# Algoritmi e strutture dati

**Libro di testo**: Algoritmi e strutture dati, McGraw-Hill, 2008

Un algoritmo è un procedimento per la risoluzione di un problema.

- **Sintesi**: dal problema si trova un algoritmo.
- **Analisi**: data una strategia risolutiva, si valuta la sua efficienza.

Le **strutture dati** sono modi organizzati per immagazzinare e gestire dati in modo efficiente. Esse definiscono come i dati vengano salvati in memoria e come possano essere recuperati o manipolati.

- La scelta della struttura dati influisce sull'efficienza computazionale degli algoritmi che la manipolano.

## Algoritmica

L'algoritmica è la parte dell'informatica che si occupa di tutti gli aspetti legati agli algoritmi:

- Progettazione degli algoritmi;
- Studio delle strutture dati da essi utilizzate;
- Analisi della loro efficienza;
- Studio delle limitazioni inerenti e complessità dei problemi;
- Definizione di nuovi modelli di calcolo.

### Analisi di un algoritmo

Nella progettazione di un algoritmo sono importanti:

- **Correttezza**: l'algoritmo deve risolvere il problema;
- **Efficienza**: l'algoritmo deve risolvere il problema nel minor tempo possibile.

Nella progettazione di un algoritmo è importante tenere conto dell'uso delle risorse: il tempo che ci impiega per operare, lo spazio utilizzato, il come si interfaccia con la rete, il numero di processori usati, il consumo di energia.

- Tutti questi fattori influiscono sull'efficienza dell'algoritmo. Spesso non è possibile adoperare efficientemente tutti questi aspetti, ma bisogna trovare un giusto compromesso.

Studiare l'algoritmica è importante per:

- Aspetto pratico: risolvere problemi tramite l'utilizzo dei computer.
- Aspetto teorico: forniscono delle metodologie che sono utili nelle altre discipline.

### Problemi difficili

Esistono problemi per cui si conoscono solo algoritmi inefficienti. Questi sono detti problemi difficili.
Un esempio è il problema del **commesso viaggiatore**:

- Input: $n$ città, distanza tra le varie città.
- Output: il percorso più breve che visiti tutte le città (passando una sola volta per ognuna) e torni al punto di partenza.

Gli algoritmi che risolvono questo problema calcolano le permutazioni delle città: $(n-1)!$ permutazioni.

# Gli algoritmi

Un algoritmo è un insieme ordinato e finito di passi eseguibili e non ambigui che definiscono un procedimento che termina.

- La descrizione dell'algoritmo deve essere quindi finita.
- Il passo dipende dall'ambito e dal livello di astrazione con cui sto descrivendo un algoritmo.
  - Questi passi devono essere eseguibili e non ambigui: tutto deve essere specificato e nulla può essere lasciato all'interpretazione.
- L'algoritmo deve terminare: non può quindi essere infinito.

Ci sono dei contesti in cui alcune di queste caratteristiche possono essere trascurate, permettendo di uscire quindi da questa definizione stringente.

- In alcuni algoritmi non tutto è scritto, e qualcosa viene lasciato all'esecutore: per esempio gli algoritmi randomizzati.
  - Un esempio possono essere gli algoritmi di calcolo, come l'algoritmo di **Monte Carlo** dove si fa un calcolo approssimato di un integrale di una funzione.
  - Nel **quick sort** si disordinano gli elementi e si sceglie un elemento pivot in modo randomico in modo da evitare il caso peggiore. Si migliorano quindi le prestazioni di un algoritmo.
- Alcuni algoritmi non terminano, come per esempio quelli che regolano i processi industriali.

### Programma

Un programma è un insieme ordinato e finito di istruzioni (ossia un algoritmo) scritte secondo le regole di uno specifico linguaggio di programmazione.

**Pseudocodice:**

```
ALGORITMO nome_algoritmo(parametri_con_tipo) -> tipo_di_ritorno
    istruzione1
    ...
    istruzioneN
    RETURN valore_di_ritorno
```

### Visione matematica degli algoritmi

Gli algoritmi possono essere visti come trasformazioni di input in output.

Un algortimo $A$ può essere visto come una funzione $F_a : D_i \ \rightarrow D_s$ dove:

- $D_i$ è il dominio delle istanze (gli input del problema)
- $D_s$ è il dominio delle soluzioni (gli output del problema)

Un problema $p$ prende in input un'istanza $x \in D_i$ e restituisce una soluzione $f(x) \in D_s$
L'algoritmo $A$ risolve il problema $p$ se e solo se per $\forall x \in D_i \quad F_a(x) = f(x)$

## Sintesi di algoritmi

Dato un problema si vuole ottenere un algoritmo che lo risolva. Alcuni metodi di sintesi sono la ricorsione, la tecnica *divide-et-impera*, la programmazione dinamica e le tecniche *greedy*.

## Analisi di algoritmi

Si valutano:

- **Correttezza**: dato un algoritmo $a$ e un problema $p$, dimostrare che $x$ risolve $p$ ( $\forall x \in D_i \ \ F_a(x) = f(x)$ ).
- **Efficienza**: valutare la quantità di risorse (come tempo e spazio) utilizzate dall'algoritmo; ne si studia perciò la complessità. Dopodiché si cerca di capire se è ulteriormente ottimizzabile.

Per fare l'analisi dell'algoritmo si utilizzano due metodi:

- **Valutazione a posteriori**: si decodifica l'algoritmo in un linguaggio di programmazione (testing) e lo si fa girare. Nel testing subentrano però dei problemi:
  - Esistono infiniti ingressi, e con il testing si può solo testare un sottoinsieme di questi ingressi.
  - Costo della codifica: passare dall'algoritmo al programma può essere molto costoso.

  La valutazione a posteriori è quindi un metodo insoddisfacente.
- **Valutazione a priori**: è una stima teorica in fase di progetto della correttezza e dell'efficienza dell'algoritmo.
  - Questo metodo permette di confrontare soluzioni diverse e di codificare solo quella che si ritiene tramite l'analisi a priori migliore.
  - Si utilizzano degli strumenti matematici.

## Algoritmi per la moltiplicazione

### Somme iterate

$a, b \geq 0;\ a \cdot b = a + \ldots + a$ $b$ volte

```
ALGORITMO moltiplicazione(intero a, intero b) -> intero
    prod <- 0 // linea 1                                                  [1]
    WHILE b > 0 DO                                                        [2]
        prod <- prod + a                                                  [3]
        b <- b - 1                                                        [4]
    RETURN prod                                                           [5]
```

### Correttezza

Siano $b_i$, $prod_i$ i valori di $b$, $prod$ dopo l'iterazione $i$. Si dimostri il seguente *lemma*:
sia $b_i = b - i$ per $i = 0, \ldots, b \Longrightarrow prod_i = a \cdot i$

#### Induzione su i

- **Base**: $i = 0$

$$\begin{aligned} &b_0 = b & \quad & b_0 = b - 0 = b \\ &prod_0 = 0 & & prod_0 = a \cdot 0 = 0 \end{aligned}$$

- **Induzione**: $i - 1 \to i$

$$b_i = b_{i-1} - 1$$
$$prod_i = prod_{i-1} + a$$

$$b_i = b_{i-1} - 1 \underset{\downarrow}{=} b - (i-1) - 1 = b - i + \cancel{1} - \cancel{1} = b - 1$$

ipotesi di induzione

$$prod_i = prod_{i-1} + a \overset{\uparrow}{=} a \cdot (i-1) + a = a \cdot i - \cancel{a} + \cancel{a} = a \cdot i$$

Per $i = b$ si ottiene che $b_b = 0$, e quindi termina l'esecuzione del ciclo while. Viene restituito dall'algoritmo $prod_b = a \cdot b$. Il lemma è stato dimostrato.

### Complessità

**Tempo**:
se $b = 0$ eseguo le linee $1, 2, 5 \to T = 3$
se $b > 0$:

- Le linee $1, 5$ sono eseguite una volta $\to T = 2$
- Le linee $3, 4$ sono eseguite $b$ volte $\to T = 2b$
- La linea $2$ è eseguita $b$+1 volte $\to T = b + 1$

$T_{tot} = 3b + 3 \quad \longrightarrow$ la crescita è lineare

**Spazio:** lo spazio è costante dato che dipende sempre da 3 variabili (e non da $a$ e $b$)

### Moltiplicazione alla russa

$$a \cdot b = 2a \cdot \frac{b}{2} \quad \longleftarrow \text{divisione in } \mathbb{R}$$

$$a \cdot b = \begin{cases} 2a \cdot \dfrac{b}{2} & \text{se } b \text{ pari} \\ 2a \cdot \dfrac{b-1}{2} + a & \text{se } b \text{ dispari, } b > 1 \\ a & \text{se } b = 1 \end{cases} \qquad \longleftarrow \text{divisione in } \mathbb{N}$$

```
ALGORTIMO moltiplicazione(intero a, intero b) -> intero
    prod <- 0                                                                   [1]
    WHILE b > 0 DO                                                              [2]
        IF b è dispari THEN                                                     [3]
            prod <- prod + a                                                    [4]
        b <- b/2 //divisione intera                                             [5]
        a <- a*2                                                                [6]
    RETURN prod                                                                 [7]
```

*Nota: In questo algoritmo si usano solo variabilli intere ed operazioni su interi*

#### Correttezza

Siano $a_i$, $b_i$, $prod_i$ i valori di $a$, $b$, $prod$ dopo l'iterazione $i$. Si dimostri che $a_i\, b_i + prod_i = ab$

##### Induzione su i

- **Base**: $i = 0$

$$\begin{aligned} & i = 0 \\ & a_0 = a, \quad b_0 = 0, \quad \text{prod}_0 = 0 \\ & a_0 \cdot b_0 + \text{prod}_0 = ab + 0 = ab \end{aligned}$$

- **Induzione**: $i - 1 \rightarrow i$

$$b_i = \left\lfloor \frac{b_{i-1}}{2} \right\rfloor = \begin{cases} \dfrac{b_{i-1}}{2} & \text{se } b_{i-1} \text{ è pari} \\ \dfrac{b_{i-1} - 1}{2} & \text{se } b_{i-1} \text{ è dispari} \end{cases}$$

$$\text{prod}_i = \begin{cases} \text{prod}_{i-1} & \text{se } b_{i-1} \text{ è pari} \\ \text{prod}_{i-1} + a_{i-1} & \text{se } b_{i-1} \text{ è dispari} \end{cases}$$

$$\begin{cases} \text{se } b_{i-1} \text{ è pari:} & a_i b_i + \text{prod}_i = \cancel{2}\, a_{i-1} \dfrac{b_{i-1}}{\cancel{2}} + \text{prod}_{i-1} = a_{i-1} b_{i-1} + \text{prod}_{i-1} \\ \text{se } b_{i-1} \text{ è dispari:} & a_i b_i + \text{prod}_i = \cancel{2}\, a_{i-1} \dfrac{b_{i-1}}{\cancel{2}} + \text{prod}_{i-1} + a_{i-1} = a_{i-1}(b_{i-1} - 1) + \text{prod}_{i-1} + a_{i-1} = \\ & = a_{i-1} + b_{i-1} - \cancel{a_{i-1}} + \text{prod}_{i-1} + \cancel{a_{i-1}} = a_{i-1} b_{i-1} + \text{prod}_{i-1} \end{cases}$$

In entrambi i casi si ottiene:

$$a_i\, b_i + \text{prod}_i = a_{i-1}\, b_{i-1} + \text{prod}_{i-1} \underset{\substack{\downarrow \\ \text{ipotesi di induzione}}}{=} ab$$

L'esecuzione dell'algoritmo termina quando $b = 0$. Sia $u$ il numero dell'iterazione dopo la quale $b = 0$, allora $a_u\, b_u + prod_u = a \cdot b$. Ma se $b_u = 0$, allora $prod_u = a \cdot b$

### Complessità

**Tempo:**
Si consideri $u$ il numero di iterazioni effettuate

- Le linee $1, 7$ vengono eseguite $1$ volta $\rightarrow T = 2$
- La linea $2$ viene ripetuta $u{+}1$ volte $\rightarrow T = u{+}1$
- Le linee $3, 5, 6$ vengono eseguite $u$ volte $\rightarrow T = 3u$
- La linea 4 viene eseguita al più $u$ volte $\rightarrow T \leq u$

$T_{tot} \leq 5u + 3$

| $\mathbf{b}$ | $0$ | $1$ | $2$ | $3$ | $4$ | $5$ | $6$ | $7$ | $8$ | $\dots$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{u}$ | $0$ | $1$ | $2$ | $2$ | $3$ | $3$ | $3$ | $3$ | $4$ | $\dots$ |

Da ciò si ricava che $u = \lfloor \log_2 b \rfloor + 1$

$T(a, b) \leq 5\ (\lfloor \log_2 b \rfloor + 1) + 3 = 5 \lfloor \log_2 b \rfloor + 8$

- La crescita del tempo non dipende da $a$ ed è logaritmica

**Spazio:** lo spazio è costate: non dipende da $a$ e da $b$ ma si utilizzano solo 3 variabili

## Crescite lineari e logaritmiche

| $\mathbf{b}$ | $\mathbf{1}$ | $\mathbf{2}$ | $\dots$ | $\mathbf{8}$ | $\dots$ | $\mathbf{1000}$ | $\dots$ | $\mathbf{1'000'000}$ | $\dots$ |
|---|---|---|---|---|---|---|---|---|---|
| $T(a, b) = 3b + 3$ | $6$ | $9$ | $\dots$ | $27$ | $\dots$ | $3003$ | $\dots$ | $3'000'003$ | $\dots$ |
| $T(a, b) = 5 \lfloor log_2 b \rfloor + 8$ | $8$ | $13$ | $\dots$ | $23$ | $\dots$ | $53$ | $\dots$ | $103$ | $\dots$ |

C'è quindi una sostanziale differenza tra una crescita lineare ed una logaritmica.

# Calcolo della potenza $x^y$

Dati $x, y \geq 0$ interi, bisogna trovare un algoritmo che calcoli $x^n$.

## Prodotti iterati

$x^y = x\ \dots\ x$ per $y$ volte

```
ALGORITMO potenza(intero x, intero y) -> intero
    power <- 1                                                                      [1]
    WHILE y > 0 DO                                                                  [2]
        power <- power * x                                                          [3]
        y <- y - 1                                                                  [4]
    RETURN power                                                                    [5]
```

**Correttezza**: dopo l'iterazione $i$ si ha che $y_i = y - i$ e $power_i = x^i$. Si fanno $y$ iterazioni e il risultato finale è $x^y$

**Complessità** (è analoga a quella della moltiplicazione a somme iterate):

- Numero di righe di codice: $T(x, y) = 3y + 3$
- Spazio: 3 variabili (i due parametri e la variabile power)

## Soluzione ricorsiva

$x^y = x^{2\frac{y}{2}} = (x^{\frac{y}{2}})^2 \quad \longleftarrow$ divisione in $\mathbb{R}$ con $y \in \mathbb{R}$

$$x^y = \begin{cases} 1 & \text{se y} = 0 \\ (x^{\frac{y}{2}})^2 & \text{se y} > 0 \ \wedge \ \text{y pari} \\ (x^{\frac{y-1}{2}})^2 \, x & \text{se y} > 0 \ \wedge \ \text{y dispari} \end{cases} \qquad \longleftarrow \text{divisione in } \mathbb{N} \text{ con } y \in \mathbb{N}$$

```
ALGORITMO potenza(intero x, intero y) -> intero
    IF y = 0 THEN                                                                  [1]
        RETURN 1                                                                   [2]
    ELSE
        power <- potenza(x, y/2) //divisione intera                                [3]
        power <- power * power                                                     [4]
        IF y è dispari THEN                                                        [5]
            power <- power * x                                                     [6]
        RETURN power                                                               [7]
```

*Nota: in questo algoritmo si usano solo variabilli intere ed operazioni su interi*

### Correttezza

Si dimostri che $\forall \text{x, y} \geq 0 \;\; \text{potenza(x, y)}$ restituisce $x^y$

#### Induzione su $\mathbf{y}$

- **Base**: $y = 0$

  $\text{restituisce 1 e } x^y = x^0 = 1$

- **Induzione**: $< y \rightarrow y \quad$ (si suppone che sia vera per tutti i valori minori di un certo y)
  - *Caso $y$ pari*

    $$\underset{\substack{\downarrow \\ \text{risultato di potenza(x, y/2)} \;\; \text{(per ipotesi di induzione)}}}{(x^{\frac{y}{2}})^{2}} = x^y \rightarrow \text{risultato}$$

  - *Caso $y$ dispari*

    $$\text{potenza}(x, y/2) = \underset{\substack{\downarrow \\ \text{per ipotesi di induzione}}}{x^{\lfloor \frac{y}{2} \rfloor}} = x^{\frac{y-1}{2}}$$

    $(x^{\frac{y-1}{2}})^{2} = x^{y-1}$

    $x^{y-1} \, x = x^{y}$

### Complessità

#### Tempo

Sia $T(x, y)$ il tempo misurato come numero di righe di codice che vengono eseguite su input $x, y$

- $y = 0$ vengono eseguite le linee $1, 2 \rightarrow T = 2$
- $y > 0$ :
  - vengono eseguite le linee $1, 3, 4, 5, 7 \rightarrow T = 5$
  - viene eseguita la linea $6$ per $y$ dispari $\rightarrow T \leq 1$
  - la linea $3$ esegue una chiamata ricorsiva con un suo tempo $\rightarrow T(x, \lfloor \frac{y}{2} \rfloor)$

$$T_{tot} \leq 6 + T(x, \lfloor \frac{y}{2} \rfloor)$$

$$T(x, y) = \begin{cases} 2 & \text{se } y = 0 \\ 6 + T(x, \lfloor \frac{y}{2} \rfloor) & \text{altrimenti} \end{cases} \quad \longrightarrow \text{equazione di ricorrenza}$$

**Risoluzione dell'equazione di ricorrenza** *(si supponga per semplicità che $y$ sia una potenza di 2)*:

$$\begin{aligned} T_{tot} &= 6 + T(x, \lfloor \frac{y}{2} \rfloor) \\ &= 6 + 6 + T(x, \frac{y}{2^2}) \\ &= 6 + 6 + 6 + T(x, \frac{y}{2^3}) = \ldots \\ &= 6k + T(x, \frac{y}{2^k}) \end{aligned}$$

Bisogna fare in modo di ricondursi al caso base $y = 0$ partento dall'equazione di sopra

Dato che con $k = \log_2 y \Rightarrow y = 2^k \Rightarrow \frac{y}{2^k} = 1,$ scelgo allora $k = 1 + \log_2 y :\quad \frac{y}{2^k} = \frac{y}{2^{\log_2 y + 1}} = \lfloor \frac{1}{2} \frac{y}{2^{\log_2 y}} \rfloor = \lfloor \frac{1}{2} \rfloor \frac{\cancel{y}}{\cancel{y}} = 0$

$$\begin{aligned} &= 6\ (1 + \log_2 y) + T(x, 0) \ \leftarrow \text{caso base} \\ &= 6 + 6 \log_2 y + 2 \\ &= 8 + 6 \log_2 y \end{aligned}$$

La crescita del tempo è perciò logaritmica: $T(x, y) \leq 6 \log_2 y + 8$

#### Spazio

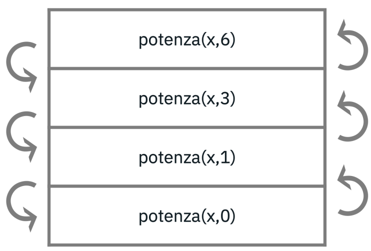
La ricorsione viene gestita utilizzando una struttura che si chiama **stack della ricorsione**, in cui vengono impilate le chiamate attive.

Associata ad ogni chiamata ricorsiva c'è un **record di attivazione**, ossia la struttura che contiene i dati della singola chiamata. Quando si chiama l'algoritmo, vengono aggiunte sulla pila dei record di attivazione per ciascuna chiamata. La dimensione di un record di attivazione è costante.

#### Altezza della pila

$$H(x, y) = \begin{cases} 1 & \text{se } y = 0 \\ 1 + H(x, \lfloor \frac{y}{2} \rfloor) & \text{altrimenti} \end{cases}$$

Questa equazione si risolve come quella precedente, e si ottiene: $H(x, y) = 2 + \log_2 y$
Essendo in questo caso la dimensione del record di attivazione pari a 3 variabili, si trova che lo spazio (inteso come numero di variabili) corrisponde a $3(2 + \log_2 y)$




*esempio dell'utilizzo dello stack per un algoritmo che calcola la potenza 6-esima di un qualsiasi numero x*

*Sia il tempo che lo spazio di questo algoritmo crescono come un logaritmo di $y$*

### Algoritmo alternativo

```
ALGORITMO potenza(intero x, intero y) -> intero
    power <- 1                                                                  [1]
    IF y > 0 THEN                                                               [2]
        power <- potenza(x, y/2) //divisione intera                             [3]
        power <- power * power                                                  [4]
        IF y è dispari THEN                                                     [5]
            power <- power * x                                                  [6]
    RETURN power                                                                [7]
```

- Le linee $1, 2, 7$ vengono eseguite $1$ volta $\rightarrow T = 3$
- Le linee $3, 4, 5$ vengono eseguite $1$ volta $\rightarrow T = 3$
- La linea $6$ viene seguita $\leq 1$ volta $\rightarrow T \leq 1$
- Chiamata ricorsiva su linea $3 \rightarrow T(x, \lfloor \frac{y}{2} \rfloor)$

$$T(x,y) \leq \begin{cases} 3 & \text{se } y = 0 \\ 7 + T(x, \lfloor \frac{y}{2} \rfloor) & \text{altrimenti} \end{cases} \quad \longrightarrow \text{equazione di ricorrenza}$$

$$T(x,y) \leq 7 \log_2 y + 10$$

*Attenzione:*
`power <- potenza(x, y/2); power <- power * power` *e* `power <- potenza(x, y/2) * potenza(x, y/2)` *non sono la stessa cosa: l'equazione di ricorrenza cambia poiché si vanno a fare due chiamate ricorsive al posto di una.*

## Potenza alla russa

```
ALGORITMO potenza(intero x, intero y) -> intero
    power <- 1                                                                [1]
    WHILE y > 0 DO                                                            [2]
        IF y è dispari THEN                                                   [3]
            power <- power * x                                                [4]
        y <- y/2 //divisione intera                                           [5]
        x <- x*x                                                              [6]
    RETURN power                                                              [7]
```

### Correttezza

Siano $x_i, y_i, power_i$ i valori di $x, y, power$ dopo l'iterazione $i$. Si dimostri che $x_i^{y_i} \cdot power_i = x^y$

La dimostrazione si fa per induzione (simile a quello della moltiplicazione alla russa), ma intuitivamente al termine dell'esecuzione $y_i$ varrà $0$, perciò $x_i^{y_i}$ varrà $1$ e di conseguenza il valore finale di $power$ sarà proprio $x^y$

### Complessità

**Tempo:**
Si consideri $u$ il numero di iterazioni effettuate

- Le linee $1, 7$ vengono eseguite $1$ volta $\rightarrow T = 2$
- La linea $2$ viene ripetuta $u + 1$ volte $\rightarrow T = u + 1$
- Le linee $3, 5, 6$ vengono eseguite $u$ volte $\rightarrow T = 3u$
- La linea 4 viene eseguita al più $u$ volte $\rightarrow T \leq u$

$$T_{tot} \leq 5u + 3$$

Si ricava che $u = \lfloor \log_2 y \rfloor + 1$ *(rivedere moltiplicazione alla russa)*

$$T(x,y) \leq 5\ (\lfloor \log_2 y \rfloor + 1) + 3 = 5 \lfloor \log_2 y \rfloor + 8 = \mathcal{O}(\log y)$$

- La crescita del tempo non dipende da $x$ ed è logaritmica

**Spazio:** lo spazio è costate: non dipende da $x$ e da $y$ ma si utilizzano solo 3 variabili: $\mathcal{O}(1)$

## Formule Utili

**1. Formula di Gauss per la somma dei primi $n$ numeri interi:**

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

**2. Somma delle prime $k$ potenze di 2:**

$$\sum_{i=0}^{k-1} 2^i = 2^k - 1$$

**3. Operazioni sui numeri binari:**
$11010_2 = n \quad 110100_2 = 2n$ aggiungi uno zero a destra per raddoppiare un numero binario

**4. Problema**: Trovare il più piccolo $N$, potenza di 2, tale che $N \geq n$

1. Converto $n$ in binario
2. Raddoppio il numero (tramite uno shift a sinistra)
3. Imposto tutti i bit a zero, tranne quello più a sinistra

*Osservazione: Per ogni intero $n$, esiste una potenza di 2 $N$ tale che $n \leq N \leq 2n$*