Algoritmi su array

- Per utilizzare gli array come insiemi dinamici di dati (la cui composizione, cioè, può variare dinamicamente) è necessario realizzare operazioni quali:
  - Ricerca di un valore nell'array (già vista)
  - 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

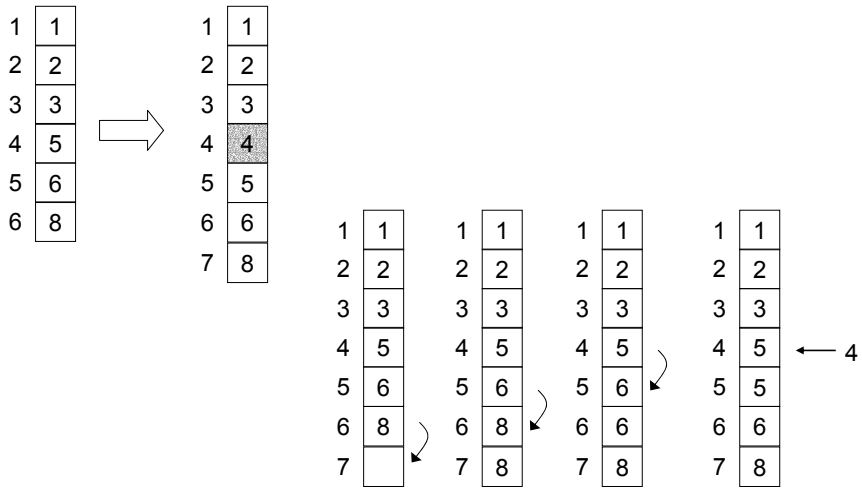
Inserimento in array ordinato

- 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

- Ricerca della posizione all'interno dell'array
- Creazione di un posto vuoto
- Programma completo

```
function p = cerca_pos(vet,num,val)
% cerca la posizione del primo elemento >= val nell'array vet di num elementi

% parametri di ingresso
% vet: array su cui effettuare la ricerca
% num: numero di elementi da leggere
% val: valore con il quale effettuare la ricerca

% parametri di uscita
% p: indice del primo elemento >= val

% variabili locali
% i: indice per lo scorrimento dell'array

i=1;

while(i<=num & vet(i)<val)
   i=i+1;
end

p = i;

% fine funzione cerca_pos
```

```
function v=crea_spazio(vet,num,index)
% crea una posizione libera in corrispondenza dell'indice index

% parametri di ingresso
% vet: array da ampliare
% num: numero di elementi nell'array
% index: posizione in cui creare lo spazio

% parametri di uscita
% v: array modificato

% variabili locali
% i: indice di scorrimento per l'array

% copia vet in v
v = vet;

% crea la posizione libera
for i=num:-1:index
    v(i+1)=v(i);
end
```

```
function main
% lettura di un array ordinato, inserimento di un nuovo valore


% variabili utilizzate
% v:    array in input
% n:    dimensione dell'array fornito in input
% val:  variabile contenente il valore da inserire
% pos:  indice dell'array in cui inserire il valore
% cont: variabile contenente il numero di occorrenze trovate

% input dimensione
n=input('Numero elementi: ');

% input array
v = leggiarray(n);

% ricerca del minimo
val = input('Valore da inserire: ');

% ricerca della posizione
pos = cerca_pos(v,n,val);

% creazione della posizione libera in corrispondenza di pos
v = crea_spazio(v,n,pos);

% inserimento del valore e aggiornamento del riempimento
v(pos) = val;
n=n+1;

% stampa dei risultati
fprintf('\nArray modificato:\n');
stampaarray(v,n);

%%%%%% fine main %%%%%%%%%%%%%%%%
```

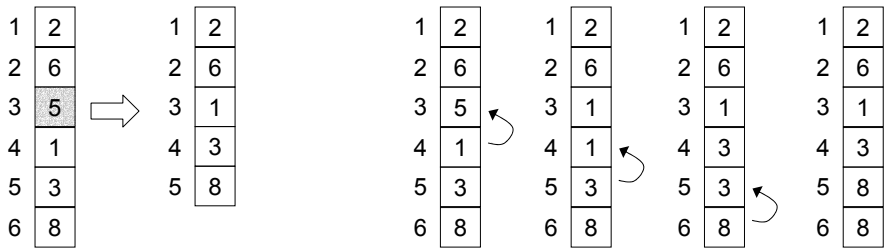
```
function vet=leggiarray(num)
% legge un array di num elementi

% parametri di ingresso
% num: numero di elementi da leggere

% parametri di uscita
% vet: array letto

% dimensionamento array
vet=zeros(num,1);
% ciclo di lettura
for i=1:num
   fprintf('Valore %d: ',i);
   vet(i)=input('');
end
% fine funzione leggiarray



function stampaarray(vet,num)
% Stampa gli elementi dell'array vet

% parametri di ingresso
% vet: array da stampare
% num: numero degli elementi dell'array

% parametri di uscita
% nessuno

% variabili usate
% i:   indice per scorrere l'array

for i=1:num
   fprintf('%g\n',vet(i));
end
% fine funzione stampaarray
```

```
function p = cerca_pos(vet,num,val)
% cerca la posizione del primo elemento >= val nell'array vet di num elementi

% parametri di ingresso
% vet: array su cui effettuare la ricerca
% num: numero di elementi da leggere
% val: valore con il quale effettuare la ricerca

% parametri di uscita
% p: indice del primo elemento >= val

% variabili locali
% i: indice per lo scorrimento dell'array

i=1;

while(i<=num & vet(i)<val)
   i=i+1;
end

p = i;

% fine funzione cerca_pos
```

```
function v=crea_spazio(vet,num,index)
% crea una posizione libera in corrispondenza dell'indice index

% parametri di ingresso
% vet: array da ampliare
% num: numero di elementi nell'array
% index: posizione in cui creare lo spazio

% parametri di uscita
% v: array modificato

% variabili locali
% i: indice di scorrimento per l'array

% copia vet in v
v = vet;

% crea la posizione libera
for i=num:-1:index
    v(i+1)=v(i);
end

% fine funzione crea_spazio
```

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

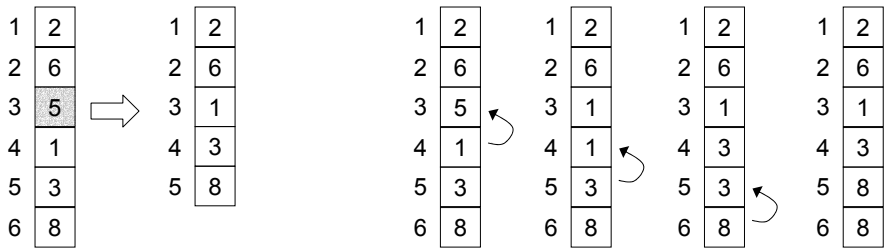


Eliminazione di un valore

- Ricerca del valore nell'array
- Eliminazione di un elemento di posizione data
- Programma completo

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

- Ricerca del valore nell'array
- Eliminazione di un elemento di posizione data
- Programma completo

```
function p = cerca_val(vet,num,val)
% cerca la posizione del primo elemento == val nell'array vet di num elementi
% restituisce la posizione, se il valore è stato trovato, 0 altrimenti

% parametri di ingresso
% vet: array su cui effettuare la ricerca
% num: numero di elementi da leggere
% val: valore con il quale effettuare la ricerca

% parametri di uscita
% p: indice del primo elemento >= val

% variabili locali
% i: indice per lo scorrimento dell'array

p=0;
i=1;

while(i<=num & vet(i)~=val)
   i=i+1;
end

if (i<=num)
   p = i;
end

% fine funzione cerca_val
```

```
function v=elimina_pos(vet,num,index)
% elimina l'elemento in posizione index

% parametri di ingresso
% vet: array da ampliare
% num: numero di elementi nell'array
% index: posizione dell'elemento da eliminare

% parametri di uscita
% v: array modificato

% variabili locali
% i: indice di scorrimento per l'array

% copia vet in v
v = vet;

% elimina l'elemento
for i = index : num-1
    v(i)=v(i+1);
end
```

```
function main
% lettura di un array, ricerca ed eliminazione di un valore


%% variabili utilizzate
% v:    array in input
% n:    dimensione dell'array fornito in input
% val:  variabile contenente il valore da eliminare
% pos:  indice nell'array dell'elemento da eliminare
% cont: variabile contenente il numero di occorrenze trovate

% input dimensione
n=input('Numero elementi: ');

% input array
v = leggiarray(n);

% ricerca del minimo
val = input('Valore da eliminare: ');

% ricerca della posizione
pos = cerca_val(v,n,val);

if (pos==0)
   % il valore non è presente nell'array
   fprintf('\nIl valore %d non e'' presente nell''array',val);
else
   % il valore è presente; può essere eliminato
   fprintf('\nIl valore %d e'' presente nell''array in posizione %d \n',val,pos);

   % eliminazione del valore in posizione pos
   v = elimina_pos(v,n,pos);

   % aggiornamento del riempimento
   n=n-1;

   % stampa dei risultati
   fprintf('\nArray modificato:\n');
   stampaarray(v,n);
end

%%%%%% fine main %%%%%%%%%%%%%%%%
```

```
function vet=leggiarray(num)
% legge un array di num elementi

% parametri di ingresso
% num: numero di elementi da leggere

% parametri di uscita
% vet: array letto

% dimensionamento array
vet=zeros(num,1);
% ciclo di lettura
for i=1:num
   fprintf('Valore %d: ',i);
   vet(i)=input('');
end
% fine funzione leggiarray
```

```
function stampaarray(vet,num)
% Stampa gli elementi dell'array vet

% parametri di ingresso
% vet: array da stampare
% num: numero degli elementi dell'array

% parametri di uscita
% nessuno

% variabili usate
% i:   indice per scorrere l'array

for i=1:num
   fprintf('%g\n',vet(i));
end
% fine funzione stampaarray
```

```
function p = cerca_val(vet,num,val)
% cerca la posizione del primo elemento == val nell'array vet di num elementi
% restituisce la posizione, se il valore è stato trovato, 0 altrimenti

% parametri di ingresso
% vet: array su cui effettuare la ricerca
% num: numero di elementi da leggere
% val: valore con il quale effettuare la ricerca

% parametri di uscita
% p: indice del primo elemento >= val

% variabili locali
% i: indice per lo scorrimento dell'array

p=0;
i=1;

while(i<=num & vet(i)~=val)
   i=i+1;
end

if (i<=num)
   p = i;
end

% fine funzione cerca_val
```

```
function v=elimina_pos(vet,num,index)
% elimina l'elemento in posizione index

% parametri di ingresso
% vet: array da ampliare
% num: numero di elementi nell'array
% index: posizione dell'elemento da eliminare

% parametri di uscita
% v: array modificato

% variabili locali
% i: indice di scorrimento per l'array

% copia vet in v
v = vet;

% elimina l'elemento
for i = index : num-1
    v(i)=v(i+1);
end
```

Eliminazione di tutte le occorrenze di un valore

- Come si comporta il programma nel caso siano presenti più occorrenze del valore da eliminare ?
- Come può essere modificato per eliminare tutte le occorrenze del valore dato ?
- Possibili soluzioni
  - Soluzione 1
  - Soluzione 2
  - Quali sono le differenze ? Qual è la soluzione più conveniente ?

```
function main
% lettura di un array, ricerca ed eliminazione di tutte le occorrenze di un valore
% soluzione 1
% le occorrenze vengono  ricercate ed eliminate una alla volta

% variabili utilizzate
% v:    array in input
% n:    dimensione dell'array fornito in input
% val:  variabile contenente il valore da eliminare
% pos:  indice nell'array dell'elemento da eliminare
% cont: variabile contenente il numero di occorrenze trovate

% input dimensione
n=input('Numero elementi: ');

% input array
v = leggiarray(n);

% lettura del valore da eliminare
%
val = input('Valore da eliminare: ');

% inizializzazione del numero di occorrenze eliminate
cont = 0;
% ricerca della posizione
pos = cerca_val(v,n,val);

while(pos~=0)
   % trovata occorrenza del valore
   fprintf('\nIl valore %d e'' presente nell''array in posizione %d \n',val,pos);
   % eliminazione del valore in posizione pos
   v = elimina_pos(v,n,pos);
   % aggiornamento del riempimento
   n=n-1;
   % aggiorna il numero di occorrenze eliminate
   cont=cont+1;

   % ricerca di un'altra occorrenza
   pos = cerca_val(v,n,val);
end

% stampa dei risultati
```

```
fprintf('\nSono state eliminate %d occorrenze del valore %d\n',cont, val);
fprintf('\nArray modificato:\n');
stampaarray(v,n);

%%%%%% fine main %%%%%%%%%%%%%%%%
```

```
function vet=leggiarray(num)
% legge un array di num elementi

% parametri di ingresso
% num: numero di elementi da leggere

% parametri di uscita
% vet: array letto

% dimensionamento array
vet=zeros(num,1);
% ciclo di lettura
for i=1:num
    fprintf('Valore %d: ',i);
    vet(i)=input('');
end
% fine funzione leggiarray



function stampaarray(vet,num)
% Stampa gli elementi dell'array vet

% parametri di ingresso
% vet: array da stampare
% num: numero degli elementi dell'array

% parametri di uscita
% nessuno

% variabili usate
% i:   indice per scorrere l'array

for i=1:num
    fprintf('%g\n',vet(i));
end
% fine funzione stampaarray
```

```
function p = cerca_val(vet,num,val)
% cerca la posizione del primo elemento == val nell'array vet di num elementi
% restituisce la posizione, se il valore è stato trovato, 0 altrimenti

% parametri di ingresso
% vet: array su cui effettuare la ricerca
% num: numero di elementi da leggere
% val: valore con il quale effettuare la ricerca

% parametri di uscita
% p: indice del primo elemento >= val

% variabili locali
% i: indice per lo scorrimento dell'array

p=0;
i=1;

while(i<=num & vet(i)~=val)
   i=i+1;
end

if (i<=num)
   p = i;
end

% fine funzione cerca_val
```

```
function v=elimina_pos(vet,num,index)
% elimina l'elemento in posizione index

% parametri di ingresso
% vet: array da ampliare
% num: numero di elementi nell'array
% index: posizione dell'elemento da eliminare

% parametri di uscita
% v: array modificato

% variabili locali
% i: indice di scorrimento per l'array

% copia vet in v
v = vet;

% elimina l'elemento
for i = index : num-1
    v(i)=v(i+1);
end
```

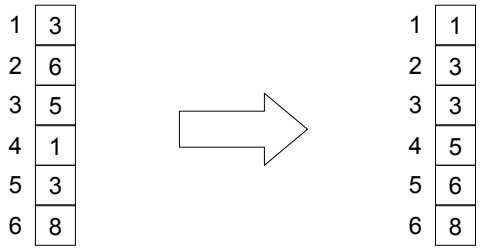
```
function main
% lettura di un array, ricerca ed eliminazione di tutte le occorrenze di un valore
% soluzione 2
% vengono contemporaneamente ricercate ed eliminate tutte le occorrenze

% variabili utilizzate
% v:    array in input
% n:    dimensione dell'array fornito in input
% val:  variabile contenente il valore da eliminare
% cont: variabile contenente il numero di occorrenze trovate

% input dimensione
n=input('Numero elementi: ');

% input array
v = leggiarray(n);

% lettura del valore da eliminare
%
val = input('Valore da eliminare: ');

% ricerca ed eliminazione
[cont,v] = eliminaogniocc(v,n,val);

% aggiornamento del riempimento
n = n-cont;

% stampa dei risultati
fprintf('\nSono state eliminate %d occorrenze del valore %d\n',cont, val);
fprintf('\nArray modificato:\n');
stampaarray(v,n);

%%%%%% fine main %%%%%%%%%%%%%%%%
```

```
function vet=leggiarray(num)
% legge un array di num elementi

% parametri di ingresso
% num: numero di elementi da leggere

% parametri di uscita
% vet: array letto

% dimensionamento array
vet=zeros(num,1);
% ciclo di lettura
for i=1:num
    fprintf('Valore %d: ',i);
    vet(i)=input('');
end
% fine funzione leggiarray



function stampaarray(vet,num)
% Stampa gli elementi dell'array vet

% parametri di ingresso
% vet: array da stampare
% num: numero degli elementi dell'array

% parametri di uscita
% nessuno

% variabili usate
% i:   indice per scorrere l'array

for i=1:num
    fprintf('%g\n',vet(i));
end
% fine funzione stampaarray
```

```
function [k,v]=eliminaogniocc(vet,num,val)
% elimina tutte le occorrenze del valore val nell'array

% parametri di ingresso
% vet: array da ampliare
% num: numero di elementi nell'array
% val: valore da eliminare

% parametri di uscita
% k: numero di occorrenze eliminate
% v: array modificato

% variabili locali
% i: indice di scorrimento per l'array

% copia vet in v
v = vet;

% inizializza il numero di occorrenze trovate
k = 0;

% cerca ed elimina le occorrenze di val
for i = 1 : num
    if(v(i) == val)
        k=k+1;
    else
        v(i-k)=v(i);
    end
end
```

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(1), vet(2), … , vet(N) e lo si scambia con il primo elemento
- Le operazioni vengono poi ripetute su N-1 elementi a partire da vet(2), poi su N-2 elementi a partire da vet(3), …, su 2 elementi a partire da vet(N-1).

```
for i=1:N
   determina l'elemento minimo in vet(i),…,vet(N) e
   scrivi in k la sua posizione

   scambia vet(i) e vet(k)
end
```

```
function main
% lettura e ordinamento di un array


% variabili utilizzate
% v:    array in input
% n:    dimensione dell'array fornito in input
% pos:  indice nell'array del minimo corrente

% input dimensione
n=input('Numero elementi: ');

% input array
v = leggiarray(n);

fprintf('\nArray prima dell''ordinamento:\n');
stampaarray(v,n);

% ordinamento
v = select_sort(v,n);

fprintf('\nArray dopo l''ordinamento:\n');
stampaarray(v,n);

%%%%%% fine main %%%%%%%%%%%%%%%%
```

```
function vet=leggiarray(num)
% legge un array di num elementi

% parametri di ingresso
% num: numero di elementi da leggere

% parametri di uscita
% vet: array letto

% dimensionamento array
vet=zeros(num,1);
% ciclo di lettura
for i=1:num
    fprintf('Valore %d: ',i);
    vet(i)=input('');
end
% fine funzione leggiarray



function stampaarray(vet,num)
% Stampa gli elementi dell'array vet

% parametri di ingresso
% vet: array da stampare
% num: numero degli elementi dell'array

% parametri di uscita
% nessuno

% variabili usate
% i:   indice per scorrere l'array

for i=1:num
    fprintf('%g\n',vet(i));
end
% fine funzione stampaarray
```

```
function v = select_sort(vet,num)
% ordina gli elementi dell'array vet secondo l'algoritmo di select sort
% restituisce l'array ordinato

% parametri di ingresso
% vet: array su cui effettuare la ricerca
% num: numero di elementi da leggere

% parametri di uscita
% v: array ordinato

% variabili locali
% i: indice per lo scorrimento dell'array
% posmin: posizione del minimo corrente
% appo: variabile di appoggio per lo scambio

% copia vet in v
v = vet;

for i=1:num-1
   posmin=cercaposmin(v,num,i);

   if(posmin ~= i)
      appo=v(i);
      v(i)=v(posmin);
      v(posmin)=appo;
   end
end


% fine funzione select_sort
```

```
function p=cercaposmin(vet,num,index)
% restituisce la posizione dell'elemento minimo nell'array a partire da
% index

% parametri di ingresso
% vet: array
% num: numero di elementi nell'array
% index: posizione da cui cominciare la ricerca

% parametri di uscita
% p: posizione del minimo

% variabili locali
% i: indice di scorrimento per l'array

p = index;

% cerca il minimo
for i = index : num
    if(vet(i) < vet(p))
        p = i;
    end
end
```

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-1),vet(N)] sia ordinata; se non lo è, si scambiano i due elementi
  - si ripete lo stesso con le coppie [vet(N-2), vet(N-1)], [vet(N-3),vet(N-2)], …, [vet(2),vet(1)]
  - al termine, in prima posizione ci sarà l'elemento minimo
- Le operazioni vengono poi ripetute altre N-1 volte per completare l'ordinamento
- Soluzione 1

Ordinamento per gorgogliamento (bubble sort)

- Possibile migliorare l'algoritmo ?
- Che cosa resta invariato durante il ciclo esterno ?
- Soluzione 2

- Si possono sfruttare i risultati delle verifiche di ordinamento delle coppie di elementi successivi ?
- Soluzione 3

```
function main
% lettura e ordinamento di un array
% Algoritmo usato: bubble sort
% versione 1


% variabili utilizzate
% v:    array in input
% n:    dimensione dell'array fornito in input


% input dimensione
n=input('Numero elementi: ');

% input array
v = leggiarray(n);

fprintf('\nArray prima dell''ordinamento:\n');
stampaarray(v,n);

% ordinamento
v = bubble_sort(v,n);

fprintf('\nArray dopo l''ordinamento:\n');
stampaarray(v,n);

%%%%%% fine main %%%%%%%%%%%%%%%%
```

```
function vet=leggiarray(num)
% legge un array di num elementi

% parametri di ingresso
% num: numero di elementi da leggere

% parametri di uscita
% vet: array letto

% dimensionamento array
vet=zeros(num,1);
% ciclo di lettura
for i=1:num
   fprintf('Valore %d: ',i);
   vet(i)=input('');
end
% fine funzione leggiarray



function stampaarray(vet,num)
% Stampa gli elementi dell'array vet

% parametri di ingresso
% vet: array da stampare
% num: numero degli elementi dell'array

% parametri di uscita
% nessuno

% variabili usate
% i:   indice per scorrere l'array

for i=1:num
   fprintf('%g\n',vet(i));
end
% fine funzione stampaarray
```

```
function v = bubble_sort(vet,num)
% ordina gli elementi dell'array vet secondo l'algoritmo di bubble sort
% restituisce l'array ordinato

% parametri di ingresso
% vet: array
% num: numero di elementi

% parametri di uscita
% v: array ordinato

% variabili locali
% i,j: indici per lo scorrimento dell'array
% appo: variabile di appoggio per lo scambio

% copia vet in v
v = vet;

for i=1:num
    for j = num:-1:2
        if(v(j-1)>v(j))
            appo = v(j);
            v(j) = v(j-1);
            v(j-1) = appo;
        end
    end
end


% fine funzione bubble_sort
```

```
function main
% lettura e ordinamento di un array
% Algoritmo usato: bubble sort
% versione 2


% variabili utilizzate
% v:    array in input
% n:    dimensione dell'array fornito in input


% input dimensione
n=input('Numero elementi: ');

% input array
v = leggiarray(n);

fprintf('\nArray prima dell''ordinamento:\n');
stampaarray(v,n);

% ordinamento
v = bubble_sort(v,n);

fprintf('\nArray dopo l''ordinamento:\n');
stampaarray(v,n);

%%%%%% fine main %%%%%%%%%%%%%%%%
```

```
function vet=leggiarray(num)
% legge un array di num elementi

% parametri di ingresso
% num: numero di elementi da leggere

% parametri di uscita
% vet: array letto

% dimensionamento array
vet=zeros(num,1);
% ciclo di lettura
for i=1:num
   fprintf('Valore %d: ',i);
   vet(i)=input('');
end
% fine funzione leggiarray



function stampaarray(vet,num)
% Stampa gli elementi dell'array vet

% parametri di ingresso
% vet: array da stampare
% num: numero degli elementi dell'array

% parametri di uscita
% nessuno

% variabili usate
% i:   indice per scorrere l'array

for i=1:num
   fprintf('%g\n',vet(i));
end
% fine funzione stampaarray
```

```
function v = bubble_sort(vet,num)
% ordina gli elementi dell'array vet secondo l'algoritmo di bubble sort
% restituisce l'array ordinato

% parametri di ingresso
% vet: array
% num: numero di elementi

% parametri di uscita
% v: array ordinato

% variabili locali
% i,j: indici per lo scorrimento dell'array
% appo: variabile di appoggio per lo scambio

% copia vet in v
v = vet;

for i=1:num
    for j = num:-1:i+1
        if(v(j-1)>v(j))
            appo = v(j);
            v(j) = v(j-1);
            v(j-1) = appo;
        end
    end
end


% fine funzione bubble_sort
```

```
function main
% lettura e ordinamento di un array
% Algoritmo usato: bubble sort
% versione 3


% variabili utilizzate
% v:    array in input
% n:    dimensione dell'array fornito in input


% input dimensione
n=input('Numero elementi: ');

% input array
v = leggiarray(n);

fprintf('\nArray prima dell''ordinamento:\n');
stampaarray(v,n);

% ordinamento
v = bubble_sort(v,n);

fprintf('\nArray dopo l''ordinamento:\n');
stampaarray(v,n);

%%%%%% fine main %%%%%%%%%%%%%%%%
```

```
function vet=leggiarray(num)
% legge un array di num elementi

% parametri di ingresso
% num: numero di elementi da leggere

% parametri di uscita
% vet: array letto

% dimensionamento array
vet=zeros(num,1);
% ciclo di lettura
for i=1:num
   fprintf('Valore %d: ',i);
   vet(i)=input('');
end
% fine funzione leggiarray



function stampaarray(vet,num)
% Stampa gli elementi dell'array vet

% parametri di ingresso
% vet: array da stampare
% num: numero degli elementi dell'array

% parametri di uscita
% nessuno

% variabili usate
% i:   indice per scorrere l'array

for i=1:num
   fprintf('%g\n',vet(i));
end
% fine funzione stampaarray
```

```
function v = bubble_sort(vet,num)
% ordina gli elementi dell'array vet secondo l'algoritmo di bubble sort
% restituisce l'array ordinato

% parametri di ingresso
% vet: array
% num: numero di elementi

% parametri di uscita
% v: array ordinato

% variabili locali
% i,j: indici per lo scorrimento dell'array
% appo: variabile di appoggio per lo scambio
% scambio: flag booleano che indica se è stato effettuato almeno uno
%          scambio

% copia vet in v
v = vet;

% scambio inizializzato a true per entrare nel WHILE
scambio = 1;

% inizializzazione di i
i=1;
while (scambio)
    % scambio inizializzato a FALSE prima del ciclo FOR
    scambio = 0;
    for j = num:-1:i+1
        if(v(j-1)>v(j))
            appo = v(j);
            v(j) = v(j-1);
            v(j-1) = appo;
            % fatto uno scambio -> viene modificato il flag
            scambio = 1;
        end
    end
    % viene aggiornato i
    i = i+1;
end

% fine funzione bubble_sort
```