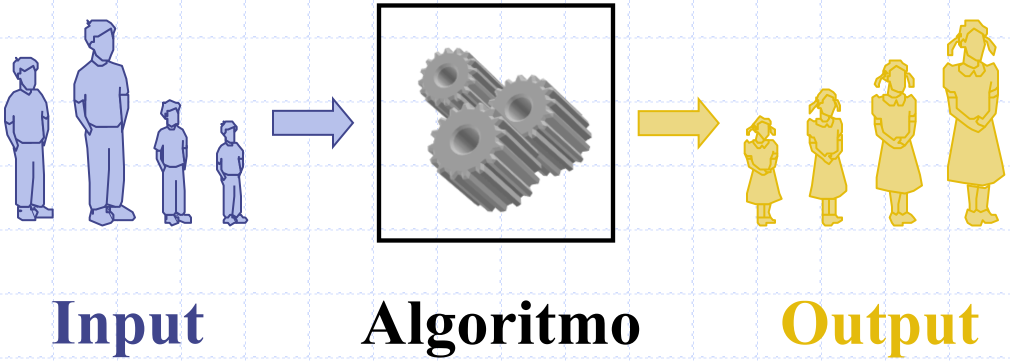
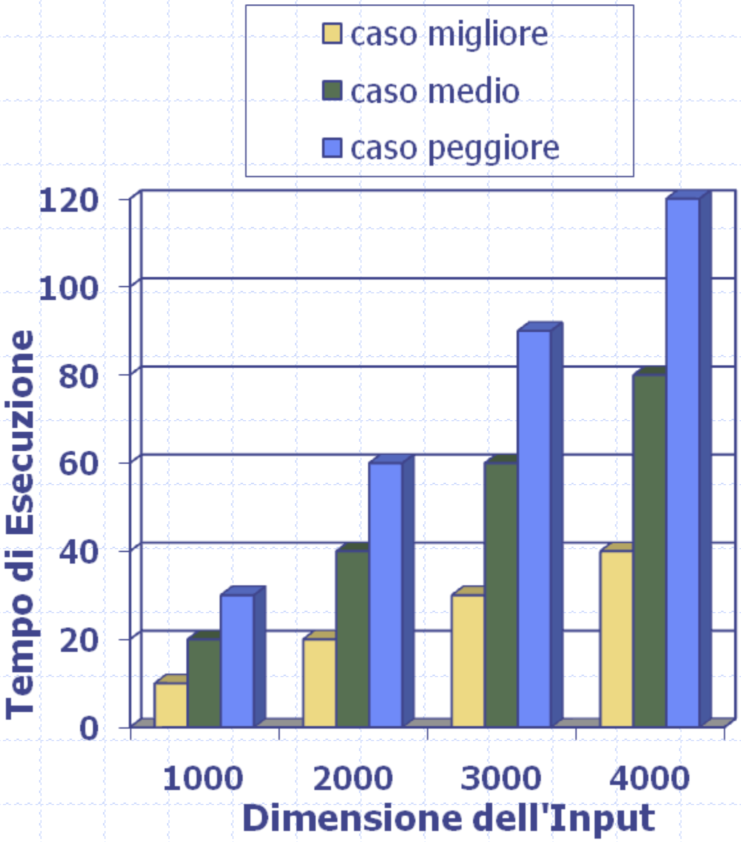
Analisi degli algoritmi



Un **algoritmo** è una procedura specificata passo per passo per risolvere un problema in una quantità di tempo finita.

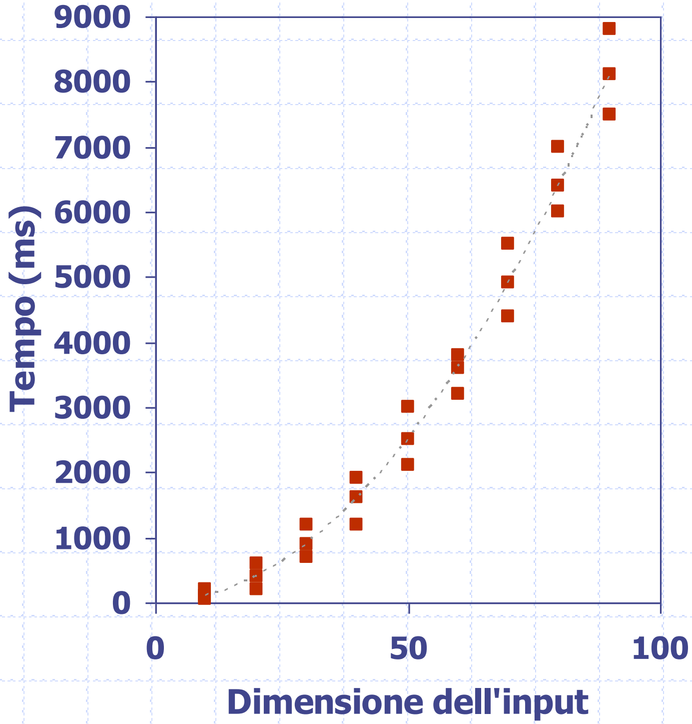
Il tempo di esecuzione

- La maggior parte degli algoritmi trasformano oggetti di input in oggetti di output.
- Il tempo di esecuzione di un algoritmo cresce tipicamente al crescere della dimensione dell'input.
- Il valore atteso del tempo di esecuzione è molto spesso difficile da determinare.
- Attribuiamo rilevanza al tempo di esecuzione nel caso peggiore.
  - Più facile da determinare.
  - Critico in molte applicazioni (giochi, finanza, robotica ecc.).



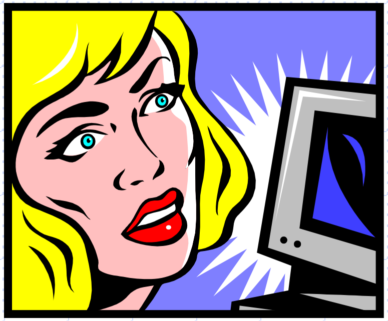
Studi sperimentali

- Scrivi un programma che implementa l'algoritmo.
- Esegui il programma su input di varie dimensioni e composizioni.
- Usa un metodo come System.currentTimeMillis() per ottenere una misura accurata del tempo di esecuzione effettivo.
- Grafica i risultati.

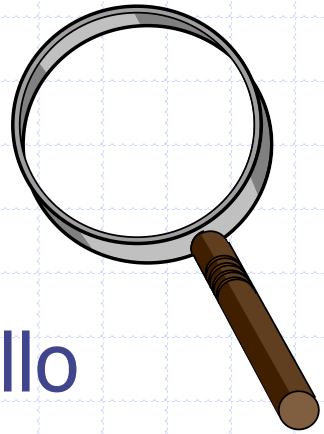


Limitazioni degli esperimenti

- È necessario implementare l'algoritmo, il che può essere difficile.
- I risultati potrebbero non essere indicativi dei tempi di esecuzione su altri input non utilizzati negli esperimenti.
- Dovendo confrontare due algoritmi, siamo costretti ad utilizzare la stessa piattaforma hardware/software.



Analisi teorica



- Utilizza una descrizione ad alto livello dell'algoritmo invece di una sua implementazione.
- Determina il tempo di esecuzione in funzione della dimensione dell'input *n*.
- Prende in considerazione tutti i possibili input.
- Ci permette di valutare la velocità di un algoritmo indipendentemente dalla piattaforma hw/sw.

Pseudocodice

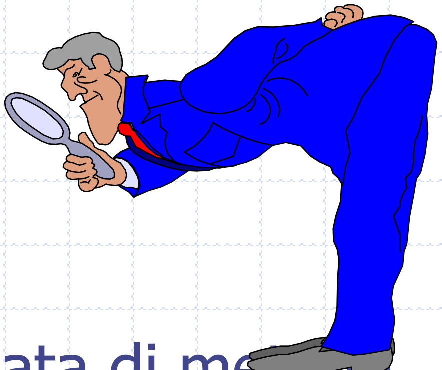
- Descrizione ad alto livello degli algoritmi.
- Più strutturato della normale prosa.
- Meno dettagliato di un programma.
- È la notazione preferita per descrivere algoritmi.
- Nasconde i dettagli della progettazione sw.

Esempio: trova il max in un array

Algoritmo *arrayMax*(*A*, *n*)
  Input array *A* di *n* interi
  Output massimo intero in *A*

  $currentMax \leftarrow A[0]$
  for $i \leftarrow 1$ **to** $n - 1$ **do**
    if $A[i] > currentMax$ **then**
      $currentMax \leftarrow A[i]$
  return $currentMax$

Pseudocodice: dettagli



- Controllo del flusso
  - **if** … **then** … [**else** …]
  - **while** … **do** …
  - **repeat** … **until** …
  - **for** … **do** …
  - uso dell'indentazione al posto delle graffe
- Dichiarazione di metodo

  Algoritmo *metodo*(*arg* [, *arg*…])

  Input …

  Output …

- Chiamata di metodo

  var.metodo (*arg* [, *arg*…])

- Valore restituito

  return *expression*

- Espressioni

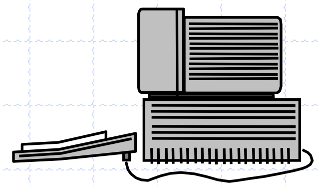
  ← Assegnazione (come = in Java)

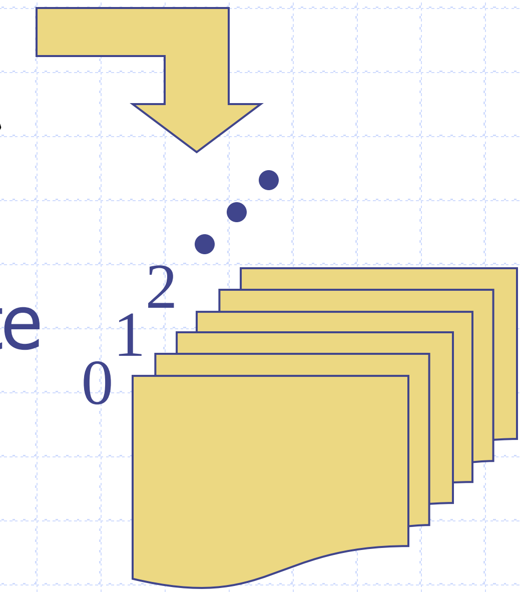
  = Eguaglianza (come== in Java)

  n^2 Consentiti gli apici ed altra notazione matematica

Il modello Random Access Machine (RAM)

- Una **CPU**

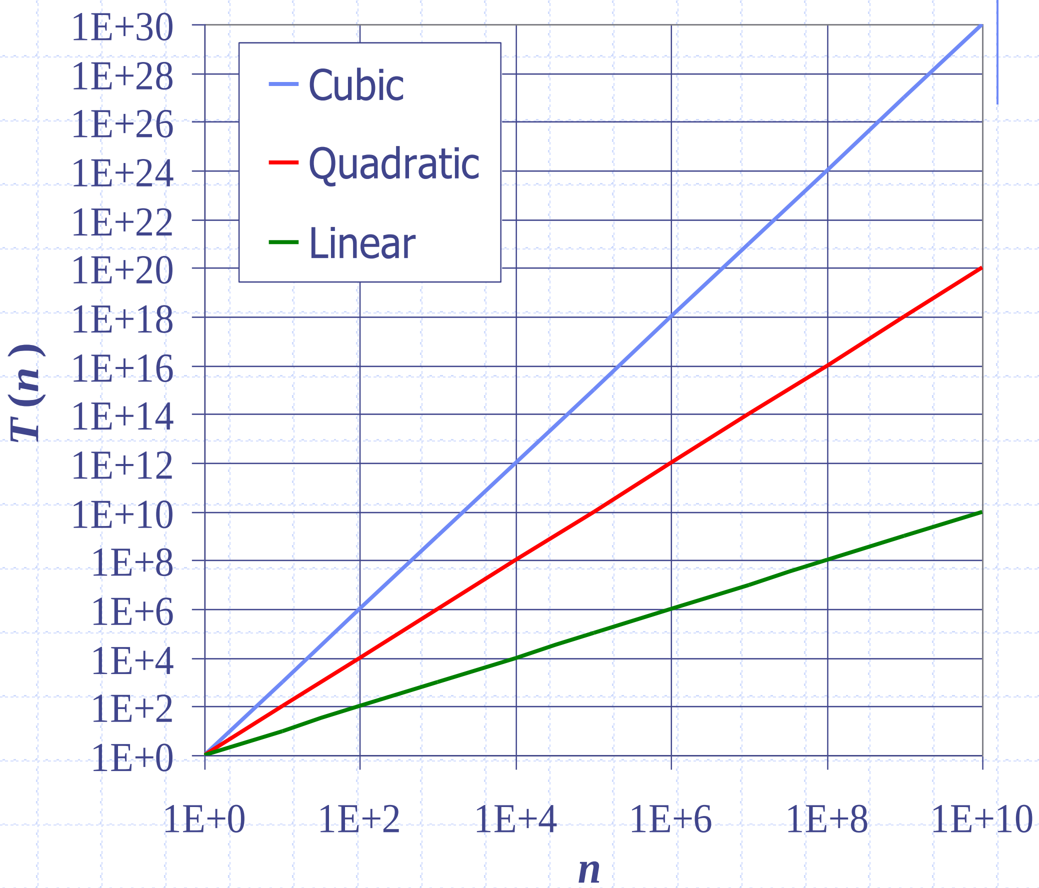




- Un banco di celle di **memoria** potenzialmente illimitato, ciascuna delle quali può contenere un numero o carattere arbitrario.
- Le celle di memoria sono numerate e possono essere consultate/aggiornate in tempo unitario.

Sette funzioni importanti

- Sette funzioni spesso presenti nell'analisi degli algoritmi:
  - Costante ≈ 1
  - Logaritmica ≈ $\log n$
  - Lineare ≈ n
  - N-Log-N ≈ $n \log n$
  - Quadratica ≈ n^2
  - Cubica ≈ n^3
  - Esponenziale ≈ 2^n
- In diagrammi log-log la pendenza delle linee corrisponde al tasso di crescita di una funzione.



Operazioni primitive

- Computazioni fondamentali eseguita da un algoritmo.
- Identificabili nello pseudocodice.
- Ampiamente indipendenti dal linguaggio di programmazione.
- L'esatta definizione non è importante (in seguito vedremo perché).
- Si assume che richiedano un tempo costante nel modello RAM.

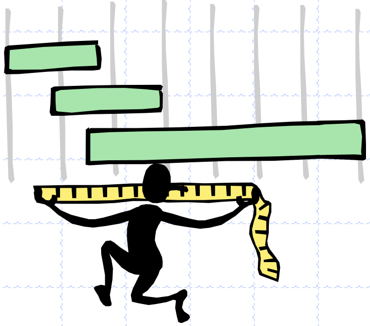
- Esempi:
  - valutazione di un'espressione
  - assegnazione di un valore a una variabile
  - accesso diretto a una cella di un array
  - chiamata di metodo
  - ritorno da un metodo

Conteggio delle operazioni primitive

- L'ispezione dello pseudocodice consente di determinare il massimo numero di operazioni primitive eseguite da un algoritmo, in funzione della dimensione dell'input.

Algorithm *arrayMax*(A, n)	# operazioni
$currentMax \leftarrow A[0]$	2
for $i \leftarrow 1$ **to** $n - 1$ **do**	$3n + 1$
if $A[i] > currentMax$ **then**	$4(n - 1)$
$currentMax \leftarrow A[i]$	$3(n - 1)$
{ increment counter i }	$2(n - 1)$
return $currentMax$	2
	Totale $12n - 4$

Calcolo del tempo di esecuzione



- L'algoritmo ***arrayMax*** esegue $8n - 2$ operazioni primitive nel caso peggiore. Definiamo:
  - a = tempo richiesto dall'operazione primitiva più *veloce*
  - b = tempo richiesto dall'operazione primitiva più *lenta*
- Indichiamo con $T(n)$ il tempo di esecuzione di ***arrayMax*** nel caso peggiore. Risulta

$$a\,(8n - 2) \leq T(n) \leq b(8n - 2)$$

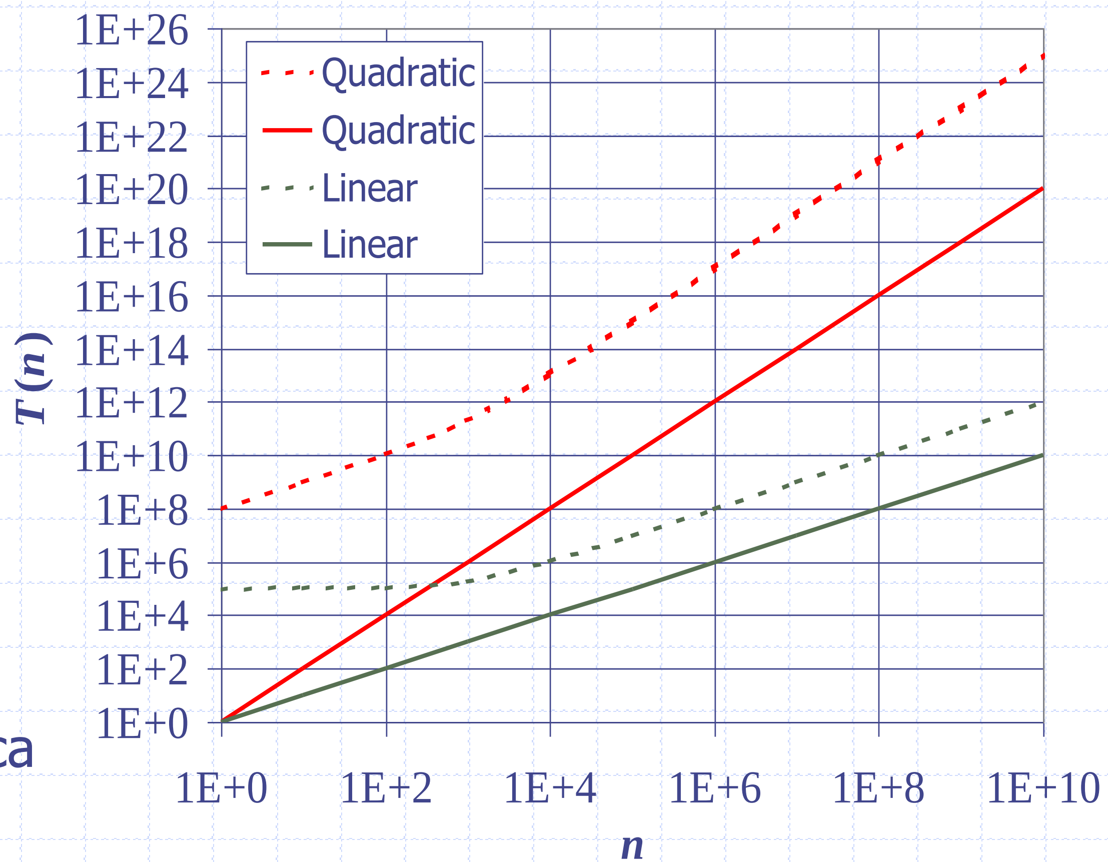
- Pertanto, il tempo di esecuzione $T(n)$ è limitato da due funzioni lineari.

Tasso di crescita del tempo di esecuzione

- La modifica della piattaforma hw/sw
  - affetta $T(n)$ per un fattore costante, ma
  - non altera il tasso di crescita di $T(n)$
- Il tasso di crescita lineare del tempo di esecuzione $T(n)$ è una proprietà intrinseca dell'algoritmo ***arrayMax***

Fattori costanti

- Il tasso di crescita non è affetto da
  - fattori costanti, o
  - termini di ordine inferiore
- Esempi
  - $10^2 n + 10^5$ è una funzione lineare
  - $10^5 n^2 + 10^8 n$ è una funzione quadratica

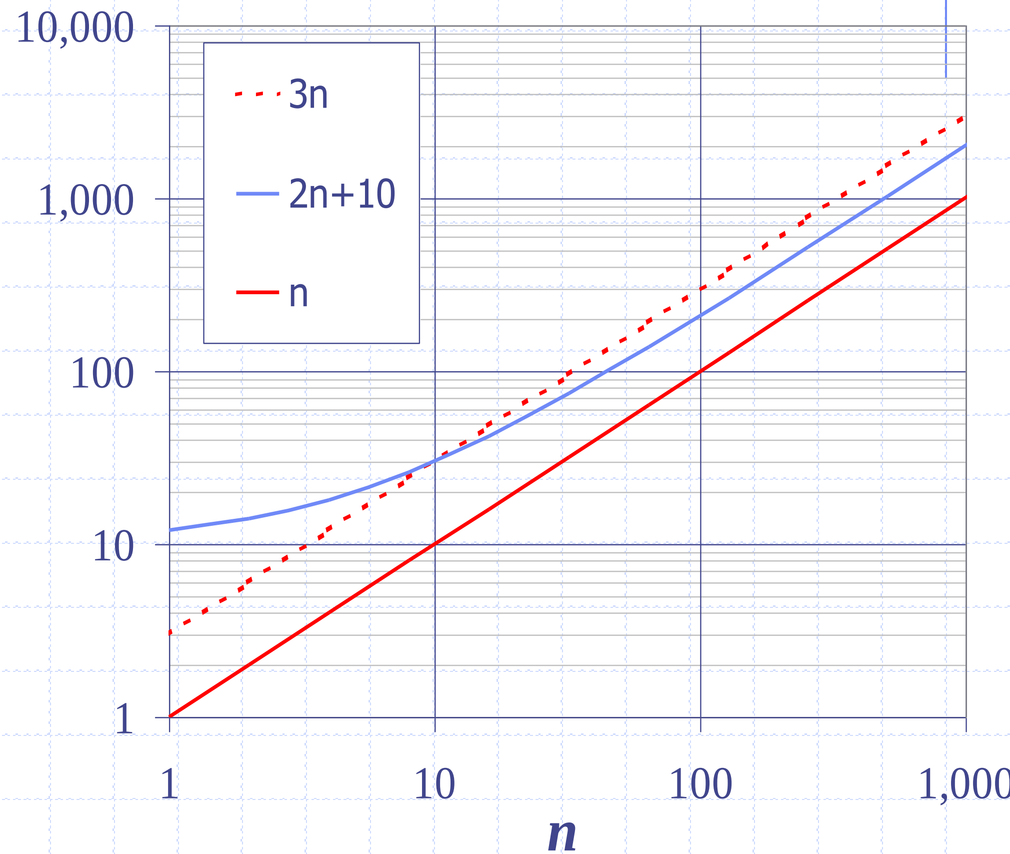


Notazione O-grande (big-Oh)

- Date due funzioni $f(n)$ e $g(n)$, si dice che $f(n)$ è $O(g(n))$ se esistono due costanti positive c e n_0 tali che

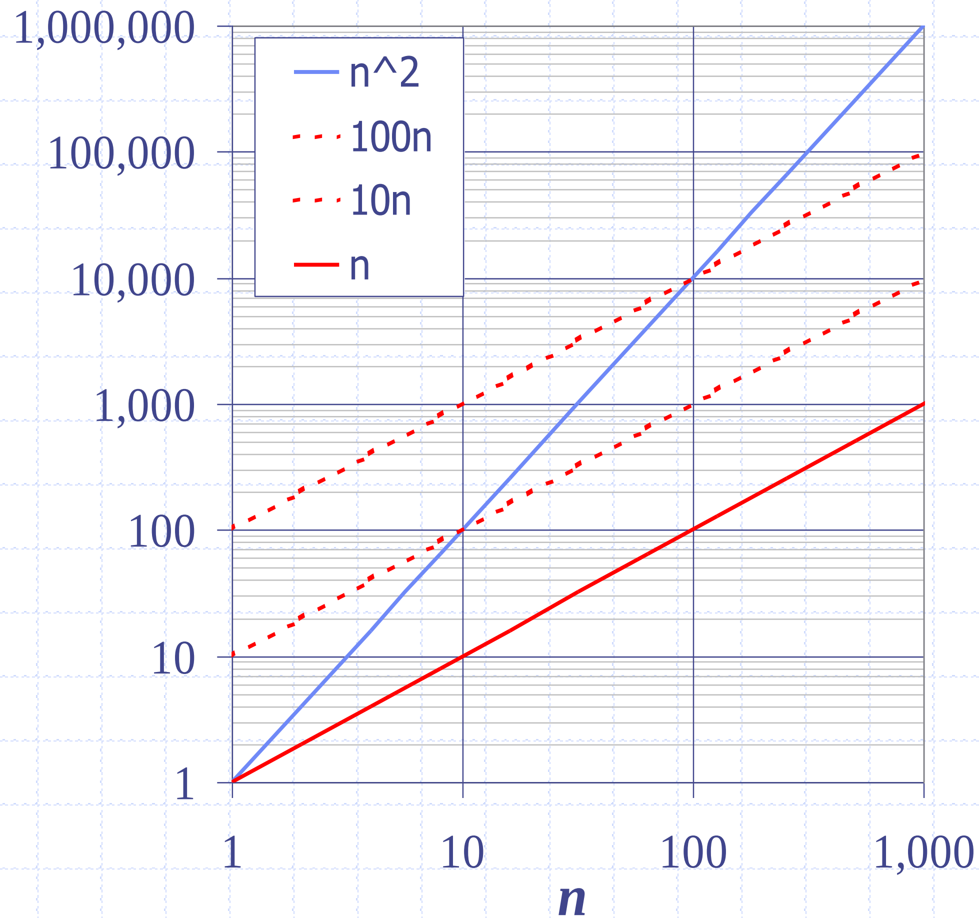
  $f(n) \leq cg(n)$ per $n \geq n_0$

- Esempio: $2n + 10$ è $O(n)$
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - scegli $c = 3$ e $n_0 = 10$

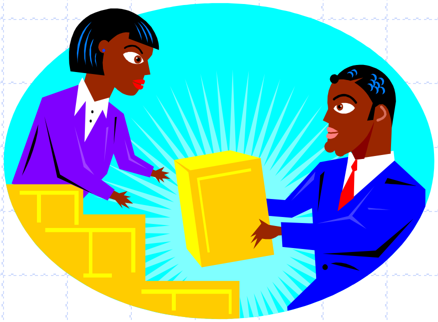


Esempio O-grande

- Esempio: la funzione n^2 non è $O(n)$
  - $n^2 \le cn$
  - $n \le c$
  - poiché c deve essere una costante, la diseguaglianza di cui sopra non può essere soddisfatta



Altri esempi O-grande



- $7n-2$

  $7n-2$ è $O(n)$

  scegliere $c > 0$ e $n_0 \geq 1$ tali che $7n-2 \leq c \bullet n$ per $n \geq n_0$

  ciò è vero per $c = 7$ e $n_0 = 1$

- $3n^3 + 20n^2 + 5$

  $3n^3 + 20n^2 + 5$ è $O(n^3)$

  scegliere $c > 0$ e $n_0 \geq 1$ tali che $3n^3 + 20n^2 + 5 \leq c \bullet n^3$ per $n \geq n_0$

  ciò è vero per $c = 4$ e $n_0 = 21$

- $3 \log n + 5$

  $3 \log n + 5$ è $O(\log n)$

  scegliere $c > 0$ e $n_0 \geq 1$ tali che $3 \log n + 5 \leq c \bullet \log n$ per $n \geq n_0$

  ciò è vero per $c = 8$ e $n_0 = 2$

O-grande e tasso di crescita

- La notazione O-grande permette di esprimere una delimitazione superiore (upper bound) al tasso di crescita di una funzione
- La frase "$f(n)$ è $O(g(n))$" significa che il tasso di crescita di $f(n)$ non supera quello della crescita di $g(n)$
- Possiamo usare la notazione O-grande per classificare le funzioni secondo il loro tasso di crescita

	$f(n)$ è $O(g(n))$	$g(n)$ è $O(f(n))$
$g(n)$ cresce di più	Sì	No
$f(n)$ cresce di più	No	Sì
stessa screscita	Sì	Sì

Regole dell'O-grande

- Se $f(n)$ è un polinomio di grado d, allora $f(n)$ è $O(n^d)$, cioè,
  1. tralascia i termini di ordine inferiore
  2. tralascia i fattori costanti
- Usa la classe di funzioni più "piccola" possibile
  - Di' "$2n$ è $O(n)$" invece di "$2n$ è $O(n^2)$"
- Usa classi di funzioni semplici
  - Di' "$3n + 5$ è $O(n)$" invece di "$3n + 5$ è $O(3n)$"

Analisi asintotica degli algoritmi

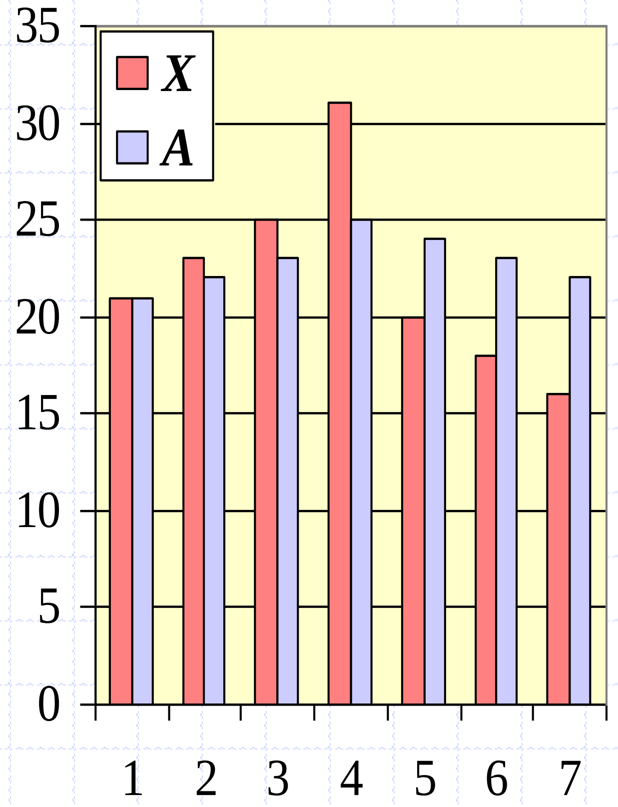
- L'analisi asintotica degli algoritmi determina il tempo di esecuzione nella notazione O-grande.
- Per effettuare l'analisi asintotica di un algoritmo:
  - calcoliamo il numero di operazioni primitive eseguite nel caso peggiore, in funzione della dimensione dell'input
  - esprimiamo tale funzione nella notazione O-grande
- Esempio:
  - Abbiamo determinato che l'algoritmo ***arrayMax*** esegue al più $8\boldsymbol{n} - 2$ operazioni primitive
  - Diciamo che l'algoritmo ***arrayMax*** "viene eseguito in tempo $\boldsymbol{O}(\boldsymbol{n})$"
- Poiché fattori costanti e termini di ordine inferiore vengono alla fine tralasciati, possiamo tralasciarli fin dal momento del conteggio del numero di operazioni primitive.

Calcolo delle medie dei prefissi

- Consideriamo come esempi di analisi asintotica due algoritmi per le medie dei prefissi.
- La i-esima media dei prefissi di un array X è la media dei suoi primi $(i+1)$ elementi:

$$A[i] = (X[0] + X[1] + \ldots + X[i])/(i+1)$$

- Il calcolo dell'array A delle medie dei prefissi di un altro array X trova applicazione nell'analisi finanziaria.



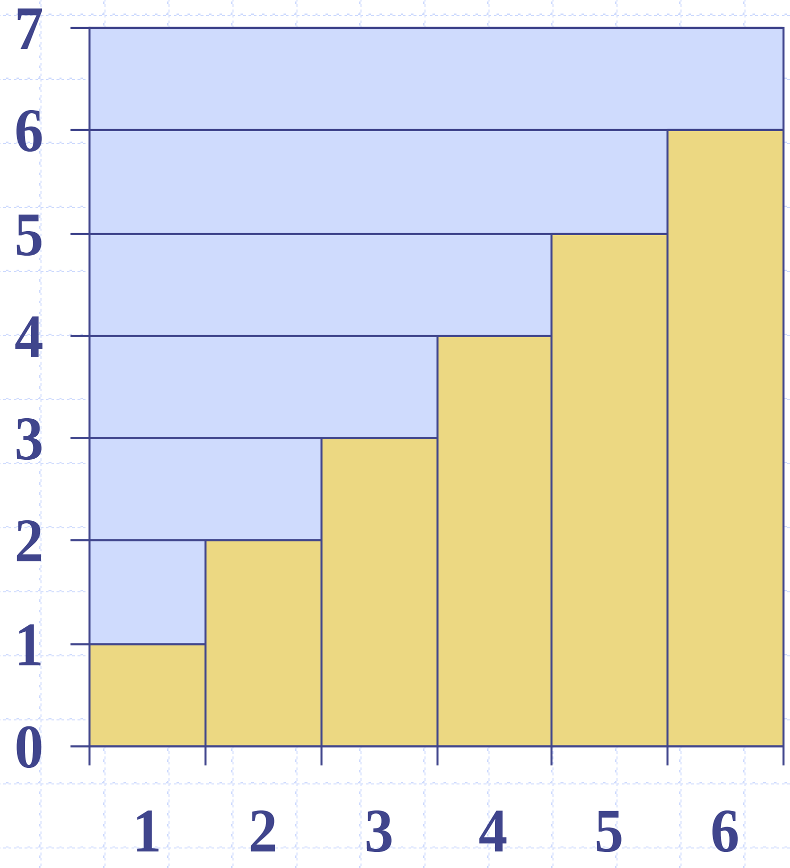
Medie dei prefissi (quadratico)

- L'algoritmo che segue calcola le medie dei prefissi in tempo quadratico applicando la definizione.

Algorithm *prefixAverages1*(X, n)	
Input array X di n interi	
Output array A delle medie dei prefissi di X	# operazioni
$A \leftarrow$ nuovo array di n interi	n
for $i \leftarrow 0$ **to** $n - 1$ **do**	n
$s \leftarrow X[0]$	n
for $j \leftarrow 1$ **to** i **do**	$1 + 2 + \ldots + (n - 1)$
$s \leftarrow s + X[j]$	$1 + 2 + \ldots + (n - 1)$
$A[i] \leftarrow s / (i + 1)$	n
return A	1

Progressione aritmetica

- Il tempo di esecuzione di ***prefixAverages1*** è $O(1 + 2 + \ldots + n)$
- La somma dei primi n interi è $n(n + 1) / 2$
  - Esiste una semplice prova visuale di ciò.
- Il tempo di esecuzione dell'algoritmo ***prefixAverages1*** è dunque $O(n^2)$.



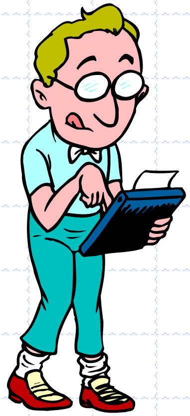
Medie dei prefissi (lineare)

- L'algoritmo che segue calcola le medie dei prefissi in tempo lineare mantenendo una somma parziale durante la sua esecuzione.

```
Algorithm prefixAverages2(X, n)
    Input array X di n interi
    Output array A delle medie dei prefissi di X
                                                #operazioni
    A ← nuovo array di n interi                 n
    s ← 0                                       1
    for i ← 0 to n − 1 do                       n
        s ← s + X[i]                            n
        A[i] ← s / (i + 1)                      n
    return A                                    1
```

- L'algoritmo *prefixAverages2* ha tempo di esecuzione $O(n)$.

Matematica da ripassare



- Sommatorie
- Logaritmi ed esponenziali

- **proprietà dei logaritmi:**

  $\log_b(xy) = \log_b x + \log_b y$

  $\log_b (x/y) = \log_b x - \log_b y$

  $\log_b xa = a \log_b x$

  $\log_b a = \log_x a / \log_x b$

- **proprietà degli esponenziali**:

  $a^{(b+c)} = a^b a^c$

  $a^{bc} = (a^b)^c$

  $a^b / a^c = a^{(b-c)}$

  $b = a^{\log_a b}$

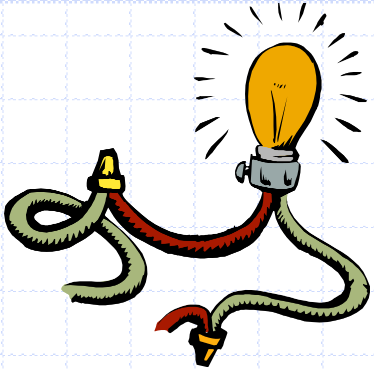
  $b^c = a^{c*\log_a b}$

- Tecniche di prova
- Elementi di base di calcolo delle probabilità

Parenti di O-grande

- **Omega-grande**
  - $f(n)$ è $\Omega(g(n))$ se esiste una constante $c > 0$
    e una costante intera $n_0 \geq 1$ tali che
    $f(n) \geq c \bullet g(n)$ per $n \geq n_0$
- **Theta-grande**
  - $f(n)$ è $\Theta(g(n))$ se esistono due costanti $c' > 0$ e $c'' > 0$ e una costante intera $n_0 \geq 1$ tali che
    $c' \bullet g(n) \leq f(n) \leq c'' \bullet g(n)$ per $n \geq n_0$

La notazione asintotica intuitivamente



O-grande

- f(n) è O(g(n)) se f(n) è asintoticamente **minore o eguale** a g(n)

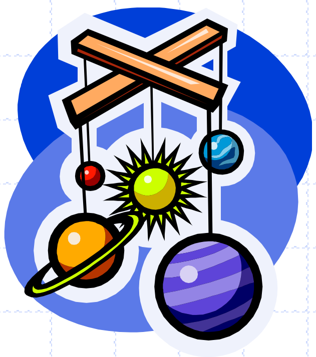
Omega-grande

- f(n) è Ω(g(n)) se f(n) è asintoticamente **maggiore o eguale** a g(n)

Theta-grande

- f(n) is Θ(g(n)) if f(n) è asintoticamente **eguale** a g(n)

Esempi



- $\mathbf{5}\boldsymbol{n}^{\mathbf{2}}$ **è** $\boldsymbol{\Omega}(\boldsymbol{n}^{\mathbf{2}})$

  $f(n)$ è $\Omega(g(n))$ se esiste una costante $c > 0$ e una costante intera $n_0 \geq 1$ tali che $f(n) \geq c \bullet g(n)$ per $n \geq n_0$

  scegliamo $c = 5$ e $n_0 = 1$

- $\mathbf{5}\boldsymbol{n}^{\mathbf{2}}$ **è** $\boldsymbol{\Omega}(\boldsymbol{n})$

  $f(n)$ è $\Omega(g(n))$ se esiste una costante $c > 0$ e una costante intera $n_0 \geq 1$ tali che $\mathrm{f}(n) \geq c \bullet g(n)$ per $n \geq n_0$

  scegliamo $c = 1$ e $n_0 = 1$

- $\mathbf{5}\boldsymbol{n}^{\mathbf{2}}$ **è** $\boldsymbol{\Theta}(\boldsymbol{n}^{\mathbf{2}})$

  $f(n)$ è $\Theta(g(n))$ se è $\Omega(n^2)$ e $O(n^2)$. Abbiamo già visto la prima; per la seconda ricordiamo che $f(n)$ è $O(g(n))$ se esiste una costante $c > 0$ e una costante intera $n_0 \geq 1$ tali che $\mathrm{f}(n) \leq c \bullet g(n)$ per $n \geq n_0$

  scegliamo $c = 5$ e $n_0 = 1$