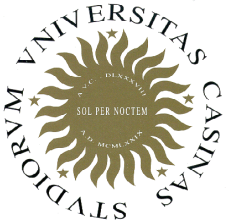



**Università degli Studi di Cassino**

# Corso di Fondamenti di Informatica

## *Algoritmi su array / 2*

Anno Accademico 2011/2012

Francesco Tortorella

# Algoritmi su array 2

- Operazioni “tipiche” sugli array:
  - ✓ inizializzazione
  - ✓ lettura
  - ✓ stampa
  - ✓ ricerca del minimo e del massimo
  - ✓ ricerca di un valore
  - inserimento di un valore
  - eliminazione di un valore
  - ordinamento del vettore

# Inserimento ed eliminazione

- A volte è utile considerare gli array come insiemi dinamici di dati [la cui composizione, cioè, può variare dinamicamente].
- A questo scopo è necessario realizzare operazioni quali:
  - Inserimento di un valore nell'array
  - Eliminazione di un elemento dall'array

# Inserimento di un valore nell'array

- Per poter inserire un nuovo valore nell'array, è necessario che questo disponga di spazio sufficiente per accogliere il nuovo elemento
- L'inserimento va realizzato diversamente per
  - Array non ordinato
  - Array ordinato

# Inserimento in array non ordinato

- Se l'array non è ordinato, non esiste una posizione precisa in cui il valore va inserito
- La scelta più conveniente è di inserire il nuovo valore in coda all'array

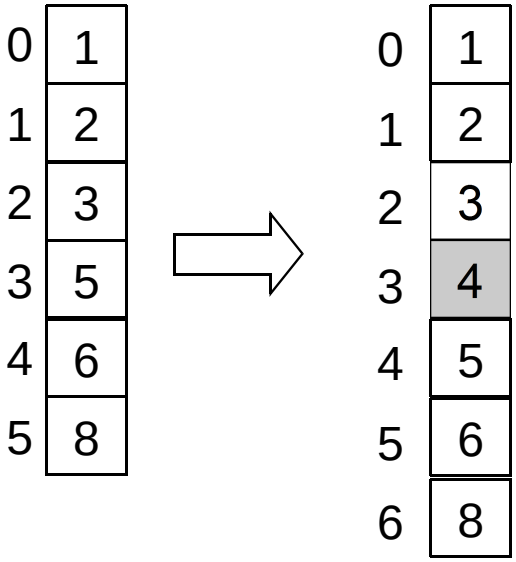
# Inserimento in array ordinato

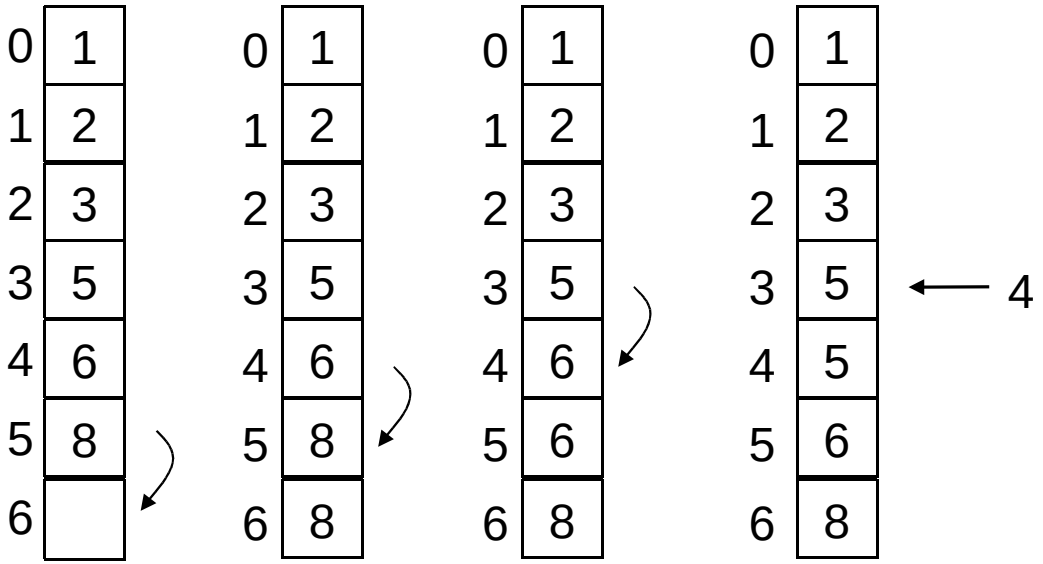
- L'inserimento di un valore in un array ordinato deve mantenere l'ordine degli elementi.
- Per questo motivo, non è possibile il semplice accodamento del nuovo valore a quelli già presenti, ma è necessario modificare la disposizione degli elementi già presenti per fare in modo che il nuovo valore possa essere inserito nella giusta posizione.

# Inserimento in array ordinato

- L'inserimento si realizza in due passi:
  - individuare la posizione che deve assumere il nuovo valore all'interno dell'array per mantenere l'ordine dell'insieme dei valori
  - rendere disponibile quella posizione facendo scorrere tutti gli elementi da quella posizione in poi di un posto in basso e infine scrivere il nuovo valore nella posizione resa disponibile

# Inserimento in array ordinato



# Inserimento in array ordinato

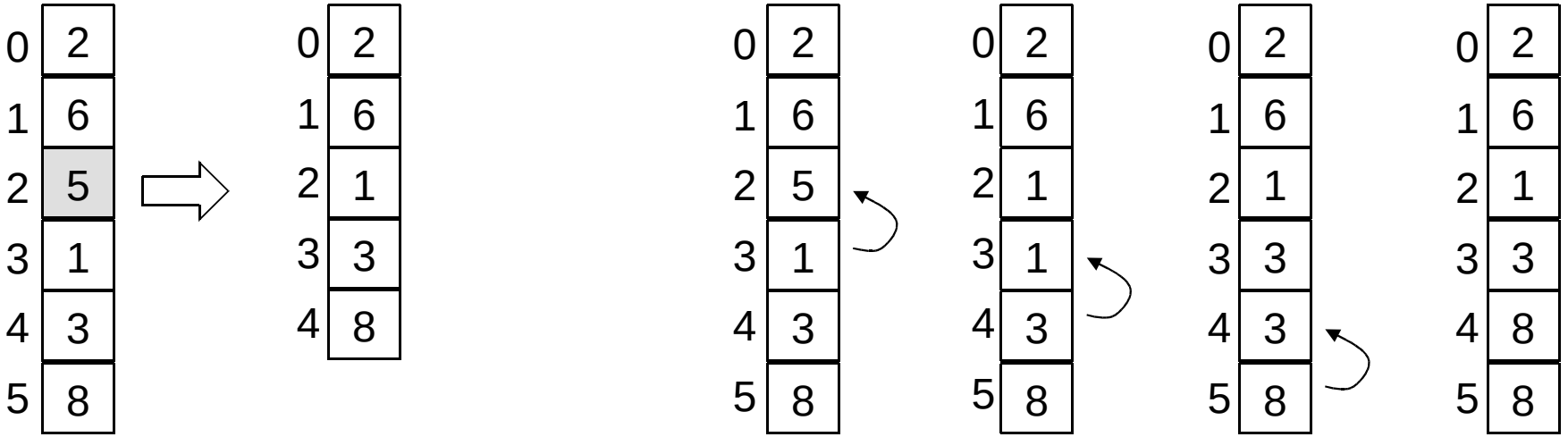
```
int cerca_pos(int v[], int n, int x){
    // cerca la posizione
    // del primo elemento >= x nell'array v
    int i;

    i = 0;
    while(i<n && v[i] <= x)
        i++;

    return(i);
}

void crea_spazio(int v[], int n, int pos)
{
    // crea una posizione libera nell'array v
    // in corrispondenza dell'indice index
    // si assume ci sia spazio disponibile nell'array
    for(int i = n-1; i >= pos; i--)
        v[i+1] = v[i];
}
```

Che succede se:
x < min[v] ? (inserimento in testa)
x > max[v] ? (inserimento in coda)

```
#include <iostream>
#include "InsExtr.h"
void stampa_array(int vet[], int riemp);
using namespace std;

int main(int argc, char** argv) {
    int vet[10] = {1, 3, 4, 6, 8, 11, 13, 18};
    int n = 8, int x, index;

    cout << "array prima dell'inserimento:\n ";
    stampa_array(vet, n);
    cout << "\nx: "; cin >> x;

    index = cerca_pos(vet, n, x);
    crea_spazio(vet, n, index);

    // inserimento e aggiornamento riempimento
    vet[index] = x; n++;

    cout << "\narray dopo l'inserimento:\n ";
    stampa_array(vet, n);

    return (EXIT_SUCCESS);
}
```

# Eliminazione di un valore

- Nel caso si debba eliminare un certo valore da un array, i passi da compiere sono:
  - ricerca nell'array per identificare la posizione del valore [se presente]
  - cancellazione logica tramite scorrimento di una posizione verso l'alto dei successivi elementi dell'array



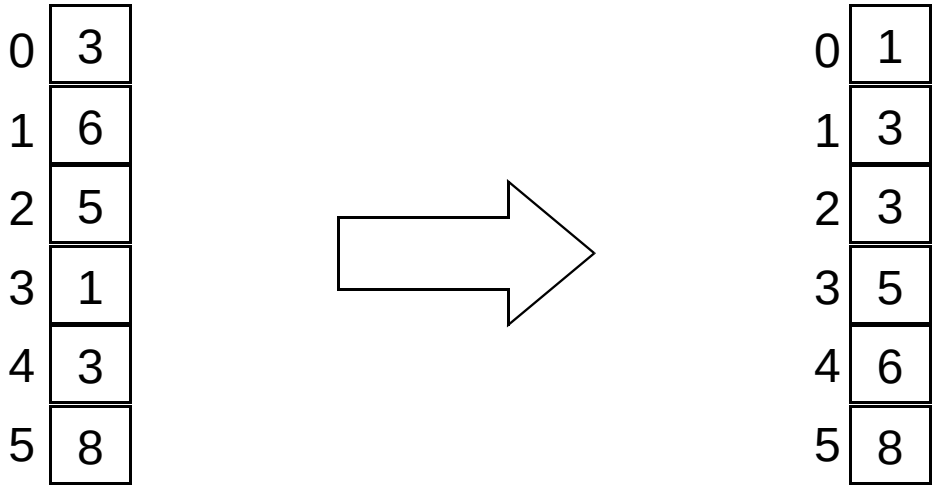
# Eliminazione di un valore

```
void elimina_pos(int v[], int n, int pos)
{
    // elimina l'elemento in posizione pos
    for (int i = pos; i < n - 1; i++)
        v[i] = v[i + 1];
}
```

```
int main(int argc, char** argv) {
    int vet[] = {2, 4, 7, 9, 12, 17}, n = 6;
    int x = 9, pos;

    pos = cerca_fun(vet,n,x);

    if(pos >= 0){
       elimina_pos(vet,n,pos);
       n--;
    }
    return (EXIT_SUCCESS);
}
```

# Ordinamento di un array

- L'operazione di ordinamento consiste in una permutazione degli elementi nell'array in modo che, al termine dell'ordinamento, la disposizione degli elementi nell'array rispetti un ordine specifico [p.es. crescente].
- Lo scopo dell'ordinamento è di facilitare successive ricerche di elementi nell'array che è stato ordinato.
- Sono stati proposti numerosi algoritmi di ordinamento, con caratteristiche diverse. Non esiste l'algoritmo di ordinamento ottimo.



# Ordinamento per selezione [select sort]

- Questo algoritmo si basa sul seguente principio:
  - dato un array **vet** di **N** elementi, si determina l'elemento minimo tra **vet[0]**, **vet[1]**, … , **vet[N-1]** e lo si scambia con il primo elemento
- Le operazioni vengono poi ripetute su **N-1** elementi a partire da **vet[1]**, poi su **N-2** elementi a partire da **vet[2]**, …, su 2 elementi a partire da **vet[N-2]**.

```
for (int i=0; i < N-1; i++){
   determina l'elemento minimo in vet[i],…,vet[N-1] e
   scrivi in k la sua posizione

   scambia vet[i] e vet[k]
}
```

```
int cercaposmin(int v[], int n, int start){
    int min,pos;

    pos = start; min = v[start];
    for(int i = start+1; i < n; i++)
        if (v[i] < min){
            min = v[i];
            pos = i;
        }
    return(pos);
}
```

```
void selectsort(int v[], int n){
    int k;

    for(int i = 0; i < n-1; i++){
        k = cercaposmin(v,n,i);
        if(k != i)
            swap(v[i],v[k]);
    }
}
```

# Problema

- Scrivere una funzione che restituisca i primi tre elementi in ordine crescente di un array di interi.

# Ordinamento per gorgogliamento [bubble sort]

- Questo algoritmo si basa sul seguente principio:
  - dato un array **`vet`** di **`N`** elementi, si verifica se la coppia (**`vet[N-2]`**,**`vet[N-1]`**) sia ordinata; se non lo è, si scambiano i due elementi
  - si ripete lo stesso con le coppie (**`vet[N-3]`**, **`vet[N-2]`**), (**`vet[N-4]`**,**`vet[N-3]`**), …, (**`vet[1]`**,**`vet[0]`**)
  - al termine, in prima posizione ci sarà l'elemento minimo
- Le operazioni vengono poi ripetute  altre N-1 volte per completare l'ordinamento

# Ordinamento per gorgogliamento [bubble sort]

```
void bubblesort1(int v[], int n){
    for(int t = 0; t < n; t++)
        for(int i = n-1; i > 0; i--)
            if(v[i] < v[i-1])
                swap(v[i], v[i-1]);
}
```

- Possibile migliorare l'algoritmo ?
- Che cosa resta invariato durante il ciclo esterno ?

# Ordinamento per gorgogliamento [bubble sort]

```
void bubblesort2(int v[], int n){
    for(int t = 0; t < n; t++)
        for(int i = n-1; i > t; i--)
            if(v[i] < v[i-1])
                swap(v[i], v[i-1]);
}
```

- Si possono sfruttare i risultati delle verifiche di ordinamento delle coppie di elementi successivi ?

# Ordinamento per gorgogliamento [bubble sort]

```
void bubblesort3(int v[], int n) {
    int last=0,end;

    for (int t = 0; t < n; t++) {
        end = last;
        for (int i = n - 1; i > end; i--)
            if (v[i] < v[i - 1]) {
                swap(v[i], v[i - 1]);
                last = i;
            }
    }
}
```

- È possibile verificare se l'array abbia completato l'ordinamento prima del termine del **`for`** esterno ?

# Ordinamento per gorgogliamento [bubble sort]

```
void bubblesort4(int v[], int n) {
    int last=0,end;
    bool fattoswap;

    do{
        fattoswap = false;
        end = last;
        for (int i = n - 1; i > end; i--)
            if (v[i] < v[i - 1]) {
                swap(v[i], v[i - 1]);
                last = i;
                fattoswap = true;
            }
    }while(fattoswap);
}
```

# Confronto tra select sort e bubble sort

- Quale dei due algoritmi è più conveniente?
  - In termini di confronti ?
  - In termini di scambi ?