

Fondamenti degli Algoritmi

Informatica

Alberto Casagrande

Email: `acasagrande@units.it`

a.a. 2019/2020

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.

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

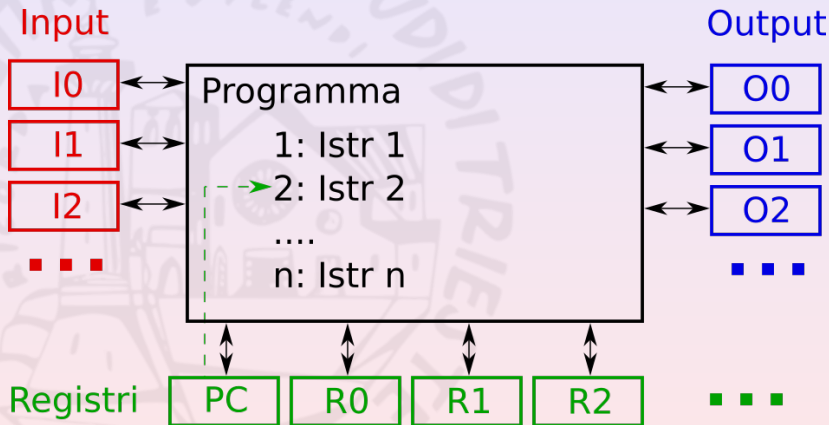
Definition (Modello di Calcolo)

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

Es., Se $*5=8$, allora $**5$ è il contenuto di R_8 ;

- **JE(**r*, *s*, *j*)** se $*r=s$ assegna j a PC
- **HLT** termina l'esecuzione del programma

Istruzioni della RAM

Istruzioni di calcolo

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

Es., Se $*5=8$, allora $**5$ è il contenuto di R_8 ;

- **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 R_r .

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

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

Es.,

```
1:  $R_r \leftarrow v$ 
```

Relazione d'ordine

Verifica se $*s < *r$

1: JE(*r,*s,9)

2: R0 \leftarrow *r

3: R1 \leftarrow *s

4: INC(0)

5: JE(*0,*s,9)

6: INC(1)

7: JE(*1,*r,11)

8: JE(0,0,4)

9: R0 \leftarrow 0

10: JE(0,0,12)

11: R0 \leftarrow 1

12: ...

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

Espressioni Booleane

Se interpretiamo 0 come FALSO

Negazione di $*s$

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

Disgiunzione logica $Rr \vee Rs$

```
1:  R0 ← 0
2:  JE(*r, 0, 6)
3:  R0 ← 1
4:  JE(*r, 0, 6)
5:  R0 ← 1
6:  ...
```

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

Es.,

```
1:  Rr ←  $\neg(*r \vee *s)$ 
```

Costrutti Condizionali e Cicli

If *s then A else B

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

While *s do A

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

Possiamo aggiungere i costrutti **it-then-else** e **while-do**.

Es.,

```
1:  if  $\neg(*r \vee *s)$ 
2:     $R_s \leftarrow c$ 
3:  endif
```

```
1:  while  $\neg(*r \vee *s)$ 
2:     $R_s \leftarrow c$ 
3:  endwhile
```

Somma e Sottrazione

Somma Rs a Rr

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

Sottrae Rs da Rr

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

Possiamo aggiungere **somma** (+) e **sottrazione** (-).

Es.,

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

Moltiplicazione e Divisione

Moltiplica R_r per R_s

```

1:   $R_0 \leftarrow 0$ 
2:   $R_1 \leftarrow 0$ 
2:  while ( $*s > *1$ )
3:     $R_0 \leftarrow *0 + *s$ 
4:     $R_1 \leftarrow *1 + 1$ 
5:  endwhile

```

Dividi R_r per R_s

```

1:   $R_0 \leftarrow 0$ 
2:   $R_1 \leftarrow 0$ 
2:  while ( $*r \geq *1 + *s$ )
3:     $R_1 \leftarrow *1 + *s$ 
4:     $R_0 \leftarrow *0 + 1$ 
5:  endwhile
6:   $R_1 \leftarrow *r - *1$ 

```

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

Es.,

```

1:   $R_r \leftarrow *r * *s$ 
2:   $R_s \leftarrow *r / 5$ 

```

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 $\langle a_1, \dots, a_n \rangle$.

Output: Il massimo tra a_1, \dots, a_n .

```
def find_max(A):  
    max_value  $\leftarrow$  A[1]  
    for i  $\leftarrow$  2.. $|A|$ :  
        if A[i] > max_value:  
            max_value  $\leftarrow$  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 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

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 0
- un assegnamento è 0

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:                        // cicla n volte  
        Z ← Z * Z                        // costo 1  
    endfor  
  
    return Z  
enddef
```

Il costo totale è $1 + n * 1$

Criterio di Costo Uniforme: un Esempio

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

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

Criterio di Costo Uniforme: un Esempio

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

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

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 0
- un assegnamento è 0

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:                        // cicla n volte  
        Z ← Z * Z                        // costo log Z  
    endfor  
  
    return Z  
enddef
```

Il costo totale è

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

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?

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



Come Confrontare l'Efficienza?

Cosa ne dite del **tempo di esecuzione**? (per quale input?)

Gli algoritmi non sono programmi

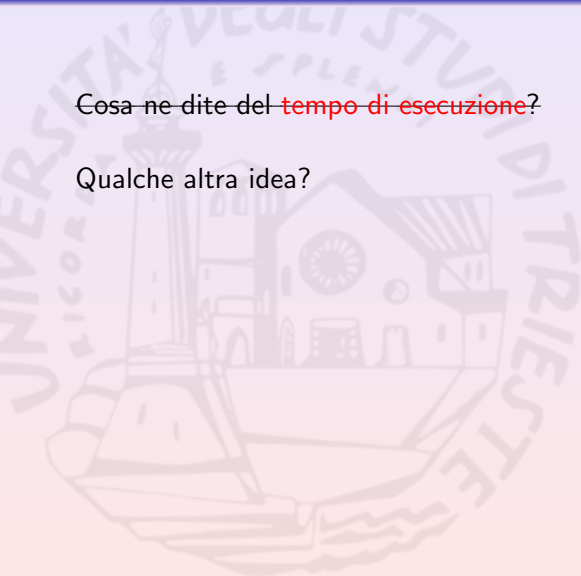
Assumere costo uniforme/logaritmico non sembra essere realistico perché il tempo di esecuzione dipende da:

- 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?

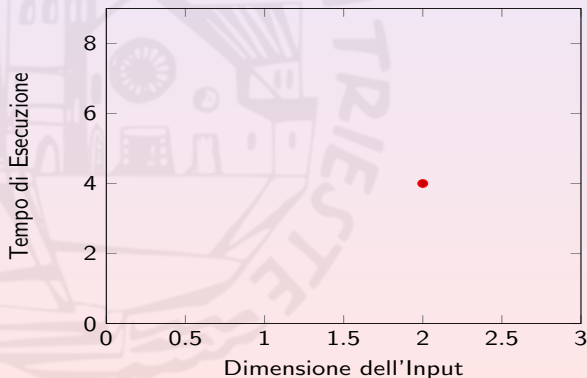
Che ne dite della scalabilità?

Definition (Scalabilità)

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

Crescita della Complessità

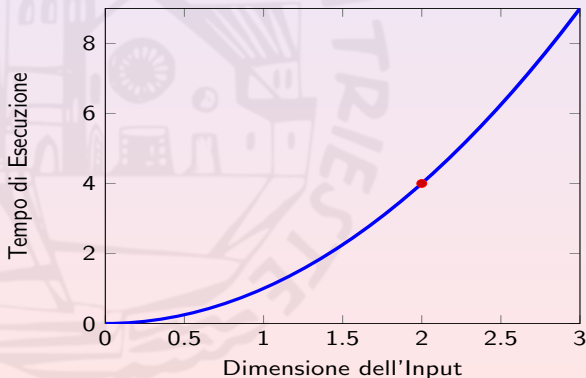
Non misuriamo il tempo di esecuzione **per un dato input**



Crescita della Complessità

Non misuriamo il tempo di esecuzione **per un dato input**

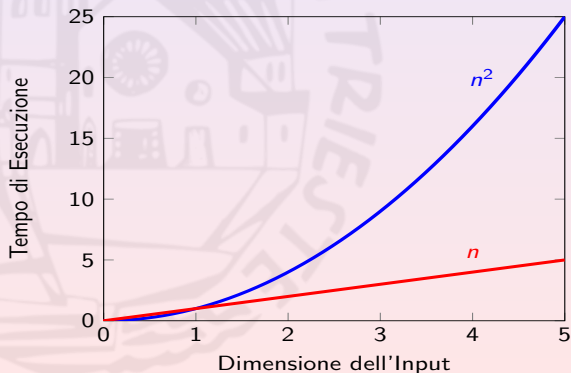
Stimiamo la relazione tra la dimensione dell'input e il tempo di esecuzione



Quiz sulla Complessità!

Quale crescita è preferibile tra:

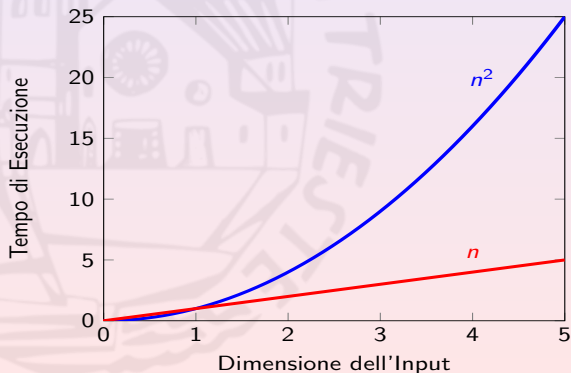
● n^2 e n ?



Quiz sulla Complessità!

Quale crescita è preferibile tra:

● n^2 e n ?

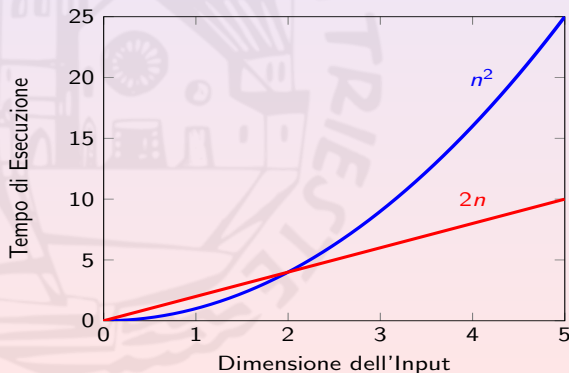


Quiz sulla Complessità!

Quale crescita è preferibile tra:

● n^2 e n ?

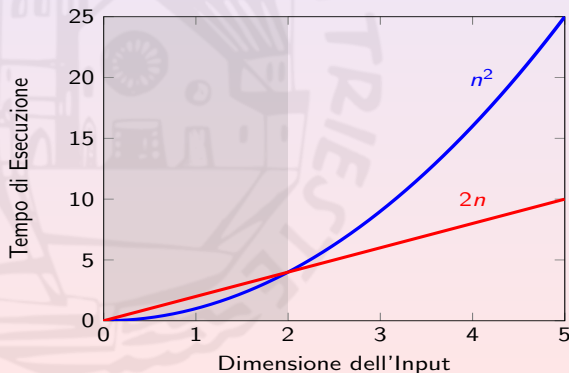
● n^2 e $2 * n$?



Quiz sulla Complessità!

Quale crescita è preferibile tra:

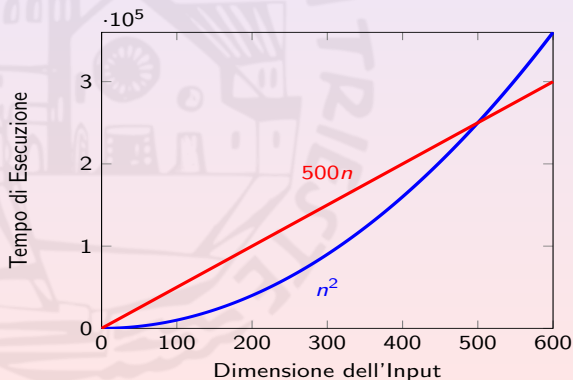
- n^2 e n ?
- n^2 e $2 * n$?



Quiz sulla Complessità!

Quale crescita è preferibile tra:

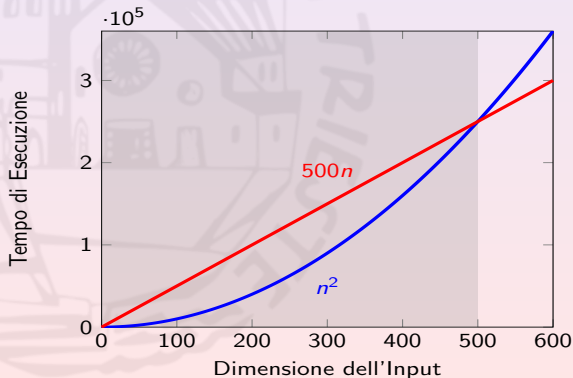
- n^2 e \underline{n} ?
- n^2 e $\underline{2 * n}$?
- n^2 e $500 * n$?



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.

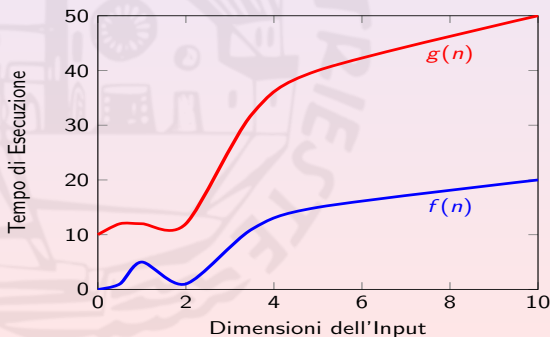
Possiamo astrarre il tempo di esecuzione della singola istruzione!!!

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

Notazione O-grande

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

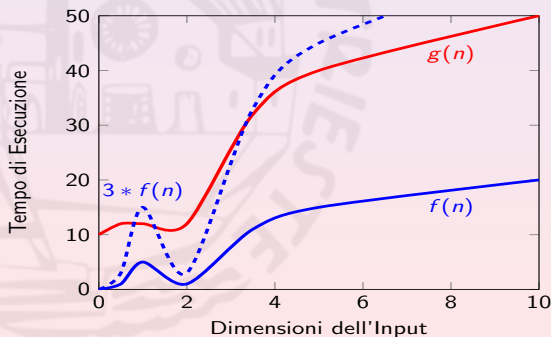
$g(n) \in O(f(n))$ se e solo se



Notazione O-grande

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

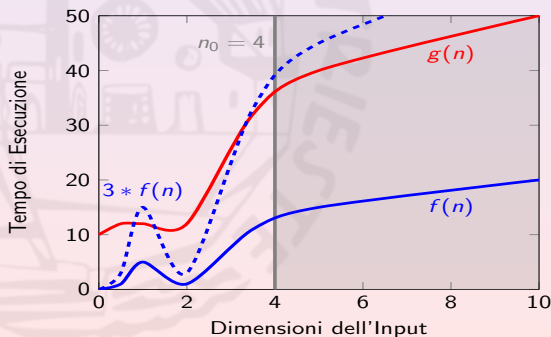
$g(n) \in O(f(n))$ se e solo se



Notazione O-grande

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

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



Alcuni Utili Proprietà

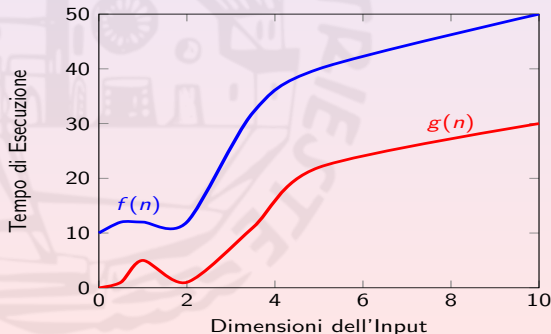
Per ogni $c_1, c_2 \in \mathbb{R}_{\geq 1}$ e per ogni $k \in \mathbb{Z}$

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

Notazione Ω -grande

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

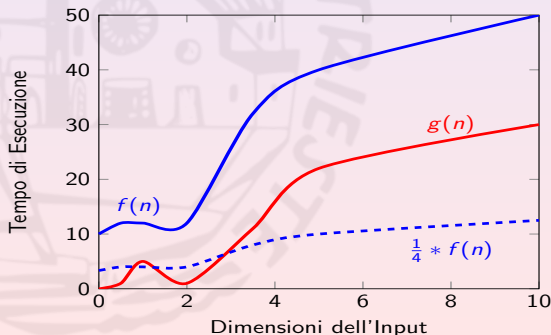
$g(n) \in \Omega(f(n))$ se e soltanto se



Notazione Ω -grande

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

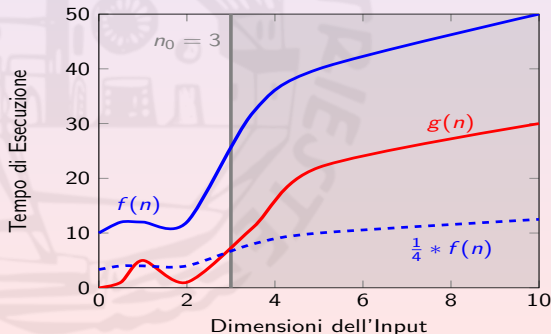
$g(n) \in \Omega(f(n))$ se e soltanto se



Notazione Ω -grande

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

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



Notazione Θ -theta

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

Theorem

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

Calcoliamo la Complessità di ...

Input: Un array A di numeri $\langle a_1, \dots, a_n \rangle$.

Output: Il massimo tra a_1, \dots, 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

```

● 2 costa $\Theta(1)$

● 4-6 costano $\Theta(1)$

● 4-6 ripetute $\Theta(n)$ volte

● 9 costa $\Theta(1)$

$\Theta(1) + \Theta(n) + \Theta(1) =$

Calcoliamo la Complessità di ...

Input: Un array A di numeri $\langle a_1, \dots, a_n \rangle$.

Output: Il massimo tra a_1, \dots, 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

```

● 2 costa $\Theta(1)$

● 4-6 costano $\Theta(1)$

● 4-6 ripetute $\Theta(n)$ volte

● 9 costa $\Theta(1)$

$\Theta(1 + 1) =$

Calcoliamo la Complessità di ...

Input: Un array A di numeri $\langle a_1, \dots, a_n \rangle$.

Output: Il massimo tra a_1, \dots, 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

```

● 2 costa $\Theta(1)$

● 4-6 costano $\Theta(1)$

● 4-6 ripetute $\Theta(n)$ volte

● 9 costa $\Theta(1)$

$$\Theta(1 + 1 * n) =$$

Calcoliamo la Complessità di ...

Input: Un array A di numeri $\langle a_1, \dots, a_n \rangle$.

Output: Il massimo tra a_1, \dots, 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
```

● 2 costa $\Theta(1)$

● 4-6 costano
 $\Theta(1)$

● 4-6 ripetute
 $\Theta(n)$ volte

● 9 costa $\Theta(1)$

$$\Theta(1 + 1 * n + 1) = \Theta(n)$$