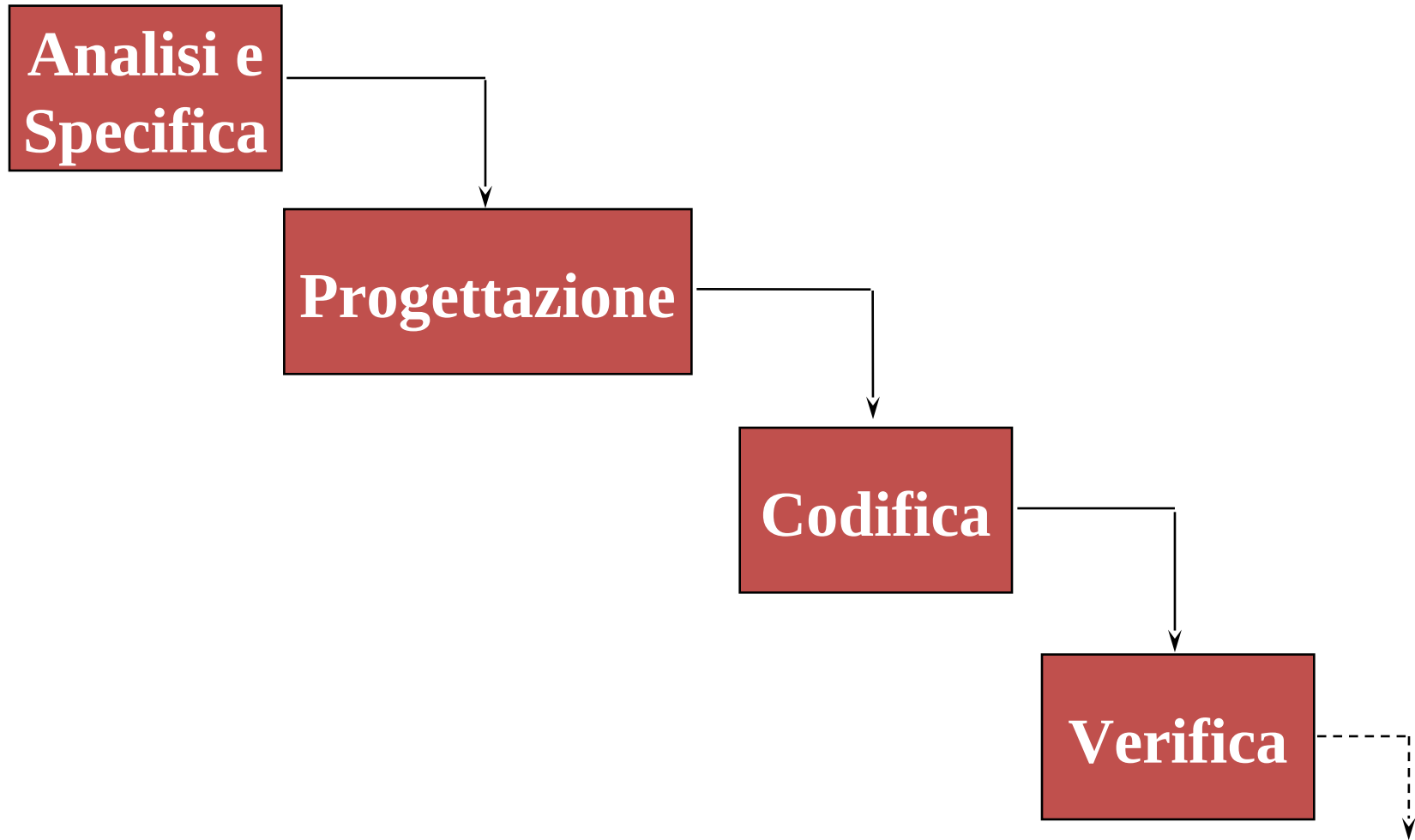


**Sviluppo di Programmi**

**Analisi e Progettazione**

# Sviluppo dei programmi



# Analisi e Specifica

- Definizione di *cosa fa* il programma
  - Individuazione dei *dati di ingresso* e di *uscita*, della *precondizione* e della *postcondizione*
  - Se la *precondizione* è vera e il programma è eseguito, allora deve valere la *postcondizione*
- **Precondizione**: condizione definita sui dati di ingresso che deve essere soddisfatta affinché la funzione sia applicabile
- **Postcondizione**: condizione definita su dati di uscita e dati di ingresso e che deve essere soddisfatta al termine dell'esecuzione del programma
  - definisce i dati di output in funzione di quelli di input

# Dizionario dei Dati

- Buona norma utilizzare un *dizionario dei dati* da arricchire durante le varie fasi del ciclo di vita
  - Una tabella il cui schema è:
    - *Identificatore, Tipo, Descrizione*
    - La descrizione serve a specificare meglio l'identificatore e a descrivere il contesto in cui il dato viene usato

# Un esempio: ordinamento di una sequenza di interi

- Dati di ingresso: sequenza  $s$  di  $n$  interi
- Precondizione:  $n > 0$
- Dati di uscita: sequenza  $s1$  di  $n$  interi
- Postcondizione:  $s1$  è una permutazione di  $s$  dove  
 $\forall i \in [0, n-2], s1_i \leq s1_{i+1}$

| <i>Identificatore</i> | <i>Tipo</i> | <i>Descrizione</i>                                 |
|-----------------------|-------------|--|
| $s$                   | sequenza    | sequenza di interi in input                        |
| $s1$                  | sequenza    | sequenza di interi di output                       |
| $n$                   | intero      | numero di elementi nella sequenza                  |
| $i$                   | intero      | indice per individuare gli elementi nella sequenza |

# Progettazione

- Definizione di *come* il programma effettua la trasformazione specificata
- Progettazione dell'algoritmo per raffinamenti successivi (stepwise refinement)
- Decomposizione funzionale

# Codifica e Verifica

- Codifica dell'algoritmo nel linguaggio scelto
- Verifica (testing) del programma (individuazione dei malfunzionamenti)
  - **Scelta dei casi di prova**
  - **Esecuzione del programma**
  - **Verifica dei risultati rispetto ai *risultati attesi***
- *Utilizzo del software di base e di un ambiente di sviluppo*

# Progettazione: ordinamento di una sequenza di interi

1. Input sequenza  $s$  in un array  $a$  di dimensione  $n$
2. Ordina array  $a$  di dimensione  $n$   
NB: per motivi di efficienza decidiamo di usare un unico array di input e output
3. Output sequenza  $s_1$  contenuta in array  $a$  di dimensione  $n$

**Raffiniamo i passi 1, 2 e 3 con delle nuove funzioni:**

- `input_array(a, n)`
- `ordina_array(a, n)`
- `output_array(a, n)`

**Per ognuna: specifica, progettazione, codifica e verifica**



# Funzione ordina\_array

1. Specifica simile a quella del programma principale, ma introduciamo l'array
2. Progettazione: scegliamo come strategia di ordinamento Selection Sort
3. Codifica e verifica

# Specifica: funzione ordina\_array

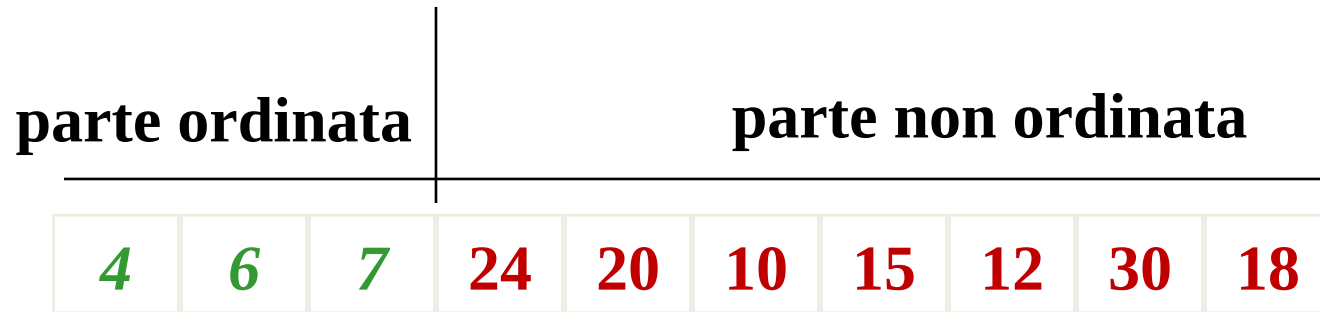
- Dati di ingresso: array a di interi di dimensione n
- Precondizione:  $n > 0$
- Dati di uscita: array a di interi di dimensione n
- Postcondizione: l'array a in output contiene una permutazione degli elementi dell'array a in input AND  
 $\forall i \in [0, n-2], a[i] \leq a[i+1]$

| Identificatore | Tipo   | Descrizione                                    |
|----------------|--------|--|
| a              | array  | Array di interi                                |
| n              | intero | Dimensione dell'array                          |
| i              | intero | Indice per individuare gli elementi dell'array |

# Selection Sort

- Effettua una visita totale delle posizioni dell'array
  - **visita totale:** visitati in sequenza tutti gli elementi dell'array
- Per ogni posizione visitata individua l'elemento che dovrebbe occupare quella posizione nell'array ordinato e scambia l'elemento trovato con quello che occupa attualmente la posizione
  - in questo modo, se  $i$  è la posizione corrente ( $0 \leq i < n$ ), tutti gli elementi nelle posizioni comprese tra 0 ed  $i-1$  rispettano l'ordinamento;
  - quindi l'elemento che deve occupare la posizione  $i$  sarà il minimo tra quelli nelle posizioni comprese tra  $i$  ed  $n-1$ ;
  - da notare che alla fine l'ultimo elemento (posizione  $n-1$ ) risulta ordinato

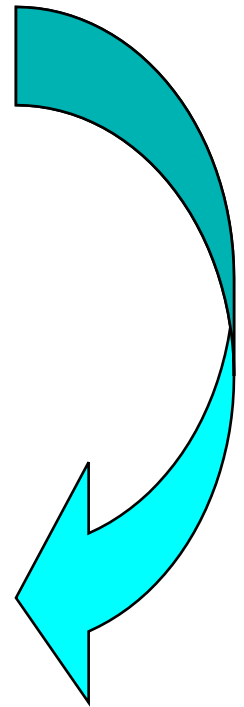
# Selection Sort



posizione corrente

nuovo minimo

scambia



# Selection sort: Algoritmo

- **for( $i = 0$ ;  $i < n-1$ ;  $i++$ )**
  - 1. Individua la posizione  $p$  dell'elemento minimo compreso tra le posizioni  $i$  e  $n-1$  dell'array  $a$**
  - 2. Scambia gli elementi di  $a$  di posizioni  $i$  e  $p$**

**Raffiniamo il passo 1 e con una nuova funzione minimo\_i ...**

**Raffiniamo il passo 2 con una funzione scambia ...**

# Individua la posizione del minimo: funzione minimo\_i

- **Specifica**

- Dati di ingresso: array  $a$  di dimensione  $n$ , posizione  $i$
- Precondizione:  $n > 0$  AND  $i \geq 0$  AND  $i < n$
- Dati di uscita: posizione  $pmin$
- Postcondizione:  $\forall j \in [i, n-1], a[pmin] \leq a[j]$

- **Progettazione**

```
min = a[i]; pmin = i;  
for(j = i+1; j < n; j++)  
    if(a[j] < min) { min = a[j]; pmin = j; }  
return(pmin);
```

# Codifica

```
void ordina_array(int a[], int n)
{ int i, p;
  for (i = 0; i < n-1; i++) {
    p = minimo_i(a, i, n);
    scambia(&a[i], &a[p]); }
}
```

```
int minimo_i(int a[], int i, int n)
{ int min, pmin, j;
  min = a[i]; pmin = i;
  for (j = i+1; j < n; j++)
    if (a[j] < min) {
      min = a[j];
      pmin = j; }
  return(pmin);
}
```

```
void scambia(int * x, int * y)
{ int temp = *x;
  *x = *y;
  *y = temp;
}
```

# Richiami: Puntatori

- Un **puntatore** è un dato che rappresenta l'indirizzo di un altro dato
- I puntatori sono “**type bound**” cioè ad ogni puntatore è associato il tipo a cui il puntatore si riferisce
- Nella dichiarazione di un puntatore bisogna specificare un asterisco (\*) prima del nome della variabile pointer:  
 $T^*p$
- Esempio:
  - `int *pointer;`                      `// puntatore a intero`
  - `char *pun_car;`                      `// puntatore a carattere`
  - `float *flt_pnt;`                      `// puntatore a float`



# Richiami: Dereferenziazione

- L'accesso all'oggetto puntato avviene attraverso l'operatore di dereferenziazione `*`
  - `*pointer = 5`                      `/* assegna all'oggetto puntato da pointer il valore 5 */`
  - `x = *pointer`                      `/* assegna il valore dell'oggetto puntato da pointer alla variabile x */`
- Prima di poter usare un puntatore, questo deve essere inizializzato, ovvero deve contenere l'indirizzo di un oggetto

# Richiami: Operatore di indirizzo

- Per ottenere l'indirizzo di un oggetto si usa l'operatore unario **&**.

```
int volume, *vol_ptr;  
vol_ptr = &volume;
```

```
int i = 10, *p1, *p2;  
p1 = &i;  
printf("%d \n", *p1);
```

# Codice delle funzioni di I/O

```
void input_array(int a[], int n)
{
    int i;
    for(i = 0; i < n; i++) {
        printf( "Elemento di posizione %d :", i);
        scanf("%d", &a[i]); }
}
```

```
void output_array(int a[], int n)
{
    int i;
    for(i = 0; i < n; i++)
        printf( "Elemento di posizione %d : %d \n", i, a[i]);
}
```

# Il main

```
# include <stdio.h>
# define MAXELEM 100

int main()
{
    int a[MAXELEM], n;

    printf( "Inserisci il numero di elementi da ordinare: ");
    scanf( "%d", &n);

    if (n <= 0)
        printf("Il numero di elementi deve essere positivo \n");
    else if (n > MAXELEM)
        printf(il numero massimo di elementi è MAXELEM \n");
    else {
        input_array(a, n);
        ordina_array(a, n);
        output_array(a, n); }
}
```

# Esercizi

- Realizzare le seguenti funzioni
  - Inserimento di un elemento in una data posizione in un array
  - Eliminazione di un elemento in una data posizione in un array
  - Ricerca del minimo in un array
  - Ricerca di un elemento in un array non ordinato
  - Ricerca di un elemento in un array ordinato (sia versione con ricerca lineare che con ricerca binaria)
  - ...
- Realizzare funzioni di ordinamento che utilizzano gli algoritmi insertion sort e bubble sort
  - Li vediamo nelle prossime slide ...

# Esercizi

- Ricerca del minimo in un array (non vuoto)

Soluzione ricorsiva

```
int minA (int A[], int n)
{
    int min;
    if (n==1) return A[0];
    min = minA(A, n-1);
    if (min<A[n-1])
        return min;
    else return A[n-1];
}
```

# Esercizi

- Ricerca di un elemento in un array (non vuoto)

Soluzione ricorsiva

```
int cercaA (int A[], int val, int n)
{
    if (val == A[n-1]) return 1;
    if (n == 1) return 0;
    return cercaA (A, val, n-1);
}
```

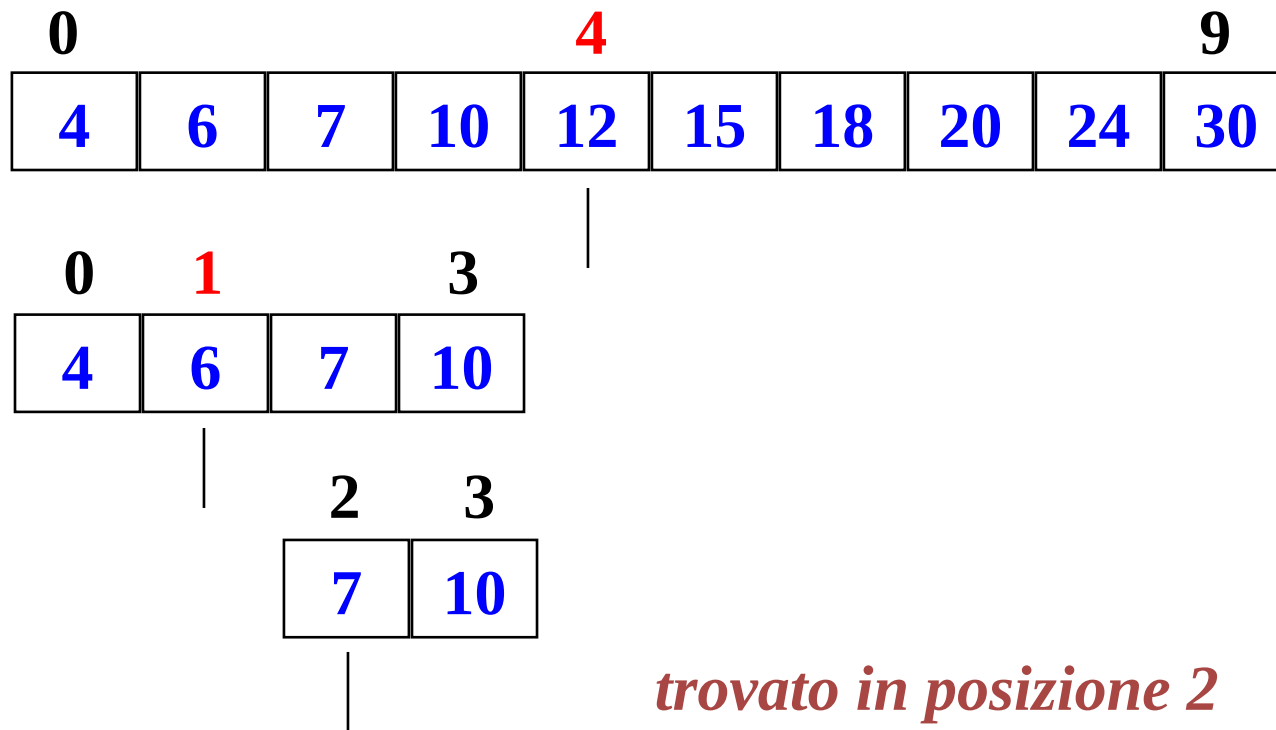
# Ricerca binaria in un array ordinato

- Consiste nel dividere l'array in due metà e confrontare l'elemento da cercare con l'elemento centrale dell'array
  - uguali --> trovato (... e ci si ferma)
  - elemento dell'array maggiore --> continuare la ricerca nella prima metà dell'array
  - elemento dell'array minore --> continuare la ricerca nella seconda metà dell'array
- Se l'elemento non è presente, l'array si ridurrà ad un solo elemento, non divisibile in due (terminazione)
  - nel caso peggiore si visitano  $\log_2 n$  elementi dell'array



# Esempio

- Cercare l'elemento 7 nell'array ...



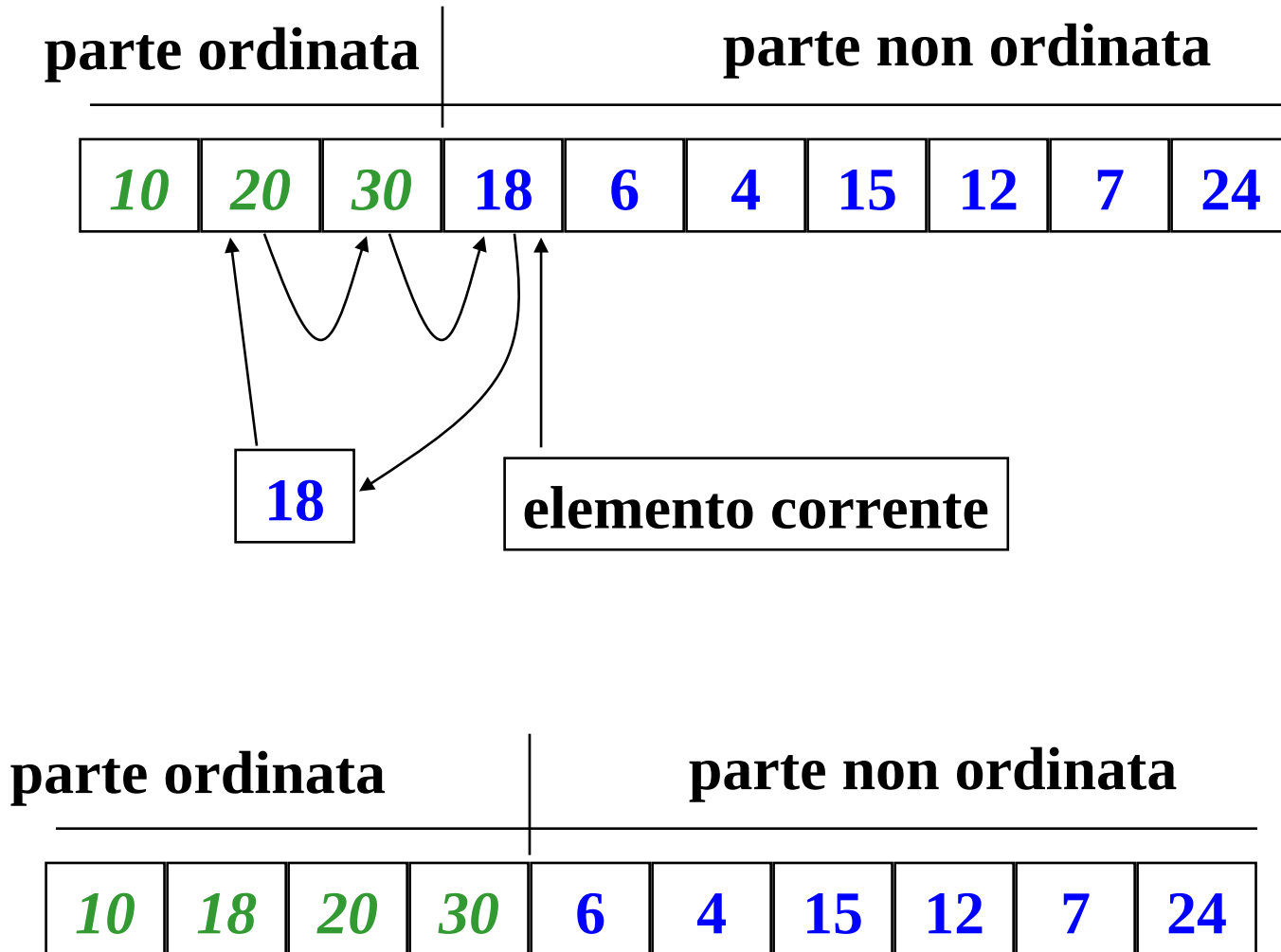
*... e se invece si fosse cercato 8 ? ...*

# Altri algoritmi di Ordinamento:

## Insertion Sort

- Visita totale: ad ogni passo gli elementi che precedono l'elemento corrente sono ordinati
  - si inserisce l'elemento corrente nella posizione che garantisce il mantenimento dell'ordinamento
  - gli elementi precedenti maggiori sono spostati in avanti
  - il primo elemento è già ordinato

# Insertion Sort



# Algoritmo di Insertion Sort

for( $i = 1; i < n; i++$ )

memorizza l'elemento di posizione  $i$  in una  
variabile temporanea  $next$

sia  $j$  la posizione in cui deve essere inserito  $next$ :  
sposta in avanti gli elementi di posizioni tra  $i-1$  e  
 $j$ ,

inserisci  $next$  in posizione  $j$

# Altri algoritmi di Ordinamento:

## Bubble Sort

- Algoritmo iterativo:
  - finché l'array non risulta ordinato si effettua una visita durante la quale si scambiano gli elementi adiacenti che non risultano ordinati
  - se in una iterazione non è stato effettuato nessuno scambio allora l'array è ordinato
- NB: ad ogni passo l'elemento più grande viene portato nella sua posizione finale ...
  - dopo il passo  $i$ -esimo, gli elementi tra le posizioni  $n-i$  ed  $n-1$  risultano ordinati e nelle loro posizioni finali
  - l'algoritmo converge in al più  $n-1$  iterazioni (dove  $n$  è il numero di elementi dell'array)

# Bubble Sort

*1<sup>a</sup> iterazione*

|    |    |   |    |   |   |
|----|----|---|----|---|---|
| 10 | 20 | 7 | 18 | 6 | 4 |
|----|----|---|----|---|---|



|    |   |    |    |   |   |
|----|---|----|----|---|---|
| 10 | 7 | 20 | 18 | 6 | 4 |
|----|---|----|----|---|---|



|    |   |    |    |   |   |
|----|---|----|----|---|---|
| 10 | 7 | 18 | 20 | 6 | 4 |
|----|---|----|----|---|---|



|    |   |    |   |    |   |
|----|---|----|---|----|---|
| 10 | 7 | 18 | 6 | 20 | 4 |
|----|---|----|---|----|---|



|    |   |    |   |   |    |
|----|---|----|---|---|----|
| 10 | 7 | 18 | 6 | 4 | 20 |
|----|---|----|---|---|----|

*2<sup>a</sup> iterazione*

|    |   |    |   |   |    |
|----|---|----|---|---|----|
| 10 | 7 | 18 | 6 | 4 | 20 |
|----|---|----|---|---|----|



|   |    |    |   |   |    |
|---|----|----|---|---|----|
| 7 | 10 | 18 | 6 | 4 | 20 |
|---|----|----|---|---|----|



|   |    |   |    |   |    |
|---|----|---|----|---|----|
| 7 | 10 | 6 | 18 | 4 | 20 |
|---|----|---|----|---|----|



|   |    |   |   |    |    |
|---|----|---|---|----|----|
| 7 | 10 | 6 | 4 | 18 | 20 |
|---|----|---|---|----|----|

• • • • •

# Algoritmo di Bubble sort

boolean ordinato = false;

i = 1;

while (i < n && ! ordinato)

ordinato = true;

*scambia gli elementi adiacenti che non risultano ordinati tra le posizioni 0 e n-i e poni ordinato a false se viene effettuato almeno uno scambio*

i = i + 1;