
ALGORITMI E STRUTTURE DATI

LAUREA TRIENNALE IN SCIENZE INFORMATICHE

Magliani Andrea
Perego Luca

Università degli studi di Milano-Bicocca

A.A. 2022/2023

INDICE

Problema Computazionale e Algoritmi.....	4
1.1 Problema Computazionale.....	4
1.2 Istanza.....	4
1.3 Algoritmo.....	4
1.4 Analisi degli Algoritmi.....	4
1.5 Struttura dati.....	4
Correttezza & Efficienza.....	5
2.1 Dimostrazione di Correttezza.....	5
2.2 Calcolo dell'Efficienza.....	5
Notazioni Asintotiche.....	6
3.1 O-Grande.....	6
3.2 Ω -Grande.....	6
3.3 θ -Grande.....	6
3.4 Gerarchie di crescita Asintotica.....	6
Caratteristiche degli Algoritmi.....	7
4.1 Stabile.....	7
4.2 In-Place.....	7
Algoritmi di Ordinamento.....	7
5.1 Definizione.....	7
5.2 Struttura del problema.....	7
Teorema dell'esperto.....	8
6.1 Enunciato.....	8
Selection sort.....	9
7.1 Pseudocodice.....	9
7.2 Funzionamento.....	9
7.3 Correttezza.....	10
7.4 Tempi di calcolo.....	10
7.5 Caratteristiche.....	10
Insertion sort.....	11
8.1 Pseudocodice.....	11
8.2 Funzionamento.....	11
8.3 Correttezza.....	12
8.4 Tempi di calcolo.....	12
8.5 Caratteristiche.....	12
Mergesort.....	13
9.1 Pseudocodice.....	13
9.2 Funzionamento.....	14
9.3 Tempi di calcolo.....	14
Ricerca Dicotomica.....	15
10.1 Pseudocodice.....	15
10.2 Tempi di calcolo.....	15

Problema di selezione.....	16
11.1 Pseudocodice.....	16
11.2 Tempi di calcolo.....	16
Counting sort.....	17
12.1 Pseudocodice.....	17
12.2 Tempi di calcolo.....	17

Problema Computazionale e Algoritmi

1.1 Problema Computazionale

Relazione matematica tra input e output.

Un problema è definito come: $\pi \subseteq input \times output$

1.2 Istanza

Set di input specifici legati ad un determinato problema.

1.3 Algoritmo

Descrizione finita, composta da una sequenza di istruzioni elementari e non ambigue che, se eseguita, trasforma gli input in output.

1.4 Analisi degli Algoritmi

Gli algoritmi vengono **analizzati** per valutarne diversi aspetti:

- Correttezza: verificata con test e dimostrazioni;
- Efficienza: verificata misurando i tempi e lo spazio occupato;

Un algoritmo che risolve un problema per ogni sua istanza in un tempo finito è detto **corretto**.

Un algoritmo che, per almeno una delle istanze, non risolve correttamente il problema è detto **non corretto**.

1.5 Struttura dati

Un modo per memorizzare e manipolare dati.

Correttezza & Efficienza

2.1 Dimostrazione di Correttezza

Invariante di ciclo: metodo per dimostrare la correttezza di un algoritmo contenente un loop. L'invariante di ciclo si divide in 3 fasi:

- **Inizializzazione**: dimostra la correttezza per la prima iterazione;
- **Conservazione**: l'algoritmo è corretto per ogni valore di i e questa incrementa correttamente ad ogni iterazione;
- **Conclusione**: assumendo la condizione del ciclo False, l'algoritmo termina restituendo il risultato corretto;

2.2 Calcolo dell'Efficienza

Un algoritmo efficiente utilizza il minor **tempo** e **risorse** possibili. È necessario definire una funzione $T(n)$, ovvero il tempo di calcolo impiegato per gestire un input di lunghezza n .

Ad ogni tipo di istruzione viene assegnato un **valore temporale** di esecuzione, per poi contarne le occorrenze nel codice.

$T(n)$ equivale alla somma delle occorrenze di tutti i valori temporali.

In un algoritmo è presente:

Caso Migliore: $T_{migl}(n) \rightarrow$

Sottoinsieme delle istanze in cui l'algoritmo impiega meno.

Caso Peggior: $T_{pegg}(n) \rightarrow$

Sottoinsieme delle istanze in cui l'algoritmo impiega di più.

Notazioni Asintotiche

3.1 O-Grande

$O(n)$ rappresenta il **limite superiore** [asintoticamente] della funzione $T(n)$ di un algoritmo.

$$O(g(n)) = \{f(n) \mid \exists c, n_0 > 0, f(n) \leq c \cdot g(n) \ \forall n > n_0\} \quad f(n) \in N \wedge f(n) > 0 \text{ def.}$$

3.2 Ω -Grande

$\Omega(n)$ rappresenta il **limite inferiore** [asintoticamente] della funzione $T(n)$ di un algoritmo.

$$\Omega(g(n)) = \{f(n) \mid \exists c, n_0 > 0, 0 \leq c \cdot g(n) < f(n) \ \forall n > n_0\}$$

3.3 θ -Grande

$\theta(n)$ rappresenta la funzione che **delimita superiormente ed inferiormente** la funzione $T(n)$ di un algoritmo.

$$\theta(g(n)) = \{f(n) \mid \exists c_1, c_2, n_0 > 0, 0 \leq c_2 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n) \ \forall n > n_0\}$$

3.4 Gerarchie di crescita Asintotica

La crescita di $T(n)$ varia in base alla funzione a cui è associata.
La **scala di crescita** è:

$$c \rightarrow \log n \rightarrow \sqrt{n} \rightarrow n \rightarrow n \log n \rightarrow n^a [a > 1] \rightarrow a^n \rightarrow n! \rightarrow n^n$$

Caratteristiche degli Algoritmi

4.1 Stabile

Algoritmo che, se nel vettore di input sono presenti due valori **uguali**, mantiene l'ordine tra di loro anche nel vettore ordinato di output.

4.2 In-Place

Algoritmo che non utilizza una **struttura dati ausiliaria**, ma lavora direttamente sull'input.

Algoritmi di Ordinamento

5.1 Definizione

Gli algoritmi di ordinamento sono utilizzati per posizionare gli elementi di un insieme secondo una **relazione d'ordine**.

5.2 Struttura del problema

Ogni algoritmo di ordinamento condivide problema e risultato.

Problema: Ordinamento di un vettore V di n elementi.

Input: Un vettore V di n elementi.

Output: Un vettore V t.c.:

- L'output è una permutazione di V ;
- $\forall i \in [1, n - 1] \quad V[i] \leq V[i + 1]$;

Teorema dell'esperto

6.1 Enunciato

Sia $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ $a \geq 1, b > 1, f(n)$ asin. pos.

1. se $\exists \varepsilon > 0$ t.c. $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$ allora $T(n) = \Theta\left(n^{\log_b a}\right)$
2. se $f(n) = \Theta\left(n^{\log_b a}\right)$ allora $T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$
3. se $\exists \varepsilon > 0$ t.c. $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ e se $\exists c < 1$ t.c. $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$
 $\forall n > n_0$ allora $T(n) = \Theta(f(n))$

Selection sort

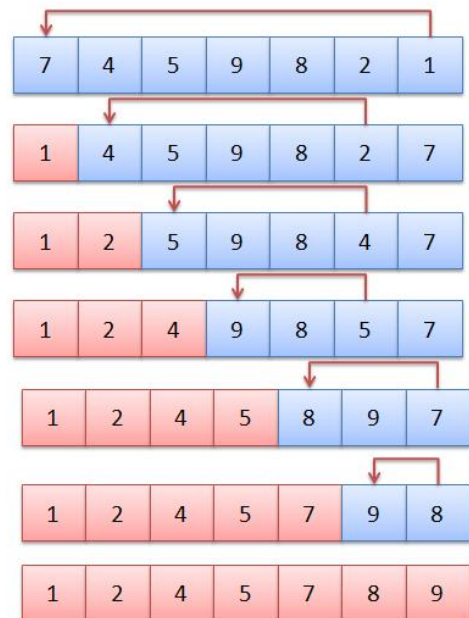
7.1 Pseudocodice

SELECTION_SORT (V)

```
for i := 1 to V.length - 1
    posmin := i
    for j := i + 1 to V.length - 1
        if V[posmin] > V[j] then
            posmin := j
    scambia V[i] con V[posmin]
```

7.2 Funzionamento

Si cerca il numero minore navigando tutto il vettore e si mette nella posizione i, con i che varia dalla prima posizione del vettore fino all'ultima.



7.3 Correttezza

I primi $i - 1$ elementi di V sono i più piccoli $i - 1$ elementi di V ordinati in ordine crescente.

INIZIALIZZAZIONE

I primi 0 elementi di V' sono i più piccoli 0 elementi di V ordinati in ordine crescente

CONSERVAZIONE

Corretto ad inizio e fine ciclo.

TERMINAZIONE

Corretto al termine dell'algoritmo.

7.4 Tempi di calcolo

CASO MIGLIORE e CASO PEGGIORE sono asintoticamente uguali.

$$T(n) = \theta(n^2)$$

7.5 Caratteristiche

STABILE

No, il selection sort non è un algoritmo di ordinamento stabile in quanto scambiando l'elemento di posizione i con l'elemento più piccolo dell'array, non sempre mantiene l'ordine originale degli elementi uguali nel vettore.

IN PLACE

Sì, il selection sort è un algoritmo di ordinamento in place in quanto non utilizza altre strutture dati per ordinare il vettore in input.

Insertion sort

8.1 Pseudocode

```
INSERTION_SORT (V)

  for i := 2 to V.length

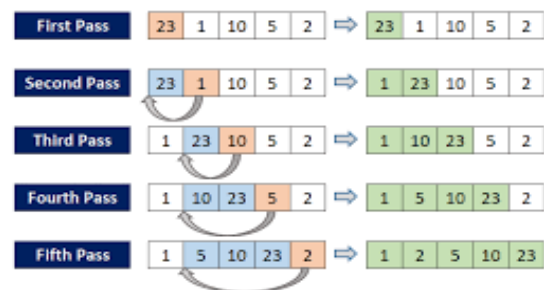
    j := i - 1
    key := V[i]

    while j >= 1 AND V[j] > Key

      V[j + 1] := V[j]
      V[j] := Key
      j := j - 1
```

8.2 Funzionamento

Si parte dal secondo elemento del vettore in input e si controlla se l'elemento precedente è minore, nel caso si scambiano i due valori, si continua successivamente con l'elemento $i + 1$ fino alla fine dell'array.



8.3 Correttezza

All'inizio di ogni iterazione i primi $i-1$ elementi di V^i sono i primi $i-1$ elementi di V in ordine crescente.

INIZIALIZZAZIONE

Il primo elemento di V^1 è il primo elemento di V in ordine crescente

CONSERVAZIONE

Vero ad inizio e fine ciclo

TERMINAZIONE

Vero a fine algoritmo

8.4 Tempi di calcolo

CASO MIGLIORE

Il vettore V è già ordinato. $T_{\text{migl}}(n) = \theta(n) \rightarrow T(n) = \Omega(n)$

CASO PEGGIORE

V è ordinato in senso decrescente. $T_{\text{pegg}}(n) = \theta(n^2) \rightarrow T(n) = O(n^2)$

8.5 Caratteristiche

STABILE

Si l'insertion sort è un algoritmo di ordinamento stabile in quanto mantiene l'ordine degli elementi uguali tra di loro.

IN PLACE

Si l'insertion sort è un algoritmo di ordinamento in place in quanto non utilizza strutture d'appoggio per eseguire le sue operazioni.

Mergesort

9.1 Pseudocodice

MERGESORT (V, l, r)

if $l < r$ then

$mid := \text{floor}[(l+r)/2]$
 MERGESORT (V, l, mid)
 MERGESORT (V, mid+1, r)
 MERGE (V, l, mid, r)

MERGE (V, l, mid, r)

 T := Vettore di lunghezza $r-l + 1$
 i := l
 j := mid + 1
 k := 1

while $i \leq mid$ AND $j \leq r$ do

 if $V[i] \leq V[j]$ then
 T[k] := V[i]
 i := i + 1

 else

 T[k] := V[j]
 j := j + 1

 k := k + 1

while $i \leq mid$ do

 T[k] := V[i]
 i := i + 1
 k := k + 1

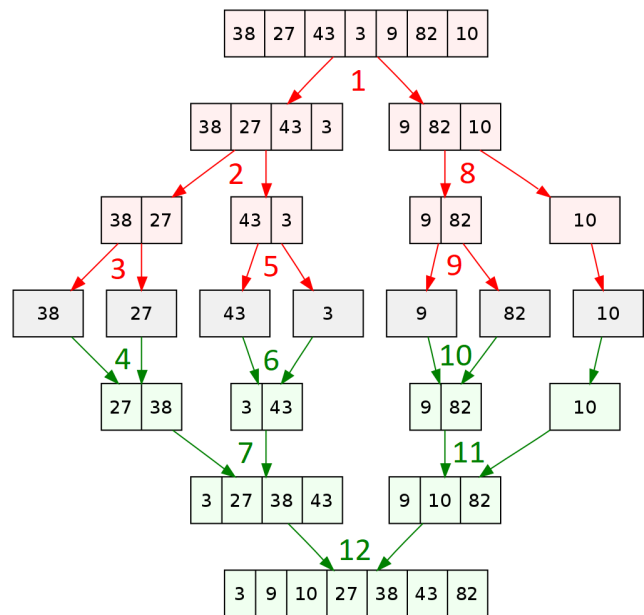
for k := 1 to T.length do

$V[l + k - 1] := T[k]$

9.2 Funzionamento

Il mergesort utilizza la strategia di programmazione **divide et impera** per ridurre il problema in più sottoproblemi.

Una volta ottenuti i singoletti e arrivati nel caso base si esegue la procedura di merge, andando a unire i singoletti riordinandoli durante il processo.



9.3 Tempi di calcolo

$a = 2$ (volte che viene chiamato il metodo ricorsivo)

$b = 2$ (in quante porzioni divido l'array [$mid = l+r / 2$])

$f(n) = \theta(n)$ (righe di codice che non c'entrano con la ricorsione)

quindi applico il secondo caso del teorema del maestro:

$$\theta(n) = \theta(n^{\log_2 2}) \rightarrow \theta(n) = \theta(n) \rightarrow T(n) = \theta(n^{\log_2 2} \cdot \log n) \rightarrow T(n) = \theta(n \cdot \log n)$$

Ricerca Dicotomica

Problema: Ricerca di un elemento in un vettore V ordinato.

Input: un vettore V di n elementi e un intero x .

Output: un intero n t.c. $n \in V \wedge n = x$

10.1 Pseudocodice

Ricerca_Dicotomica(V , x , l , r)

```
if  $r < l$  then  
    return false
```

```
if  $r = l$  then  
    return  $x == V[l]$ 
```

```
mid := floor( $\frac{l+r}{2}$ )
```

```
if  $x > V[mid]$  then  
    return Ricerca_Dicotomica( $V$ ,  $x$ , mid + 1,  $r$ )  
else  
    return Ricerca_Dicotomica( $V$ ,  $x$ ,  $l$ , mid)
```

10.2 Tempi di calcolo

Caso migliore e caso peggiore sono **asintoticamente uguali**: $T(n) = \Theta(\log_2 n)$.

Problema di selezione

Input: un vettore V di n interi distinti, un intero i t.c $1 \leq i \leq n$.

Output: Il valore di V che è maggiore di esattamente $i-1$ elementi di V .

11.1 Pseudocodice

SELEZIONE (V, i, l, r)

```
    if  $l = r$  then  
        return  $V[l]$ 
```

```
    cut := RANDOM_PARTITION ( $V, l, r$ )  
    dim_sx := cut -  $l + 1$ 
```

```
    if  $i \leq \text{dim\_sx}$  then  
        return SELEZIONE ( $V, i, l, \text{cut}$ )
```

```
    return SELEZIONE ( $V, i - \text{dim\_sx}, \text{cut} + 1, V$ )
```

11.2 Tempi di calcolo

Caso peggiore: $\theta(n^2)$

Caso migliore: $\theta(n)$

Si può notare che il caso peggiore è $\theta(n^2)$ quindi asintoticamente maggiore del caso peggiore di un algoritmo di ordinamento come il mergesort, che potremmo usare per riordinare il nostro vettore in input e risolvere facilmente il nostro problema di ricerca, quindi, perchè usare questo algoritmo e non ordinare l'array prima col mergesort?

Utilizzando random partition rendiamo l'algoritmo randomico e non più deterministico ed il suo tempo di calcolo diventa $\theta(n)$ in quanto non consideriamo il caso peggiore essendo un evenienza molto sfortunata e improbabile con l'approccio randomico, per cui l'algoritmo di selezione sviluppato è asintoticamente più veloce rispetto al mergesort, essendo $\theta(n)$ asintoticamente minore di $\theta(n \cdot \log n)$.

Counting sort

12.1 Pseudocode

COUNTING_SORT (A, k)

```
C := vettore di k elementi

for i := 1 to k
    C[i] := 0

for j := 1 to A.length
    C[A[j]] := C[A[j]] + 1

for i := 2 to k
    C[i] := C[i - 1] + C[i]

B := vettore con n elementi

for j := A.length down to 1
    B[C[A[j]]] := A[j]
    C[A[j]] := C[A[j]] - 1

return B
```

12.2 Tempi di calcolo

Primo for: $\theta(k)$

Secondo for: $\theta(n)$

Terzo for: $\theta(k)$

Quarto for: $\theta(n)$

$T(n, k) = \theta(n + k)$ con $k = O(n)$

$\rightarrow T(n) = \theta(n)$