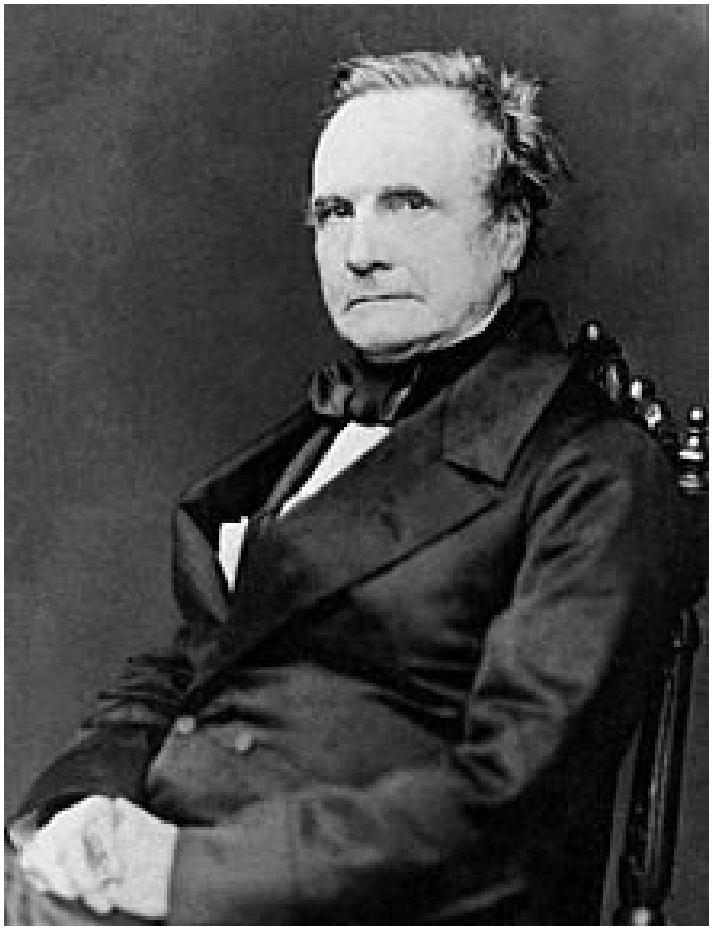
...Quanto costa?

Charles Babbage

“As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time?” (1864)



Donald Knuth

Tempo di esecuzione di un algoritmo = somma dei costi x frequenza di ogni operazione (1974)

Quanto costa?

Supponiamo che l'assegnazione **“=”** costi **c1**, il confronto **“<”** costi **c2**, l'incremento **“++”** costi **c3**, la stampa **“cout<<”** costi **c4** (per semplicità, supponiamo che il costo di una stampa sia costante a prescindere dal numero degli argomenti stampati).

```
int n = 10;
for (int i=0; i<n; ++i)
   cout << i << endl;
```

Quanto costa?

Supponiamo che l'assegnazione **“=”** costi **c1**, il confronto **“<”** costi **c2**, l'incremento **“++”** costi **c3**, la stampa **“cout<<”** costi **c4** e la lettura da stdin **“cin>>”** costi **c5.**

```
int n;
cin >> n;
for (int i=0; i<n; ++i)
   cout << i << endl;
```

Quanto costa?

Supponiamo che l'assegnazione **“=”** costi **c1**, il confronto **“<”** e **“==”** costino **c2**, l'incremento **“++”** costi **c3**, la stampa **“cout<<”** costi **c4** e la lettura da stdin **“cin>>”** costi **c5.**

```
int n;
cin >> n;
if (n==42)
   cout << “Quarantadue!\n”;
else
   for (int i=0; i<n; ++i)
      cout << i << endl;
```

Quanto costa?

Supponiamo che l'assegnazione **“=”** costi **c1**, il confronto **“<”** e **“==”** costino **c2**, l'incremento **“++”** costi **c3**, la stampa **“cout<<”** costi **c4** e la lettura da stdin **“cin>>”** costi **c5.**

```
int n;
cin >> n;
for (int i=0; i<n; ++i)
   for (int j=0; j<n; ++j)
      cout << “(“ << i << “,” << j << “)\n”;
```

Principi guida dell'analisi degli algoritmi

- Non vogliamo prestare attenzione alle costanti moltiplicative e ai termini di ordine inferiore
- Analisi asintotica

Perché l'analisi asintotica

$7n \log_2 n + 7n$ [la complessità di mergesort calcolata in modo preciso, ma su questo ci torneremo!] “è meglio di” $(1/2)n^2$

anche se per n=2 questa affermazione non è vera.

Però...

- Per n = 512, $7n \log_2 n + 7n = 35840$ e $(1/2)n^2 = 131072$
- Per n = 1024, $7n \log_2 n + 7n = 78848$ e $(1/2)n^2 = 524288$

Theta, Omega, Big-O

Notazioni abbastanza grossolane da permetterci di trascurare i fattori costanti moltiplicativi e i termini di ordine inferiore, ma abbastanza raffinate da permettere il confronto di algoritmi diversi

$$7n \log_2 n + 7n \text{ “=” } n \log n$$

Per adesso usiamo Theta, poi formalizzeremo e saremo in grado di capire cosa scegliere tra le tre notazioni di volta in volta.

Dobbiamo determinare il costo dell'algoritmo nel caso migliore e nel caso peggiore; in alcuni casi si fa anche l'analisi nel caso medio

Quanto costa?

```
int n;
cin >> n;
for (int i=0; i<n; ++i)
   for (int j=i; j<n; ++j)
      cout << “(“ << i << “,” << j << “)\n”;
```

Quanto costa?

```
void create_empty(basic_list& list)
{
   cell* aux = new cell;
   aux->next = aux;
   list = aux;
}
```

Quanto costa?

```
void head_insert(basic_list& list, DataType new_value) {
    cell* aux = new cell;
    aux->payload = new_value;
    cell* tmp = list->next;
    list->next = aux;
    aux->next = tmp;
}
```

Quanto costa?

```
void print_list(std::ostream& output_stream, basic_list list)
{
    cell* aux = list->next; // devo saltare la sentinella
    while (aux != list) {
        WriteData(output_stream, aux->payload);
        aux = aux->next;
        output_stream << std::endl;
    }
}
```

Quanto costa?

```
void read_list(std::istream& input_stream, basic_list& list)
{
    create_empty(list);
    DataType d;
    while (ReadData(input_stream, d)) {
        head_insert(list, d);
    }
}
```

Quanto costa?

```
double fact(int n)
{
if (n==0)
   return 1;
else
  return n*fact(n-1);
}
```

Quanto costa?

```
int prodRecAux(const Seq& l){
   if (isEmpty(l)) return 1;
   else return l->head * prodRecAux(l->tail);
}


int prodRec(const Seq& l){
   if (isEmpty(l)) return -1;
   else return prodRecAux(l);
}
```

Quanto costa?

```
bool isThereLesserThanRec(const Elem elem, const
Seq& l){
   if (isEmpty(l)) return false;        // se l'insieme e' vuoto
restituisce false
   if (l->head < elem) return true;
   else return isThereLesserThanRec(elem, l->tail);
}
```

Quanto costa?

```
bool areAllGreaterEqThan(const Elem elem, const Seq& l){
   return (!isThereLesserThan(elem, l)); // se l'insieme e' vuoto restituisce true
}
```

Quanto costa?

```
bool member(const Elem e, const Seq& s)
// se si sa che la sequenza è implementata mediante una lista ordinata, e' possibile ottimizzare il codice; in questo caso non facciamo alcuna assunzione
{
   if (isEmpty(s))  // se la sequenza e' vuota l'elemento non c'e': restituisco false
      return false;

   if (s->head == e)  // se la head e' l'elemento cercato restituisco true
      return true;
   else
      return member(e, s->tail); // altrimenti chiamo ricorsivamente sulla coda, restituendo il risultato della chiamata ricorsiva
}
```

Quanto costa?

```
Error insertElem(const Elem x, Seq& s)
// insertElem aggiunge sempre in prima posizione; non rispetta quindi l'ordine "temporale" con cui gli elementi vengono inseriti, ma e' molto piu' efficiente rispetto a un inserimento in coda; se stiamo implementando un insieme, in cui la posizione degli elementi non conta, va benissimo inserire nel punto che rende l'operazione piu' efficiente
{
    if (member(x, s))
       return PRESENT; // l'elemento e' gia' presente, non lo aggiungo

    cell *aux;
    try  { aux = new cell; }
    catch(...) { return FAIL;} //heap esaurito
    aux->head=x;
    aux->tail=s;
    s = aux;
    return OK;
}
```

Quanto costa?

```
Seq seqReadFromStream(istream& str)
{
    Seq s = EMPTY_SEQ;
    Elem e;
    str>>e;
    if (!str) throw runtime_error("Errore inserimento dati\n");
    while (e!= FINEINPUT)  // assumiamo che il segnale di fine input nel file sia il
numero  FINEINPUT
        {
        insertElem(e, s);
        str>>e;
        if (!str) throw runtime_error("Errore inserimento dati\n");
        }
return s;
}
```

Quanto costa?

Inserimento in coda in una lista semplice circolare con sentinella

Quanto costa?

Inserimento in coda in una lista doppiamente collegata circolare con sentinella

Quanto costa?

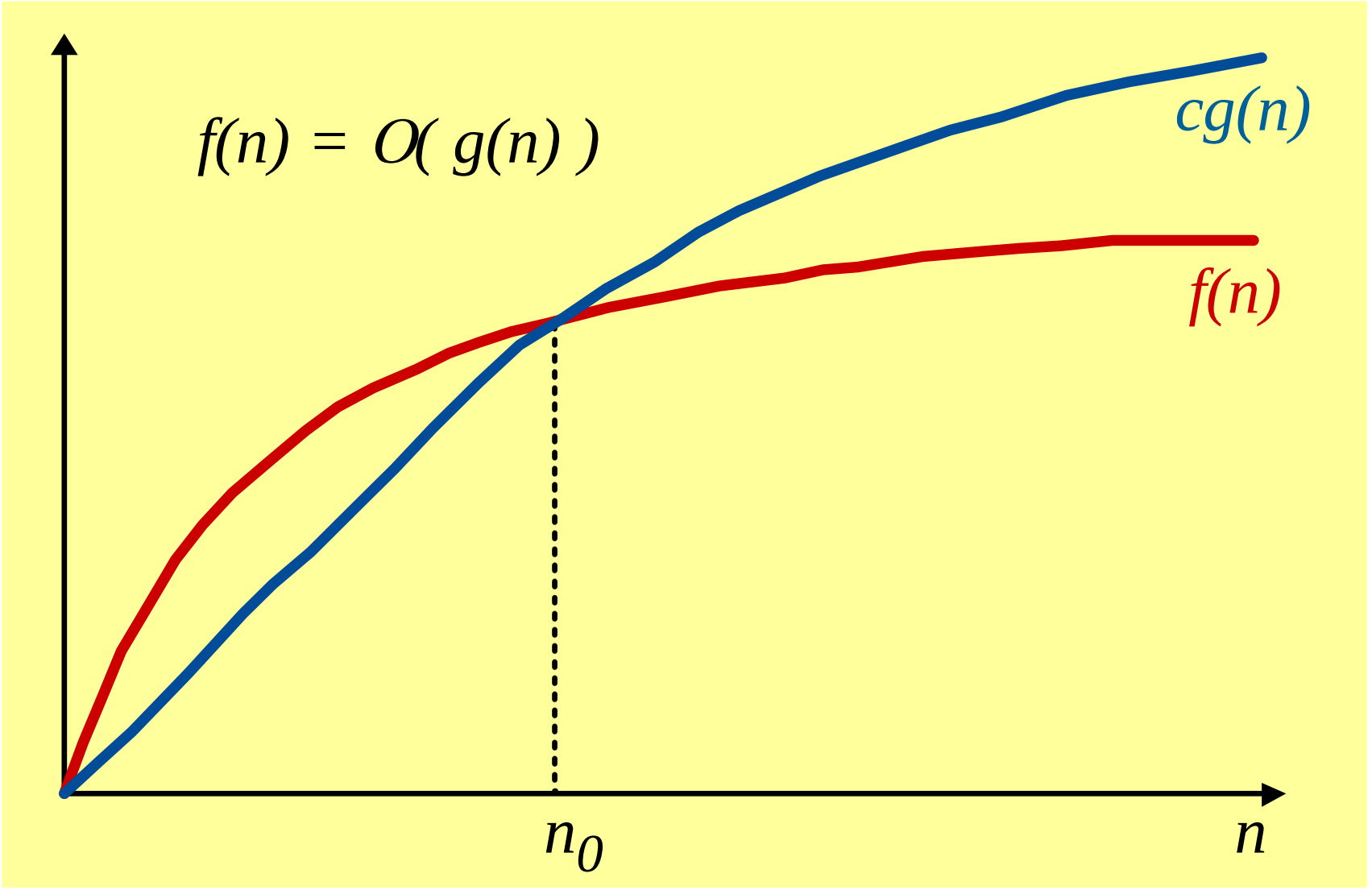
- dato un array **A** non ordinato, verificare se l'elemento **e** appartiene all'array
- dati due array **A** e **B** non ordinati, verificare se un elemento **e** appartiene a uno dei due
- dati due array **A** e **B** non ordinati, verificare se hanno un elemento in comune
- dato un array **A** non ordinato, verificare se presenta elementi duplicati
- dato un array **A** ordinato, verificare se l'elemento **e** appartiene all'array

Big-O (“O grande”), definizione

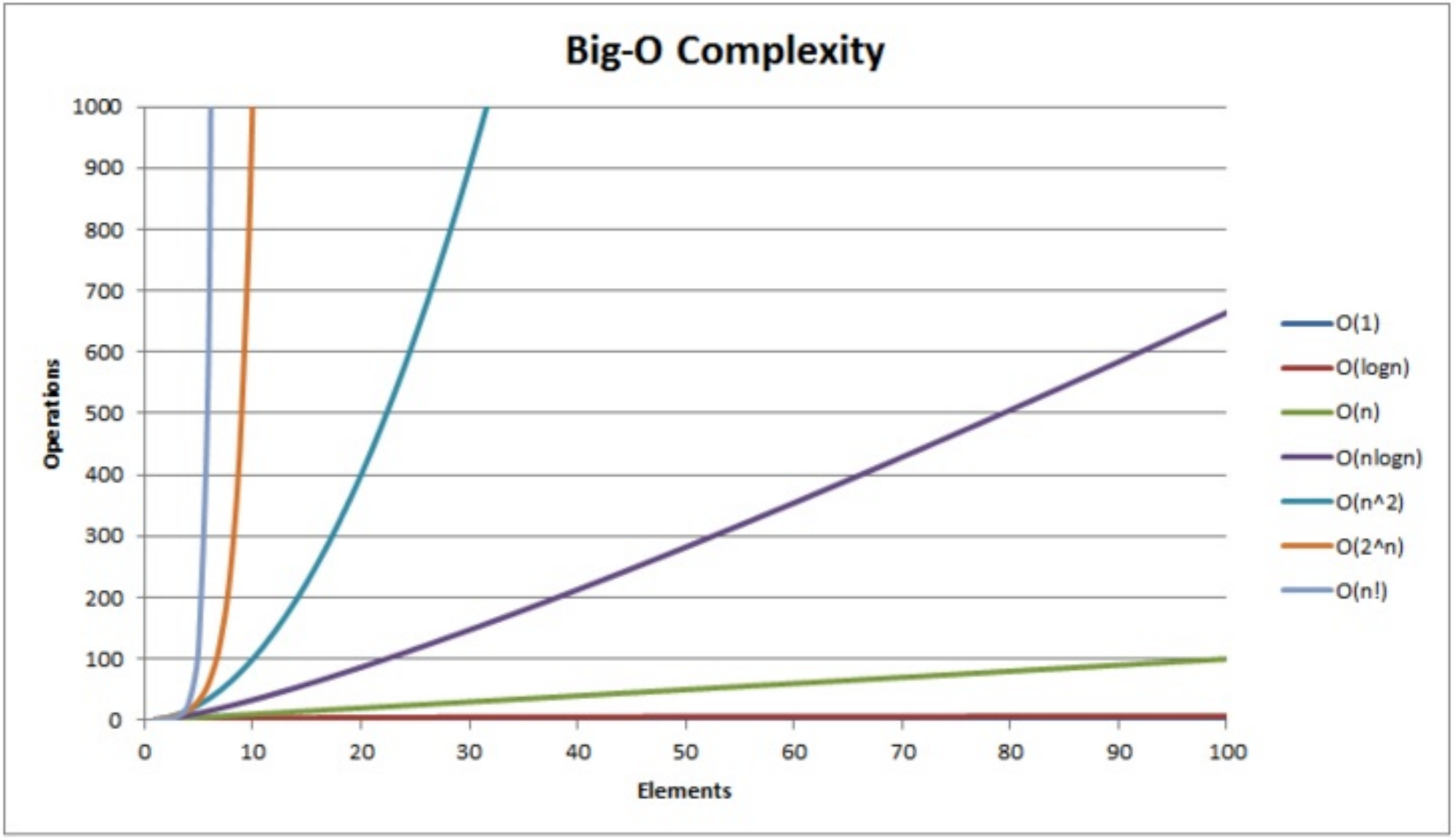
Siano f e g due funzioni dai numeri naturali ai numeri reali ≥0 ($f,g: N \rightarrow R^+$).

f(n) = O(g(n)) se ∃ due costanti c>0 e $n_0 \geq 0$ tali che $f(n) \leq c\, g(n)$ per ogni $n \geq n_0$

O(g(n)) individua una classe di funzioni, dovremmo dire “f(n) appartiene O(g(n))” e non “f(n) = O(g(n))”, ma purtroppo nei testi si trova spesso l'uguale e ci adattiamo alle presentazioni in uso


f(n) = O(g(n))
cg(n)
f(n)
n0
n

Big-O Complexity Chart



Ma quanto crescono le funzioni?

	constant	*logarithmic*	*linear*	*N-log-N*	*quadratic*	*cubic*	*exponential*
n	$O(1)$	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^3)$	$O(2^n)$
1	1	1	1	1	1	1	2
2	1	1	2	2	4	8	4
4	1	2	4	8	16	64	16
8	1	3	8	24	64	512	256
16	1	4	16	64	256	4,096	65536
32	1	5	32	160	1,024	32,768	4,294,967,296
64	1	6	64	384	4,069	262,144	1.84×10^{19}

Alcuni problemi e la loro complessità temporale

- O(1), tempo costante: determinare se un numero binario è pari o dispari.
- O(log n), tempo logaritmico: trovare un elemento in un array ordinato usando binary search.
- O(n), tempo lineare: trovare un elemento in un array non ordinato.

  O(n log n), tempo linearitmico: ordinare un array usando mergesort.

Alcuni problemi e la loro complessità temporale

- $O(n^2)$, tempo quadratico: ordinare un array usando insertion sort, selection sort, o bubble sort.
- $O(n^3)$, tempo cubico: moltiplicare due matrici quadrate, ciascuna di dimensione nxn, usando l'algoritmo classico di moltiplicazione righe per colonne

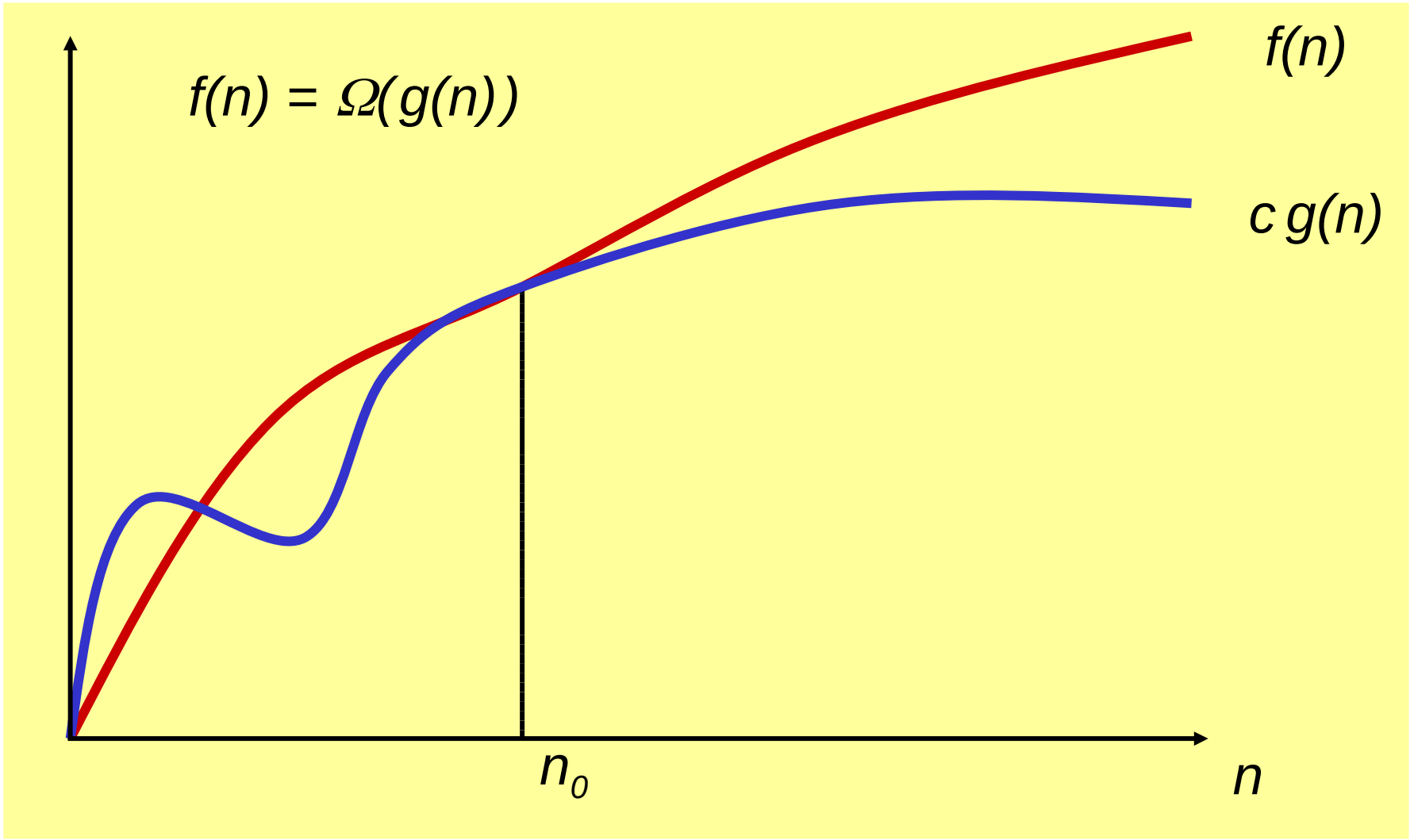
Alcuni problemi e la loro complessità temporale

- $O(c^n)$, $c>1$, tempo esponenziale: trovare la soluzione esatta al problema del commesso viaggiatore usando programmazione dinamica; determinare se due formule logiche sono equivalenti usando un algoritmo brute force.
- $O(n!)$, tempo fattoriale: trovare la soluzione esatta al problema del commesso viaggiatore usando un approccio brute force; elencare tutte le partizioni di un insieme.

Omega, definizione

Siano f e g due funzioni dai numeri naturali ai numeri reali ≥0 ($f,g: N \rightarrow R^+$).

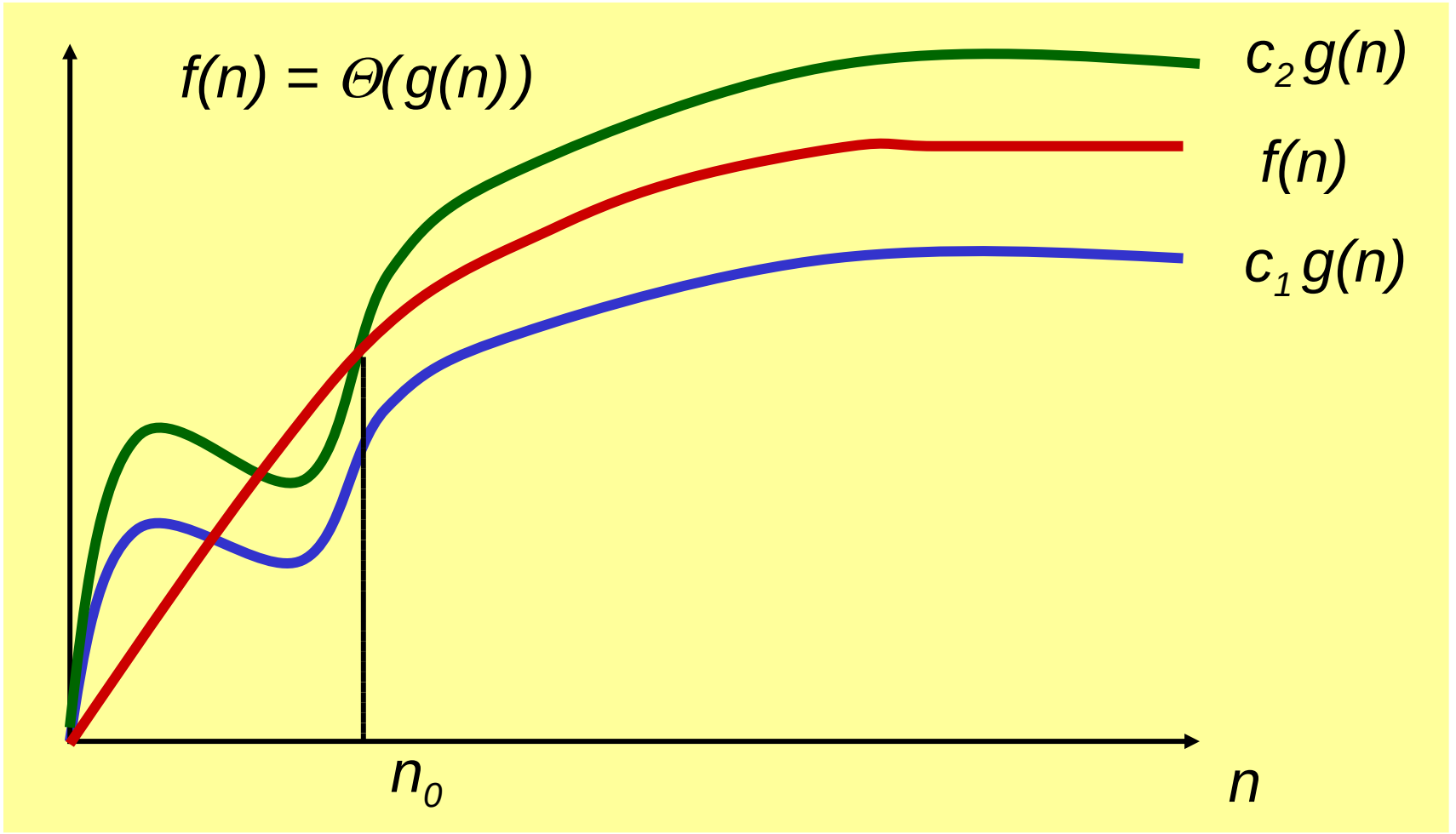
$f(n) = \Omega(g(n))$ se $\exists$ due costanti $c>0$ e $n_0 \geq 0$ tali che $f(n) \geq c\, g(n)$ per ogni $n \geq n_0$


f(n) = Ω(g(n))
f(n)
c g(n)
n0
n

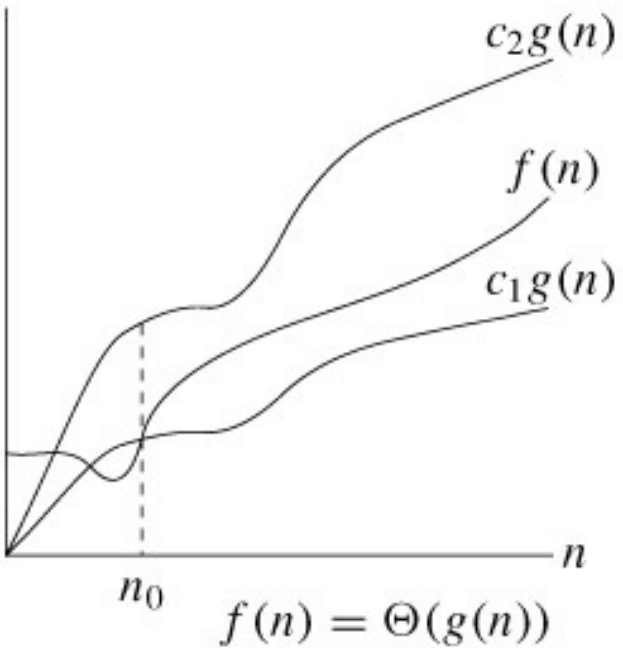
Theta, definizione

Siano f e g due funzioni dai numeri naturali ai numeri reali ≥0 ($f,g: N \rightarrow R^+$).

$f(n) = \Theta(g(n))$ se $\exists$ tre costanti $c_1, c_2 > 0$ e $n_0 \geq 0$ tali che $c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ per ogni $n \geq n_0$

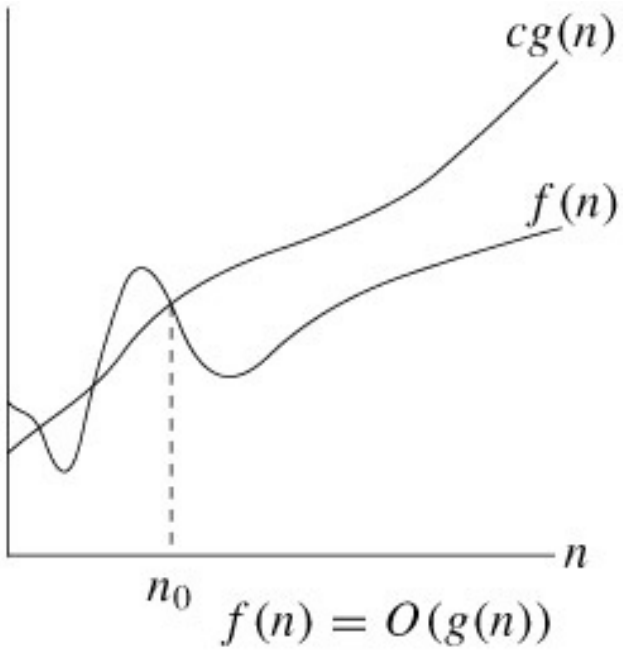

f(n) = Θ(g(n))
c2 g(n)
f(n)
c1 g(n)
n0
n

Notazioni O-grande, Omega e Theta



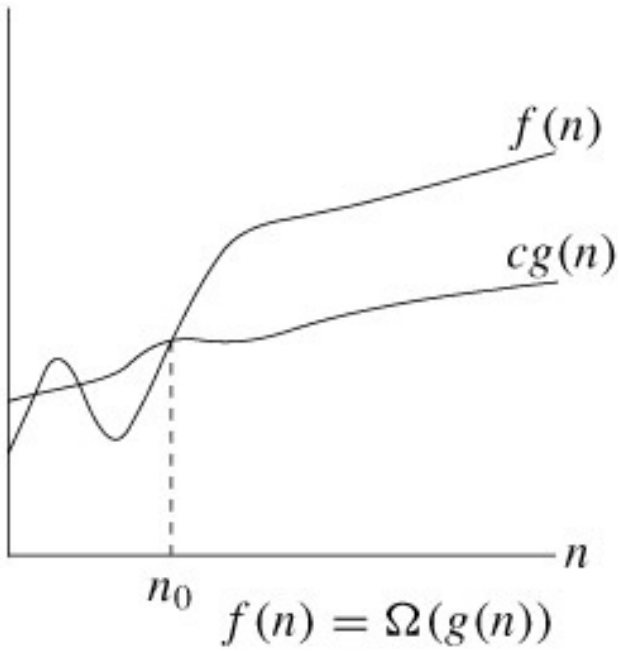
$f(n) = \Theta(g(n))$

(a)



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

(b)



$f(n) = \Omega(g(n))$

(c)

BIG OMICRON AND BIG OMEGA AND BIG THETA

Donald E. Knuth
Computer Science Department
Stanford University
Stanford, California 94305

Most of us have gotten accustomed to the idea of using the notation $O(f(n))$ to stand for any function whose magnitude is upper-bounded by a constant times $f(n)$, for all large n. Sometimes we also need a corresponding notation for lower-bounded functions, i.e., those functions which are <u>at least</u> as large as a constant times $f(n)$ for all large n. Unfortunately, people have occasionally been using the O-notation for lower bounds, for example when they reject a particular sorting method "because its running time is $O(n^2)$." I have seen instances of this in print quite often, and finally it has prompted me to sit down and write a Letter to the Editor about the situation.

The classical literature does have a notation for functions that are

Donald Knuth

People think that computer science is the art of geniuses but the actual reality is the opposite, just many people doing things that build on eachother, like a wall of mini stones.

Donald Knuth

meetville.com

Notazione O grande e analisi nel caso peggiore

La notazione O grande e l'analsi degli algoritmi nel caso peggiore, sono la stessa cosa?

NO!

E' un errore molto comune e diffuso, ma l'analisi nei casi peggiore, migliore e medio, e le notazioni O, Omega, Theta, sono concetti ortogonali.