

# Vettori (array)

- Un vettore è una struttura dati che permette di memorizzare sequenze di dati omogenei (sequenze di interi, di valori booleani,...)
- I vettori sono caratterizzati da
  - dimensione
  - tipo
- Es.  $v[10]$   $v =$ 

3	5	7	9	0	0	2	1	3	4
0	1	2	3	4	5	6	7	8	9
- I singoli elementi di un vettore possono essere riferiti mediante un indice.
- notazione:  $v[i]$ ="i-esimo elemento del vettore  $v$ "

# Operazioni sui vettori

- Possiamo scrivere e leggere una singola cella di un vettore.

Esempi di istruzioni di input/output su vettori

- leggi  $v[i]$
- scrivi  $v[i]$

Esempi di istruzioni di assegnamento su vettori

- $v[i] \leftarrow \text{exp}$
- $x \leftarrow v[i]$
- $w[j] \leftarrow v[i]$

# Alcuni esercizi sui vettori

**Problema:** Dato un vettore di interi  $w$  di dimensione  $m$ , calcolare l'elemento minimo di  $w$ .

```
sottoprogramma minvet(in:w,m;out: min)
{
  1. min←w[0]
  2. i←1
  3. mentre (i<m) fai{
    3.1. se w[i]<min allora
      3.1.1. min ← w[i]
    3.2  i←i+1
  }
}
```

**Problema:** dati due vettori di interi  $v1$  e  $v2$  di dimensioni  $n1$  e  $n2$ , verificare se  $v1$  e  $v2$  sono uguali.

```
sottoprogramma vett=(in:v1,n1,v2,n2: out: uguali)
{
    1. se ( $n1 \neq n2$ ) allora
        1.1 uguali ← falso
    altrimenti
        {
            1.2 uguali ← vero
            1.3  $i \leftarrow 0$ 
            1.4 mentre ( $i < n1$ ) e ( $uguali = vero$ ) fai{
                1.4.1 se  $v1[i] \neq v2[i]$  allora
                    2.4.1.1 uguali ← falso
                1.4.2  $i \leftarrow i + 1$ 
            }
        }
}
```

**Problema:** Dato un vettore di caratteri  $v$  di dimensione  $n$ , verificare se la parola o frase contenuta è palindroma.

**Esempi di palindrome:**

“Onorarono”

“Avida di vita, desiai ogni amore vero, ma ingoiai sedativi da diva”

“O mordo tua nuora o aro un autodromo”

“Was it a cat I saw?” (L. Carroll - Alice; inoltre lo dice Titti quando vede Gatto Silvestro)

# Soluzione

```
sottoprogramma palindroma?(in:v,n:out:pal)
{
  1. I←0
  2. J←n-1
  3. pal←vero
  4. mentre (I<n/2) e (pal=vero) fai{
      4.1. se (v[I]<>v[J]) allora
          4.1.2. pal←falso
      4.2. I←I+1
      4.3. J←J-1
  }
}
```

# Algoritmi di ricerca

- Problema: dato un vettore  $v$  di dimensione  $n$  e un elemento  $X$ , determinare se  $X$  è contenuto in  $v$ .
- Soluzioni possibili:
  - ricerca lineare
  - ricerca binaria

# Ricerca lineare

- Soluzione: scorrere tutto il vettore  $v$  da posizione 0 a posizione  $(n-1)$  e controllare ogni singolo elemento. Fermarsi nel caso in cui  $v[i]=X$ , per qualche  $i=1,\dots,n$ .

# L' algoritmo

```
sottoprogramma ricerca-lineare (in:v,n,x;out: trovato)
{
  trovato ← falso
  i ← 0
  mentre ((i<n) and (not trovato)) fai
  {
    se (v[i]=X) allora trovato ← vero
    i ← i+1
  }
}
```

- Nel caso pessimo (elemento non trovato):  
dovrò scorrere tutta la lista di elementi

# Indovina chi?



**Gioco:** Un giocatore A sceglie una persona in un'aula di 80 persone. Un giocatore B deve indovinare di chi si tratta ponendo ad A al massimo 7 domande.

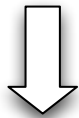
# Ricerca binaria

- Nel caso in cui il vettore fosse ordinato possiamo utilizzare un metodo piu' efficiente per ricercare un elemento in un vettore.
- Metodo
  - Determino il valore  $v[m]$  che sta nella posizione centrale del vettore
  - Se  $v[m]=x$ , allora ho trovato l'elemento cercato, altrimenti...
  - Se  $x < v[m]$ , cerco l'elemento nella prima metà del vettore, altrimenti lo cerco nella seconda metà.
- Il metodo mi permette sempre di scartare metà degli elementi senza doverli leggere.

# Esempio

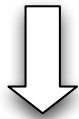
Cerco il valore 7

3	4	5	7	9	10	12	13	14	24
---	---	---	---	---	----	----	----	----	----



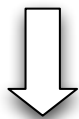
$7 < 9$

3	4	5	7
---	---	---	---



$7 > 4$

5	7
---	---



$7 > 5$

<b>7</b>
----------

# L' algoritmo

```
sottoprogramma RicercaBinaria(in:v,n,x;out:trovato)
{
  trovato ← falso
  max ← n-1
  min ← 0
  mentre (min <= max) and (not trovato) fai
  {
    m ← (max+min)/2
    se (x = v[m]) allora
      trovato ← vero
    altrimenti
      se (x < v[m]) allora
        max ← m-1
      altrimenti
        min ← m+1
  }
}
```

# Ordinamento

- Problema dato un vettore  $v$  di lunghezza  $n$  contenente una sequenza disordinata di interi, ordinare la sequenza in senso crescente (o decrescente).
- Esempio

input 

3	5	7	9	0	0	2	1	3	4
---	---	---	---	---	---	---	---	---	---

output 

0	0	1	2	3	3	4	5	7	9
---	---	---	---	---	---	---	---	---	---

# Bubble sort

- È l'algorithmo di ordinamento più vecchio e anche il più lento!
- Metodo: si scandisce il vettore confrontando elementi adiacenti, nel caso sia necessario si scambiano gli elementi. L'algorithmo ripete tale operazione finché non ci sono più posizioni da scambiare (vettore ordinato).
- Durante l'esecuzione dell'algorithmo gli elementi più grandi (più piccoli, nel caso di ordinamento decrescente) tendono a spostarsi (come bolle!) verso la fine del vettore.

# Esempio

Ordiniamo con bubble sort il vettore

3 2 0 1

3 2 0 1

2 0 1 3

0 1 2 3

0 1 2 3

2 3 0 1

0 2 1 3

0 1 2 3

2 0 3 1

0 1 2 3

2 0 1 3

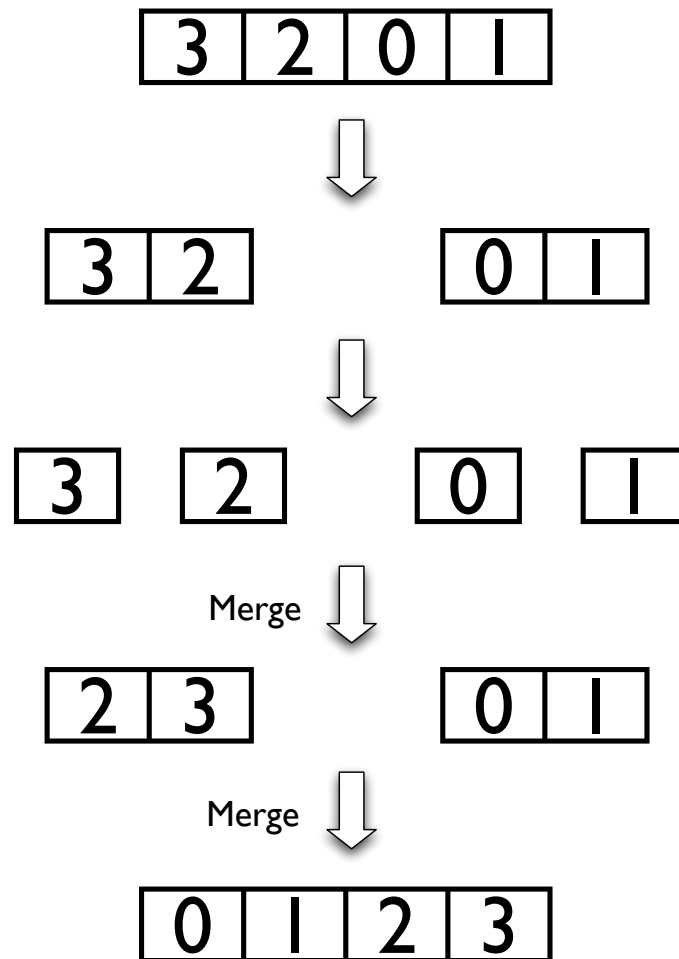
# L' algoritmo

```
sottoprogramma bubblesort(in:v,n;out:v)
{
  i ← 0
  mentre (i < n-1) fai
  {
    j ← 0
    mentre (j < n-i-1) fai
    {
      se (v[j+1] > v[j]) allora
        scambia(in:v[j],v[j+1];out:v[j],v[j+1]);
      j ← j+1
    }
    i ← i+1
  }
}
```

# Merge sort

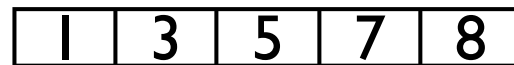
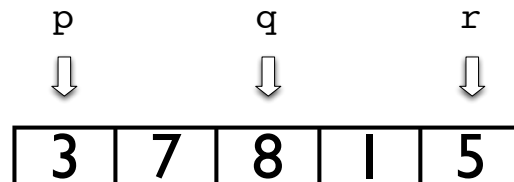
- È un algoritmo di ordinamento ricorsivo basato sull'approccio "divide and conquer"
  - "dividi" il problema in sottoproblemi
  - "conquista" i sottoproblemi, risolvendoli ricorsivamente
  - combina le soluzioni dei sottoproblemi per ottenere la soluzione del problema originale
- Algoritmo di merge sort
  - DIVIDE: dividi il vettore di  $n$  elementi in due sottovettori di  $n/2$  elementi
  - CONQUER: ordina i due sottovettori
  - COMBINE: fonda assieme i due sottovettori ordinati in un unico vettore ordinato

# Esempio



# Merge

- Problema: fondere assieme due sottovettori ( $v[p\dots q]$ ,  $v[q+1\dots r]$ ) ordinati ottenendo un unico vettore ordinato  $v$ .



# Merge - l'algoritmo

```
sottoprogramma merge(in:v,p,q,r: out v)
{
  i ← p
  j ← q+1
  k ← 0
  mentre (i≤q) and (j≤r) fai
    se v[i]≤v[j] allora {
      aux[k] ← v[i]
      k ← k+1
      i ← i+1
    }
    altrimenti {
      aux[k] ← v[j]
      k ← k+1
      j ← j+1
    }
  mentre (i≤q) fai {
    aux[k] ← v[i]
    k ← k+1
    i ← i+1
  }
  mentre (j≤r) fai {
    aux[k] ← v[j]
    k ← k+1
    j ← j+1
  }
  copia(in aux; out v)
}
```

# Merge sort l'algorithmo

```
sottoprogramma mergesort(in: v,p,r;out: v)
  se (p<r) allora
  {
    q ← (p+r/2)
    mergesort(in: v,p,q; out: v)
    mergesort(in: v,q+1,r; out v)
    merge(in:v,p,q,r)
  }
```

# Matrici

Una **matrice** è un vettore multidimensionale, in cui gli elementi memorizzati sono riferiti attraverso delle n-uple di indici. Ci limiteremo ad usare matrici di dimensione 2.

Nel caso di dimensione 2, una matrice può essere interpretata come una griglia di valori (vettore di vettori). Ogni elemento di una matrice di dimensione 2 è indicizzato mediante una coppia di indici (numero di riga, numero di colonna).

	0	1	2
0	5	1	0
1	1	3	7
2	10	9	8

elemento 7 identificato dalla coppia di indici (1,2)

# Notazione matrici

L'elemento di riga  $I$  e colonna  $J$  di una matrice di nome  $M$  è identificato mediante la notazione

**$M[I][J]$**

**Es.**  $M[0][0]$  elemento di riga 0 e colonna 0 di  $M$  (i.e. elemento corrispondente alla cella nell'angolo superiore sx della matrice)