MODELLO DI COMPUTAZIONE
COMPLESSITÀ DEGLI ALGORITMI

ALGORITMI E STRUTTURE DATI

Algoritmi e Modello di Computazione

Cos'è un algoritmo?

Definition (Algoritmo)

È una sequenza di passi "*ben definiti*" per trasformare in un tempo finito un insieme di dati in input in un insieme di dati in output.

"*ben definiti*" dipende dal modello di calcolo

Cos'è un algoritmo?

Definition (Algoritmo)

È una sequenza di passi "*ben definiti*" per trasformare in un tempo finito un insieme di dati in input in un insieme di dati in output.

"*ben definiti*" dipende dal modello di calcolo

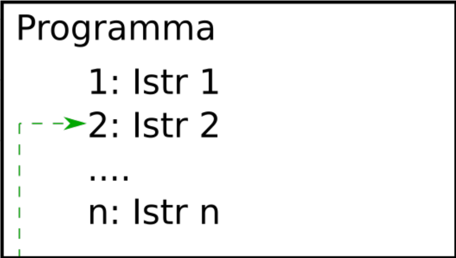
Definition (Modello di Calcolo)

È uno strumento formale per eseguire delle computazioni.

Random-Access Machine (RAM)

- memoria infinita (registri)
- ogni registro può contenere un qualsiasi numero naturale
- input (sola lettura) e output (sola scrittura)
- il programma è una sequenza di istruzioni
- il **P**rogram **C**ounter è un registro che indica la prossima istruzione da eseguire
- ad ogni istruzione eseguita il **PC** viene automaticamente incrementato (se non cambiato altrimenti)

Random-Access Machine (RAM)



Istruzioni della RAM

Istruzioni di calcolo

- `CLR(r)` assegna al registro `Rr` il valore 0;
- `INC(r)` assegna al registro `Rr` il valore contenuto in `Rr` + 1;
- `*r` ottieni il valore contenuto nel registro `r`.

  Es., Se `*5`=8, allora `**5` è il contenuto di `R8`;
- `JE(*r,s,j)` se `*r`=`s` assegna j a `PC`
- `HLT` termina l'esecuzione del programma

Cosa possiamo fare con questo numero limitato di istruzioni?

Assegnamento

Il seguente programma assegna il valore v a Rr.

1: CLR(r)
2: INC(r)
. . .
v+1: INC(r)

Possiamo estendere l'insieme delle istruzioni della RAM aggiungendo l'assegnamento ($\leftarrow$).

Es.,

1: Rr $\leftarrow$ v

Relazione d'ordine

Verifica se `*s` $<$ `*r`

```
1:  JE(*r,*s,9)
2:  R0 ← *r
3:  R1 ← *s
4:  INC(0)
5:  JE(*0,*s,9)
6:  INC(1)
7:  JE(*1,*r,11)
8:  JE(0,0,4)
9:  R0 ← 0
10: JE(0,0,12)
11: R0 ← 1
12: ...
```

Il risultato è in `R0`

Possiamo aggiungere le relazioni l'ordine ($\leq$, $<$, $=$, $>$, e $\geq$).

Espressioni Booleane

Se intepretiamo 0 come `FALSO`

Negazione di `*s`

```
1:  JE(*s,0,4)
2:  R0 ← 0
3:  JE(0,0,5)
4:  R0 ← 1
5:  ...
```

Disgiunzione logica `Rr` $\vee$ `Rs`

```
1:  R0 ← 1
2:  JE(*r,1,5)
3:  JE(*s,1,5)
4:  R0 ← 0
5:  ...
```

Possiamo aggiungere la logica Booleana ($\vee$, $\wedge$, e $\neg$).

Es.,

```
1:  Rr ← ¬(*r ∨ *s)
```

Costrutti Condizionali e Cicli

If ∗s then A else B

```
1:    JE(*s,0,c)
      A
      JE(0,0,d)
c:    B
d:    ...
```

While ∗s do A

```
1:    JE(*s,0,c+1)
      A
c:    JE(0,0,1)
c+1:  ...
```

Possiamo aggiungere i costrutti `if-then-else` e `while-do`.

Es.,

```
1:  if ¬(*r ∨ *s)
2:     Rs ← c
3:  endif
```

```
1:  while ¬(*r ∨ *s)
2:     Rs ← c
3:  endwhile
```

Somma e Sottrazione

Somma `Rs` a `Rr`

```
1:   R0 ← 0
2:   while ¬(*0=*s)
3:       INC(0)
4:       INC(r)
5:   endwhile
```

Il risultato è in `Rr`

Sottrae `Rs` da `Rr`

```
1:   R0 ← 0
2:   while (*r>*s)
3:       INC(0)
4:       INC(s)
5:   endif
```

Il risultato è in `R0`

Possiamo aggiungere somma (+) e sottrazione (−).

Es.,

```
1:   Rr ← *r + *s
2:   Rs ← *r − 5
```

Moltiplicazione e Divisione

Moltiplica `Rr` per `Rs`

```
1:  R0 ← 0
2:  R1 ← 0
2:  while (*s>*1)
3:      R0 ← *0 + *r
4:      R1 ← *1 + 1
5:  endwhile
```

Dividi `Rr` per `Rs`

```
1:  R0 ← 0
2:  R1 ← 0
2:  while (*r≥*1+*s)
3:      R1 ← *1 + *s
4:      R0 ← *0 + 1
5:  endwhile
6:  R1 ← *r − *1
```

Possiamo aggiungere moltiplicazione (*) e divisione (/).

Es.,

```
1:  Rr ← *r * *s
2:  Rs ← *r / 5
```

La nostra RAM

In sostanza possiamo semplificare la RAM e assumere di avere:

- variabili (no tipi)
- array
- costanti intere e floating point
- funzioni algebriche: $+$, $-$, $/$, $*$, $\lfloor\cdot\rfloor$, $\lceil\cdot\rceil$
- puntatori
- istruzioni condizionali e cicli (while e for)
- procedure e ricorsione (provate a capire come)
- semplici funzioni “ragionevoli” come $|\cdot|$

…e rimuovere gli indirizzi nel codice

Un Semplice Algoritmo

Input: Un array A di numeri $< a_1, \ldots, a_n >$.
Output: Il massimo tra $a_1, \ldots, a_n$.

```
def find_max(A):
    max_value ← A[1]
    for i ← 2..|A|:
        if A[i] > max_value:
            max_value ← A[i]
        endif
    endfor

    return max_value
enddef
```

La RAM NON è una Macchina Reale!!!

RAM rappresenta le macchine fisiche, ma non ha:

- la finitezza della memoria
- la finitezza della rappresentazione dei numeri
- le gerarchie della memoria

Complessità degli Algoritmi

Complessità degli Algoritmi

Complessità in termini di tempo: è la somma dei costi delle istruzioni da eseguire

Complessità in termini di spazio: è la somma dello spazio occupato

Qual'è il costo delle operazioni della RAM?

Complessità degli Algoritmi

Complessità in termini di tempo: è la somma dei costi delle istruzioni da eseguire (dipende dal modello di calcolo)

Complessità in termini di spazio: è la somma dello spazio occupato

Qual'è il costo delle operazioni della RAM?

Criterio di Costo Uniforme

Il costo in termini di tempo di:

- una operazione $+$, $-$, $*$, $/$ è 1
- un'operazione di riferimento in memoria è 1
- un'istruzione di controllo è 1
- un assegnamento è 1

Il costo in termini di spazio di un registro è 1

Criterio di Costo Uniforme: un Esempio

```
def test(n):
   Z ← 2                        // costo 1
   for i ← 1..n:                //  n+1 volte: costo 1
         Z ← Z * Z              // n volte: costo 2
   endfor

   return Z
enddef
```

Il costo totale è $1 + (n + 1) * 1 + n * 2$

Criterio di Costo Uniforme: un Esempio

```
def test(n):
   Z ← 2                          // costo 1
   for i ← 1..n:                  //  n+1 volte: costo 1
         Z ← Z * Z                // n volte: costo 2
   endfor

   return Z
enddef
```

Il costo totale è $1 + (n+1) * 1 + n * 2$ e alla fine $test(n) = 2^{2^n}\ldots$

in tempo lineare l'output occupa spazio 2^n!!!

Criterio di Costo Logaritmico

Il costo in termini di tempo di:

- una operazione $a \cdot b$ con $\cdot \in \{+, -, *, /\}$ è $\max(\log a, \log b)$
- un'operazione di riferimento in memoria è 1
- un'istruzione di controllo è 1
- un assegnamento è 1

Il costo in termini di spazio di un registro è log del valore memorizzato

Criterio di Costo Logaritmico: l'Esempio Precedente

```
def test(n):
  Z ← 2                        // costo log 2
  for i ← 1..n:                // n+1 volte: costo 1
        Z ← Z * Z              // n volte: costo log Z
  endfor

  return Z
enddef
```

Il costo totale è

$$
\begin{aligned}
\log 2 + n + 1 + \sum_{i=0}^{n-1} \log 2^{2^i} &= \log 2 + n + 1 + \sum_{i=0}^{n-1} 2^i \\
&= \log 2 + n + 1 + (2^n - 1)
\end{aligned}
$$

Costo Uniforme vs Costo Logaritmico

Il costo logaritmico modella algoritmi che lavorano con grandi numeri.

Se lo spazio occupato dai valori è limitato (come nei computer reali), il costo uniforme va benissimo.

Come Confrontare l'Efficienza?

Tempo di esecuzione?

Come Confrontare l'Efficienza?

Tempo di esecuzione? (per quale input?)

Gli algoritmi non sono programmi

Assumere costo uniforme/logaritimico non sembra essere realistico perché il tempo di esecuzione dipende da:

- l'insieme delle istruzioni della CPU
- Clock di CPU/Memoria/Bus
- il linguaggio e il compilatore usati
- come il sistema operativo gestisce la memoria
- ...

Come Confrontare l'Efficienza?

~~Cosa ne dite del tempo di esecuzione?~~

Qualche altra idea?

Come Confrontare l'Efficienza?

~~Cosa ne dite del tempo di esecuzione?~~

Qualche altra idea?

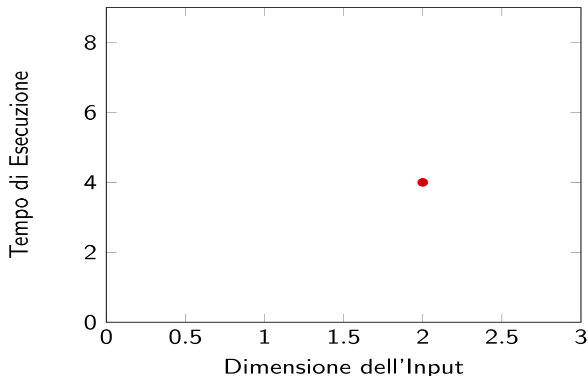
Scalabilità?

Definition (Scalabilità)

Efficienza di un sistema nel gestire crescite nella dimensione dell'input.

Crescita della Complessità

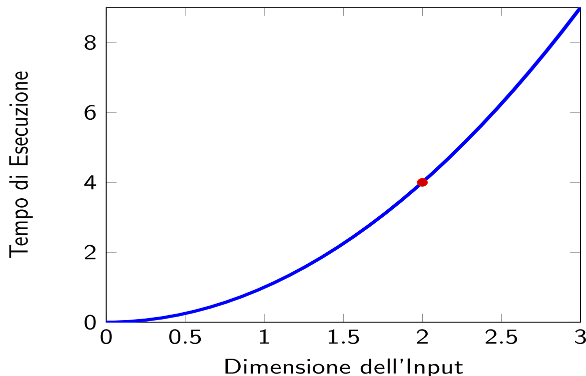
Non misuriamo il tempo di esecuzione per un dato input



Crescita della Complessità

Non misuriamo il tempo di esecuzione per un dato input

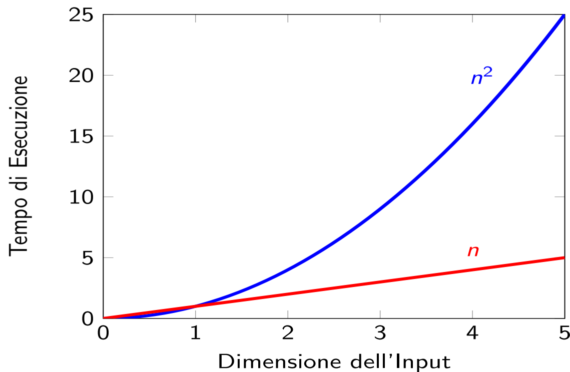
Stimiamo la relazione tra la dimensione dell'input e il tempo di esecuzione



Quiz sulla Complessità!

Quale crescita è preferibile tra:

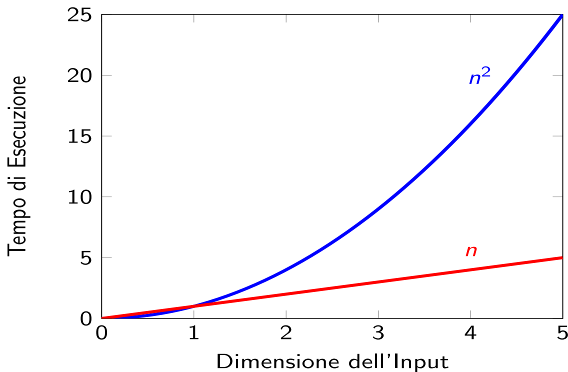
- n^2 e n?



Quiz sulla Complessità!

Quale crescita è preferibile tra:

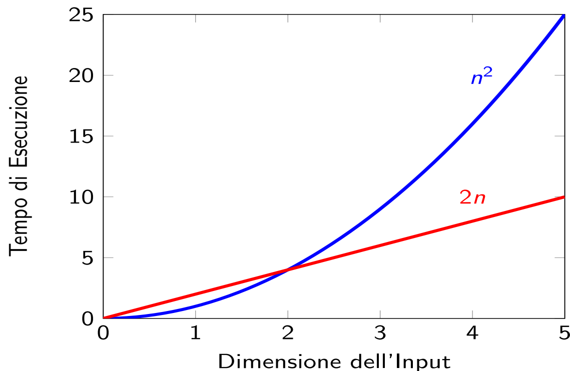
- n^2 e $\underline{n}$?



Quiz sulla Complessità!

Quale crescita è preferibile tra:

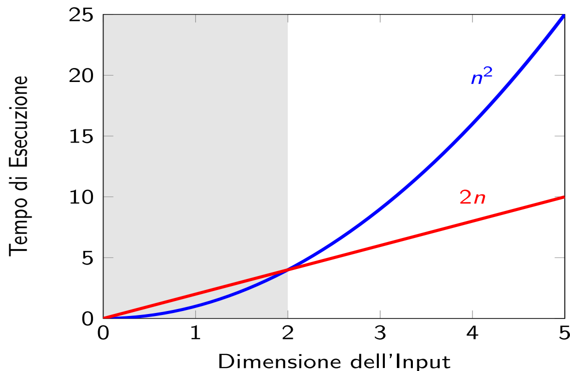
- n^2 e $\underline{n}$?
- n^2 e $2 * n$?



Quiz sulla Complessità!

Quale crescita è preferibile tra:

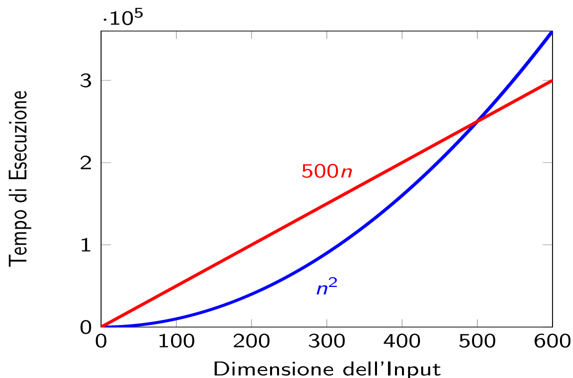
- n^2 e $\underline{n}$?
- n^2 e $\underline{2 * n}$?



Quiz sulla Complessità!

Quale crescita è preferibile tra:

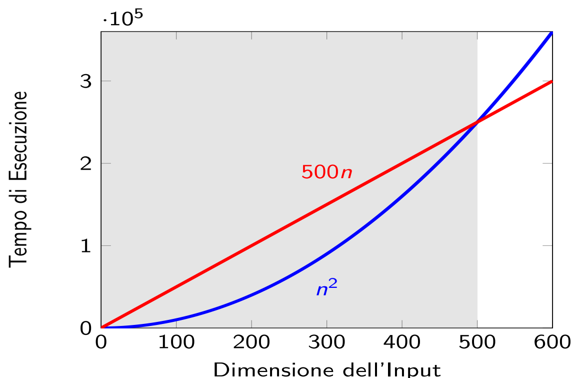
- n^2 e $\underline{n}$?
- n^2 e $\underline{2 * n}$?
- n^2 e $500 * n$?



Quiz sulla Complessità!

Quale crescita è preferibile tra:

- n^2 e n?
- n^2 e $2 * n$?
- n^2 e $500 * n$?



Complessità Asintotica

Le costanti non sono utili. Ci interessano i comportamenti asintotici.

La RAM e i criteri uniforme e logaritmico tornano a essere interessanti.

Possiamo astrarre il tempo di esecuzione della singola istruzione!!!

Complessità Asintotica

Le costanti non sono utili. Ci interessano i comportamenti asintotici.

La RAM e i criteri uniforme e logaritmico tornano a essere interessanti.

Possiamo astrarre il tempo di esecuzione della singola istruzione!!!

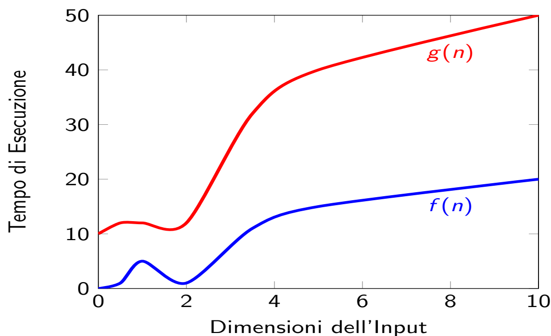
Come raggruppare tutte le funzioni che dal punto di vista asintotico si comportano nello stesso modo?

Consideriamo solo le funzioni a codominio in $\mathbb{R}_{>0}$

Notazione O-grande

$$O(f(n)) \stackrel{def}{=} \{g(n) | \exists c > 0 \exists n_0 > 0 \; m \geq n_0 \implies g(m) \leq c * f(m)\}$$

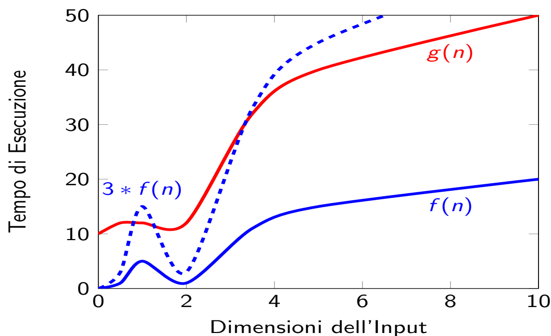
$g(n) \in O(f(n))$ se e solo se



Notazione O-grande

$$O(f(n)) \stackrel{def}{=} \{g(n) | \exists c > 0 \exists n_0 > 0 \ m \geq n_0 \Longrightarrow g(m) \leq c * f(m)\}$$

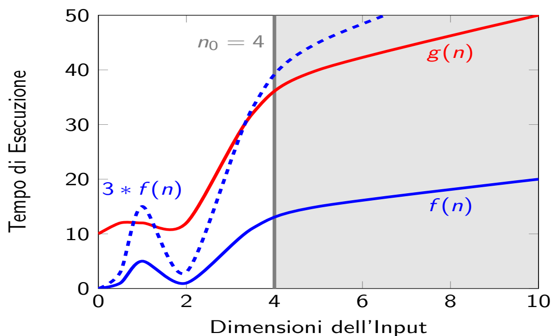
$g(n) \in O(f(n))$ se e solo se



Notazione O-grande

$$O(f(n)) \stackrel{def}{=} \{g(n) | \exists c > 0 \exists n_0 > 0 \; m \geq n_0 \implies g(m) \leq c * f(m)\}$$

$g(n) \in O(f(n))$ se e solo se



Alcuni Utili Proprietà

Per ogni $c_1, c_2 \in \mathbb{R}_{\geq 1}$ e per ogni $k \in \mathbb{Z}$

- $f(n) \in O(f(n))$
- $O(f(n)) = O(c_1 * f(n) + k)$
- $O(f(n)^{c_1}) \subseteq O(f(n)^{c_1+c_2})$ es. $n \in O(n^2)$
- se $h(n) \in O(f(n))$ e $h'(n) \in O(g(n))$, allora
  - $h(n) + h'(n) \in O(g(n) + f(n))$
  - $h(n) * h'(n) \in O(g(n) * f(n))$

Alcuni Utili Proprietà

Per ogni $c_1, c_2 \in \mathbb{R}_{\geq 1}$ e per ogni $k \in \mathbb{Z}$

- $f(n) \in O(f(n))$
- $O(f(n)) = O(c_1 * f(n) + k)$
- $O(f(n)^{c_1}) \subseteq O(f(n)^{c_1 + c_2})$ es. $n \in O(n^2)$
- se $h(n) \in O(f(n))$ e $h'(n) \in O(g(n))$, allora
  - $h(n) + h'(n) \in O(g(n) + f(n))$
  - $h(n) * h'(n) \in O(g(n) * f(n))$

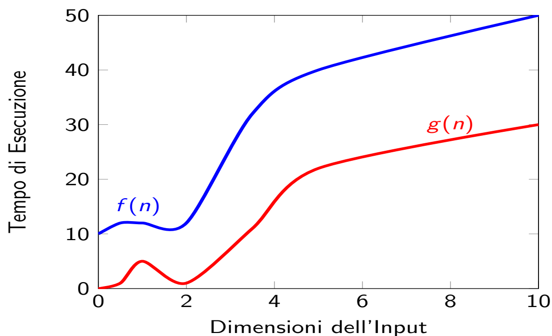
Abusando della notazione, potremmo scrivere:

- $g(n) + O(f(n))$ al posto di $O(g(n) + f(n))$
- $g(n) * O(f(n))$ intendendo $O(g(n) * f(n))$

Notazione Ω-grande

$$\Omega(f(n)) \stackrel{def}{=} \{g(n) | \, \exists c > 0 \exists n_0 > 0 \; m \geq n_0 \Longrightarrow c * f(m) \leq g(m)\}$$

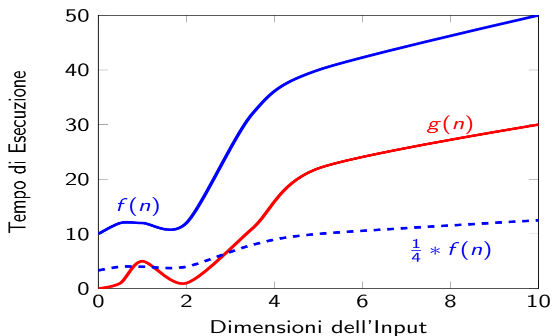
$g(n) \in \Omega(f(n))$ se e soltanto se



Notazione Ω-grande

$$\Omega(f(n)) \stackrel{def}{=} \{g(n) | \exists c > 0 \exists n_0 > 0 \; m \geq n_0 \implies c * f(m) \leq g(m)\}$$

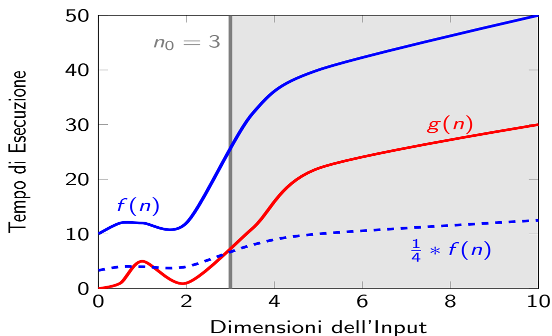
$g(n) \in \Omega(f(n))$ se e soltanto se



Notazione Ω-grande

$$\Omega(f(n)) \stackrel{def}{=} \{g(n) | \exists c > 0 \exists n_0 > 0 \ m \geq n_0 \implies c * f(m) \leq g(m)\}$$

$g(n) \in \Omega(f(n))$ se e soltanto se



Notazione Θ-theta

$$\Theta(f(n)) \stackrel{def}{=} \{g(n) | \exists c_1, c_2 > 0 \exists n_0 > 0 \quad m \geq n_0 \Longrightarrow c_1 * f(m) \leq g(m) \leq c_2 * f(m)\}$$

Theorem

$$f(n) \in \Theta(g(n)) \Longleftrightarrow f(n) \in O(g(n)) \cap \Omega(g(n))$$

Calcoliamo la Complessità di . . .

Input: Un array A di numeri $< a_1, \ldots, a_n >$.
Output: Il massimo tra $a_1, \ldots, a_n$.

```
1   def find_max(A):
2     max_value ← A[1]
3     for i ← 2..|A|:
4       if A[i] > max_value:
5         max_value ← A[i]
6       endif
7     endfor
8
9     return max_value
10  enddef
```

Criterio di Costo Logaritmico: l'Esempio Precedente

```
def test(n):
  Z ← 2                        // costo log 2
  for i ← 1..n:                // n+1 volte: costo 1
        Z ← Z * Z              // n volte: costo log Z
  endfor

  return Z
enddef
```

Il costo totale è

$$\log 2 + n + 1 + \sum_{i=0}^{n-1} \log 2^{2^i} = \log 2 + n + 1 + sum_{i=0}^{n-1} 2^i$$

$$= \log 2 + n + 1 + (2^n - 1)$$

Complessità e sviluppo tecnologico

Supponiamo di avere un computer di velocitá v, e siano n_j la dimensione delle istanze che riesce ad eseguire con un algoritmo di complessitá indicata. Come varia tale dimensione al crescere di v?

	v	$10v$	$100v$	$1000v$	kv
$O(\log_{10} n)$	n_1	n_1^{10}	n_1^{100}	n_1^{100}	n_1^k
$O(n)$	n_2	$10n_2$	$100n_2$	$1000n_2$	kn_2
$O(n^3)$	n_3	$3.16n_3$	$10n_3$	$31.6n_3$	$\sqrt{k}n_3$
$O(10^n)$	n_4	$n_4 + 1$	$n_4 + 2$	$n_4 + 3$	$n_4 + \log_{10} k$