

# Ordinamento (algoritmi quadratici)

March 12, 2020

**Obiettivi:** attraverso lo studio degli algoritmi di ordinamento quadratici, applicare la nozione di invariante e introdurre l'analisi di complessità.

**Argomenti:** problema dell'ordinamento, insertion-sort e selection-sort con analisi di correttezza e complessità.

## 1 Problema dell'ordinamento (sorting)

La ricerca in un vettore di  $n$  elementi richiede  $n$  confronti nel caso peggiore.

Se il vettore è ordinato, si può applicare la ricerca binaria (ricerca dicotomica) che richiede al più  $\log_2 n$  confronti:

```
BINSEARCH-RIC( $x, A, i, j$ )
  ▷ Pre:  $A[i..j]$  ordinato
  ▷ Post:  $true$  se  $x \in A[i..j]$ 
if  $i > j$  then    ▷  $A[i..j] = \emptyset$ 
  return  $false$ 
else
   $m \leftarrow \lfloor (i + j)/2 \rfloor$ 
  if  $x = A[m]$  then
    return  $true$ 
  else
    if  $x < A[m]$  then
      return BINSEARCH-RIC( $x, A, i, m - 1$ )
    else    ▷  $A[m] < x$ 
      return BINSEARCH-RIC( $x, A, m + 1, j$ )
    end if
  end if
end if
```

Per ordinare un vettore, si potrebbe pensare di generare tutte le permutazioni e scegliere quella nella quale gli elementi sono ordinati:

```
SORTED( $A[1..n]$ )
for  $i \leftarrow 2$  to  $n$  do
  if  $A[i - 1] > A[i]$  then
    return  $false$ 
  end if
end for
return  $true$ 
```

```

TRIVIAL-SORT( $A$ )
for all  $A'$  permutazione di  $A$  do
    if SORTED( $A'$ ) then
        return  $A'$ 
    end if
end for

```

Ma il numero di permutazioni è  $n!$  che cresce più velocemente di  $2^n$ . Non è praticabile.

## 2 Insertion-sort

L'idea generale: data un vettore con la parte sinistra,  $A[1..i-1]$ , ordinata, è facile inserirci l'elemento  $A[i]$  in modo tale che il sottovettore  $A[1..i]$  risulti ordinato (aumentando così la parte ordinata). Punto di partenza:  $i = 2$  (così la parte a sinistra di  $A[i]$  contiene un singolo elemento e quindi è ordinata).

```

0: INSERTION-SORT( $A[1..n]$ )
1: for  $i \leftarrow 2$  to  $n$  do
2:      $j \leftarrow i$ 
3:     while  $j > 1$  and  $A[j-1] > A[j]$  do
4:         scambia  $A[j-1]$  con  $A[j]$ 
5:          $j \leftarrow j - 1$ 
6:     end while
7: end for

```

Simulazione con  $A = (2, 6, 3, 1, 5, 4)$ . La parte ordinata del vettore viene scritta in grassetto. Il numero che deve essere inserito al posto giusto viene scritto in corsivo.

$i = 2, j = 2$  : (**2**,*6*,3,1,5,4)  
 $i = 3, j = 3$  : (**2**,**6**,*3*,1,5,4)  
 $i = 3, j = 2$  : (**2**,*3*,**6**,1,5,4)  
 $i = 4, j = 4$  : (**2**,**3**,**6**,*1*,5,4)  
 $i = 4, j = 3$  : (**2**,*3*,**1**,**6**,5,4)  
 $i = 4, j = 2$  : (**2**,*1*,**3**,**6**,5,4)  
 $i = 4, j = 1$  : (*1*,**2**,**3**,**6**,5,4)  
 $i = 5, j = 5$  : (**1**,**2**,**3**,**6**,*5*,4)  
 $i = 5, j = 4$  : (**1**,**2**,**3**,*5*,**6**,4)  
 $i = 6, j = 6$  : (**1**,**2**,**3**,**5**,**6**,*4*)  
 $i = 6, j = 5$  : (**1**,**2**,**3**,*4*,**5**,**6**)  
 $i = 6, j = 4$  : (**1**,**2**,**3**,*4*,**5**,**6**)  
 $i = 7$  : (**1**,**2**,**3**,**4**,**5**,**6**)

### 2.1 Dimostrazione della correttezza con invarianti

**Invariante del ciclo esterno:**

il sottovettore  $A[1..i-1]$  è ordinato.

Inizializzazione: con  $i = 2$  è vero (fa riferimento ad un vettore di un singolo elemento).

Mantenimento: assumendo che il ciclo interno inserisce al posto giusto  $A[i]$  (la dimostreremo di seguito) l'invariante viene mantenuta.

All'uscita: con  $i = n + 1$  l'invariante implica che il vettore è ordinato.

**Invariante del ciclo interno:**

i vettori  $A[1..j-1]$  e  $A[j..i]$  sono ordinati  $\wedge \forall l, 1 \leq l \leq j-1, \forall k, j+1 \leq k \leq i. A[l] \leq A[k]$

dove la seconda parte esprime il fatto che ciascun elemento in  $A[1..j-1]$  è minor uguale di qualunque elemento in  $A[j+1..i]$ .

Inizializzazione: grazie all'invariante del ciclo esterno e al valore iniziale di  $j = i$  l'invariante vale.

Mantenimento: come ipotesi induttiva assumiamo che l'invariante vale prima di eseguire il corpo del ciclo; ci sono due possibilità:

1. se  $A[j-1] \leq A[j] \vee j = 1$  allora si esce dal ciclo e all'uscita, grazie al fatto che  $A[j-1] \leq A[j] \leq A[j+1]$  (dove la parte destra c'è solo se  $j \neq i$  e la parte sinistra solo se  $j \neq 1$ ) e all'ipotesi induttiva, tutto il sotto vettore  $A[1..i]$  è ordinato,
2. se  $A[j-1] < A[j]$  allora si fa lo scambio fra  $A[j-1]$  e  $A[j]$  e l'invariante rimane valido

All'uscita: discusso al punto 1. sopra.

Quindi l'algoritmo INSERTION-SORT è corretto.

## 2.2 Tempo di esecuzione del INSERTION-SORT

Calcoliamo il tempo di esecuzione assumendo che eseguire la riga  $i$ ,  $1 \leq i \leq 5$ , una volta richiede  $c_i$  unità di tempo.

La riga 1 viene eseguita con  $i = 2, 3, \dots, n, n+1$ , cioè  $n$  volte (viene eseguita con  $i = n+1$  perché bisogna accorgersi di dover uscire dal ciclo).

La riga 2 viene eseguita con  $i = 2, 3, \dots, n$ , cioè  $n-1$  volte.

Dato un valore per  $i$ , denotiamo con  $t_i$  il numero di volte che la riga 3 viene eseguita. Nel caso migliore  $t_i = 1$  (non servono scambi per inserire  $A[i]$  al posto giusto) e nel caso peggiore  $t_i = i$  (bisogna portare  $A[i]$  all'inizio del vettore e quindi la condizione del while viene valutata con  $j = i, i-1, \dots, 2, 1$ ). In totale la riga 3 viene eseguita  $\sum_{i=2}^n t_i$  volte.

La riga 4 viene eseguita  $t_i - 1$  volta dato un valore per  $i$  (una volta in meno rispetto alla riga 3). In totale la riga 4 viene eseguita  $\sum_{i=2}^n (t_i - 1)$  volte.

La riga 5 viene eseguita  $t_i - 1$  volta dato un valore per  $i$  (una volta in meno rispetto alla riga 3). In totale la riga 5 viene eseguita  $\sum_{i=2}^n (t_i - 1)$  volte.

Considerando il caso peggiore ( $cp$ ) il tempo di esecuzione è

$$T_{cp}(n) = c_1 n + c_2(n-1) + c_3 \sum_{i=2}^n i + c_4 \sum_{i=2}^n (i-1) + c_5 \sum_{i=2}^n (i-1) =$$

$$(c_1 + c_2)n - c_2 + c_3 \frac{2+n}{2}(n-1) + (c_4 + c_5) \frac{1+n-1}{2}(n-1)$$

Segue che  $T_{cp}(n)$  è un polinomio di secondo grado in  $n$ , cioè può essere espresso come

$$T_{cp}(n) = an^2 + bn + c$$

Di conseguenza, per valori grandi di  $n$ ,

$$T_{cp}(n) \approx an^2$$

e quindi il **tempo di esecuzione nel caso peggiore è proporzionale al quadrato del numero di elementi** del vettore. Nel caso peggiore l'algoritmo è quadratico.

Considerando il caso migliore ( $cm$ ) il tempo di esecuzione è

$$T_{cm}(n) = c_1 n + c_2(n-1) + c_3 \sum_{i=2}^n 1 + c_4 \sum_{i=2}^n (1-1) + c_5 \sum_{i=2}^n (1-1) =$$

$$(c_1 + c_2)n - c_2 + c_3(n-1)$$

Segue che  $T_{cm}(n)$  è un polinomio di primo grado in  $n$ , cioè può essere espresso come

$$T_{cm}(n) = an + b$$

Di conseguenza, per valori grandi di  $n$ ,

$$T_{cm}(n) \approx an$$

e quindi il **tempo di esecuzione nel caso migliore è proporzionale al numero di elementi** del vettore. Nel caso migliore l'algoritmo è lineare.

### 3 Selection-sort

L'idea generale: data un vettore in cui la parte sinistra,  $A[1..i-1]$ , è ordinata e contiene gli  $i-1$  numeri più piccoli del vettore, si cerca il minimo della parte  $A[i..n]$  e si mette nella posizione  $i$  (aumentando così la parte ordinata). Punto di partenza:  $i = 1$  (così la parte a sinistra di  $A[i]$  è un vettore “vuoto”).

```
0: SELECTION-SORT( $A[1..n]$ )
1: for  $i \leftarrow 1$  to  $n-1$  do
2:    $k \leftarrow i$ 
3:   for  $j \leftarrow i+1$  to  $n$  do
4:     if  $A[k] > A[j]$  then
5:        $k \leftarrow j$ 
6:     end if
7:   end for
8:   scambia  $A[i]$  con  $A[k]$ 
9: end for
```

#### 3.1 Dimostrazione della correttezza con invarianti

**Invariante del ciclo esterno:**

il sottovettore  $A[1..i-1]$  è ordinato  $\wedge \forall k, l, 1 \leq k \leq i-1, i \leq l \leq n. A[k] \leq A[l]$

Inizializzazione: con  $i = 1$  la proposizione è “vuota” e quindi vale.

Mantenimento: assumendo che il ciclo interno selezioni il minimo del  $A[i..n]$  correttamente (la discuteremo di seguito) l'invariante viene mantenuta.

All'uscita: con  $n = 1$  l'invariante implica

il sottovettore  $A[1..n-1]$  è ordinato  $\wedge A[n-1] \leq A[n]$

e quindi il vettore è ordinato.

**Invariante del ciclo interno:**

$A[k]$  è il minimo in  $A[i..j-1]$

Inizializzazione: con  $k = i$  e  $j = i+1$  è triviale.

Mantenimento: come ipotesi induttiva assumiamo che l'invariante vale prima di eseguire il ciclo; il corpo del ciclo aggiorna la posizione del massimo se  $A[k] > A[j]$  e poi in ogni caso incrementa  $j$ ; quindi l'invariante viene mantenuta.

All'uscita: con  $j = n+1$  l'invariante implica che  $A[k]$  è il minimo in  $A[i..n]$  e quindi il ciclo interno funziona correttamente.

Quindi l'algoritmo SELECTION-SORT è corretto.

#### 3.2 Tempo di esecuzione del SELECTION-SORT

Deriviamo il tempo di esecuzione calcolando quante volte vengono eseguite le righe 1,2,3,4,5 e 8. Associamo con ognuna di queste righe un tempo di esecuzione uguale a 1 unità di tempo, cioè  $c_1 = c_2 = c_3 = c_4 = c_5 = c_8 = 1$ . Questa scelta non cambia in che modo il tempo di esecuzione dipende dal numero di elementi del vettore.

Nel caso peggiore l'indice del minimo in  $A[i..n]$  viene aggiornato ogni volta (cioè la condizione della riga 4 risulta sempre “true”):

$$\begin{aligned}
T_{cp}(n) &= \underbrace{n}_{\text{riga 1}} + \underbrace{n-1}_{\text{riga 2}} + \underbrace{\sum_{i=1}^{n-1} (n - (i+1) + 1 + 1)}_{\text{riga 3}} + \underbrace{\sum_{i=1}^{n-1} (n - (i+1) + 1)}_{\text{riga 4}} + \underbrace{\sum_{i=1}^{n-1} (n - (i+1) + 1)}_{\text{riga 5}} + \underbrace{n-1}_{\text{riga 8}} = \\
&\quad \underbrace{n}_{\text{riga 1}} + \underbrace{n-1}_{\text{riga 2}} + \underbrace{\sum_{i=1}^{n-1} (n - i + 1)}_{\text{riga 3}} + \underbrace{\sum_{i=1}^{n-1} (n - i)}_{\text{riga 4}} + \underbrace{\sum_{i=1}^{n-1} (n - i)}_{\text{riga 5}} + \underbrace{n-1}_{\text{riga 8}} = \\
&\quad \underbrace{n}_{\text{riga 1}} + \underbrace{n-1}_{\text{riga 2}} + \underbrace{\sum_{i=1}^{n-1} (i+1)}_{\text{riga 3}} + \underbrace{\sum_{i=1}^{n-1} i}_{\text{riga 4}} + \underbrace{\sum_{i=1}^{n-1} i}_{\text{riga 5}} + \underbrace{n-1}_{\text{riga 8}}
\end{aligned}$$

La precedente è un polinomio di secondo grado in  $n$ . Per grandi valori di  $n$ , il tempo di esecuzione è proporzionale al quadrato del numero di elementi. Quindi **nel caso peggiore l'algoritmo è quadratico**.

Nel caso migliore il termine associato con la riga 5 è 0 (non viene mai aggiornato la posizione del minimo). Nonostante ciò, il tempo di esecuzione,  $T_{cm}(n)$ , è un polinomio di secondo grado in  $n$ . Quindi anche **nel caso migliore l'algoritmo è quadratico**.