Esempio:

Ricerca esaustiva di un elemento in un vettore di dimensione N

```
boolean ricerca (vettore vet, int el)
{int  i=0;
 boolean T=falso;
 while ((i<N)&&(T==falso)) /* (1) */
  { if (el==vet[i])        /* (2) */
                 T=vero;
    i++;}
 return T;
}
```

Esempi su vettori: Ricerca del valore minimo e massimo di un vettore

```
#define N 15
typedef int vettore[N];

 /* dichiarazione di due funzioni */
int minimo (vettore vet);
int massimo (vettore vet);

main ()
{int i;
 vettore a;

 printf ("Scrivi %d numeri interi\n", N);
 for (i = 0; i < N; i++)
            { scanf ("%d", &a[i]); }
 puts ("L'insieme dei numeri è: ");
 for (i = 0; i<N; i++)
       { printf(" %d",a[i]); }
 puts("\n");
 printf ("Il minimo vale %d e il
          massimo è %d\n",
     minimo(a), massimo(a));
}

int minimo (vettore vet)
{int  i, min;
 for  (min = vet[0], i = 1; i < N; i ++)
   {if (vet[i]<min) /* istr. dom. */
         min = vet[i]; }
 return min;
}
```

```
int massimo (vettore vet)
{int  i, max;
 for (max = vet[0], i = 1; i < N; i ++)
    {if (vet[i]>max) /* istr. dominante*/
        max=vet[i];}
  return max;
}
```

Esempi su vettori: Ricerca esaustiva di un elemento in un vettore

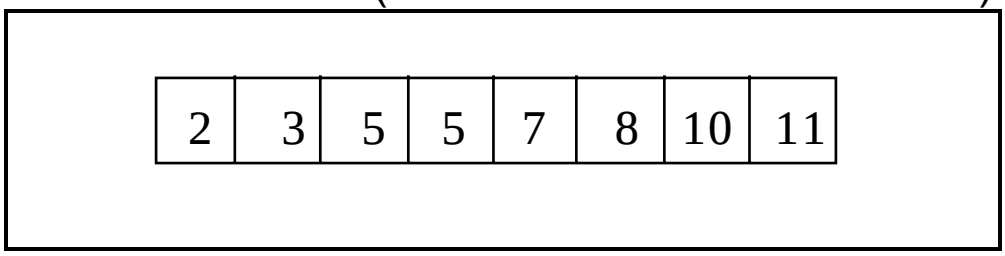
```
#include <stdio.h>
#define N 15
typedef int vettore[N];
typedef enum {falso,vero} boolean;
main ()
{boolean ricerca (vettore vet, int el);
 int i;
 vettore a;
 printf ("Scrivi %d numeri interi\n", N);
 for (i = 0; i < N; i++)
     scanf ("%d", &a[i]);
 scanf("\n ");
 scanf ("Valore da cercare: %d",&i);
 if (ricerca(a,i)) printf("\nTrovato\n");
 else printf("\nNon trovato\n");
}

boolean ricerca (vettore vet, int el)
{int  i=0;
 boolean T=falso;
 while ((i<N)&&(T==falso))/* istr.
                          dominante */
  { if (el==vet[i]) T=vero;
    i++;}
 return T;
}
```

Sapendo che il vettore è ***ordinato***, la ricerca può essere ottimizzata.

Vettore ordinato:
Esiste una relazione d'ordine totale sul dominio degli elementi del vettore e:

$\forall i,j$: $i<j$ si ha $V[i]<=V[j]$

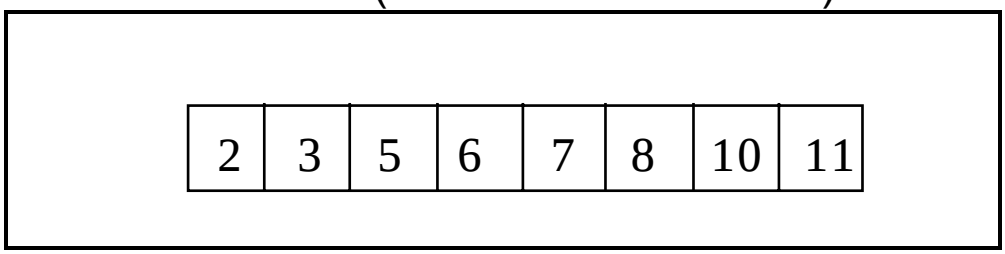
(***in senso non decrescente***)



se invece:

$\forall i,j$: $i<j$ si ha $V[i]<V[j]$

(***in senso crescente***)



In modo analogo si definiscono l'ordinamento in senso ***non crescente*** e ***decrescente***.

Supponiamo di cercare un elemento in un vettore ordinato in senso non decrescente.

Ricerca binaria su un vettore ordinato in senso non decrescente con componenti di indice da *first* a *last*.

La tecnica di ***ricerca binaria*** rispetto alla ricerca esaustiva, consente di eliminare ad ogni passo meta` degli elementi del vettore.

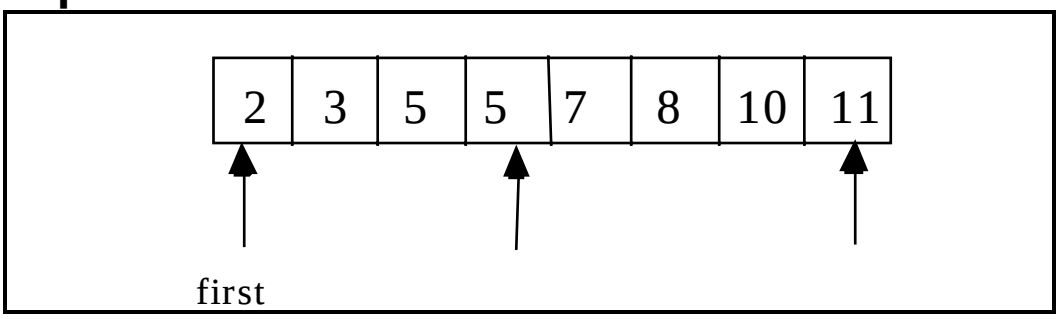
Si confronta l'elemento cercato *el* con quello mediano del vettore, V[*med*].

Se *el*==V[*med*], fine della ricerca (trovato=true).

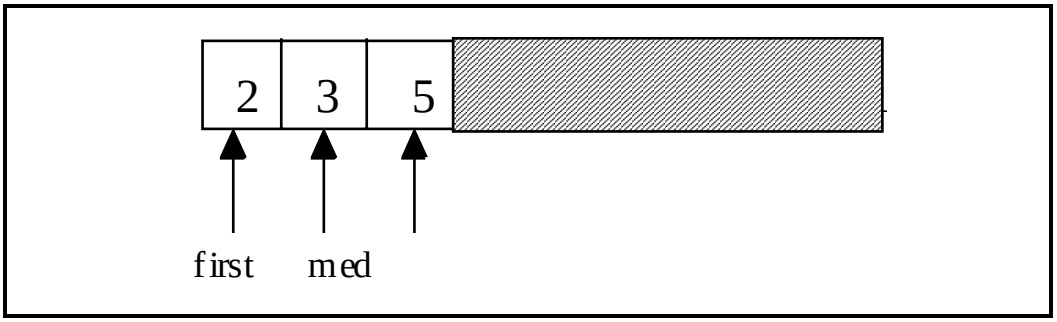
Altrimenti,

> se il vettore ha almeno due componenti (*first* <*last*):
>
> > se *el*<V[*med*], ripeti la ricerca nella prima meta` del vettore (indici da *first* a *med*-1);
> >
> > se *el*>V[*med*], ripeti la ricerca nella seconda meta` del vettore (indici da *med*-1 a *last*).

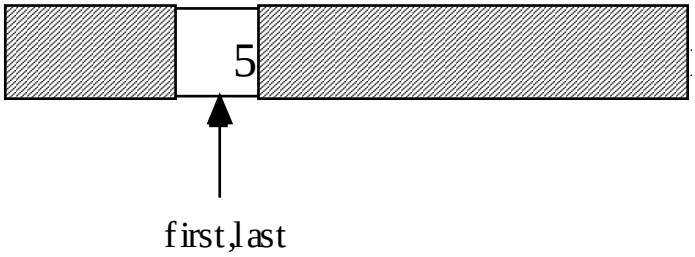
Esempio: si cerca 4



med= (first+last) / 2
el<V[med]



el>V[med]
Vettore ad una componente:



```
typedef enum {falso, vero} boolean;

boolean ricerca_bin (vettore vet, int el,
                     int *pos)
{int  first=0,
   last=N-1,
   med=(first+last)/2 ;
 boolean T=falso;
 while ((first<=last)&&(T==falso))
                  /* istr. dom. */
  { if (el==vet[med])
      {T=vero; *pos=med;}
    else
       if (el < vet[med])  last=med-1;
       else   first=med+1;
    med = (first + last) / 2;
  }
 return  T;
}
```

Chiamata:

```
if (ricerca(a,i,&pos))
  printf("\nTrovato in posizione
     %d\n", pos);
else printf("\nNon trovato\n");
```

✐ **Esercizio:**

1) Scrivere una versione ricorsiva della funzione di ricerca binaria.

2) Scriverne una versione che ha come parametri anche gli indici first e last.

Esercizio:

È dato un vettore di dimensione N+k contenente N numeri interi, ordinati in senso non decrescente.
Si suppone di ricevere uno alla volta k interi e di inserirli nel vettore, mantenendo l'ordine ad ogni passo di inserimento.

```
#include <stdio.h>
#define dim 10
#define k 4
typedef int vettore[dim];
typedef enum {falso,vero} boolean;
int N=6;
void main (void)
{void inserisci (vettore vet, int el);
 int i;
 vettore V;
   ...
}

void inserisci (vettore vet, int el)
{int  i=0;
 boolean T=falso;
 while ((i<N)&&(T==falso))
  { if (el<=vet[i]) T=vero;
    else i++;}
 if (T=vero)
  for(j=dim-2;j>=i;j--)
        V[j++]=V[j];  /* shift */
 V[i]=el;  N++;
}
```

Algoritmi di ordinamento

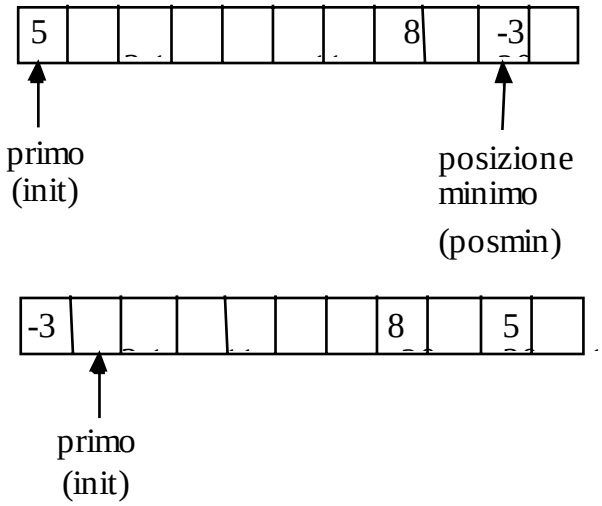
Consideriamo algoritmi di ordinamento interni (elementi in memoria centrale).

Vettore di elementi di un certo tipo, sul quale e` definita una ***relazione d'ordine totale*** (ad esempio, tipo float)

```
#define MAX 11
#define true 1
#define false 0
typedef float vector [MAX];
```

Naive sort (o selection sort, o ordinamento per minimi successivi):

Ad ogni passo seleziona il minimo nel vettore e lo pone nella prima posizione, richiamandosi ed escludendo dal vettore il primo elemento.




```
while (<il vettore ha piu` di una componente>)
   {     <individua il minimo nel vettore corrente>
         <scambia se necessario il primo elemento del
               vettore corrente con A[posmin]>
         <considera come vettore corrente quello
               precedente tolto il primo elemento>   }
```

Naive Sort: Realizzazione

```
#include <stdio.h>
#define N 5
typedef int vettore[N];
typedef enum {falso,vero} boolean;
void leggi(vettore a); /*input del
vettore */
void scrivi(vettore a); /*stampa*/

main ()
{void naive_sort (vettore vet);

 int i;
 vettore a;

 leggi(a);
 naive_sort(a);
 scrivi(a);
}


void  naive_sort (vettore vet)
{int  j, i, posmin, min;
 for  (j=0; j < N; j++)
   {posmin=j;
    for (min=vet[j],i=j+1;i<N; i++)
        if (vet[i]<min)/*i. dom.*/
          {min=vet[i];posmin=i;}

    if (posmin != j) /*scambio */
       { min=vet[posmin];
         vet[posmin]=vet[j];
         vet[j]=min;
       }
   }
}
```

```
void leggi(vettore a)
{int i;

printf ("Scrivi %d interi\n", N);
 for (i = 0; i < N; i++)
      scanf ("%d", &a[i]);
 }

void scrivi(vettore a)
{int i;

 printf ("Vettore ordinato:\n");
 for (i = 0; i < N; i++)
      printf ("%d\t", a[i]);
 }
```

Si eseguono tutti i confronti anche se il vettore è già ordinato.

Esercizio:
Scrivere una versione ricorsiva dell'agoritmo di naive sorting.

Bubble sort (ordinamento a bolla):

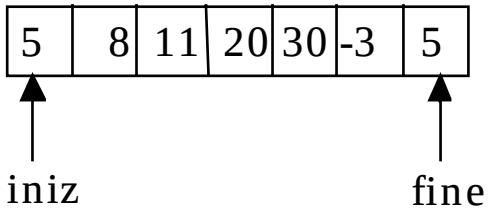
Si basa sul fatto che esiste un ***ordinamento totale*** sugli elementi del vettore: dati due elementi adiacenti A[i] e A[i+1], se non rispettano l'ordinamento vengono scambiati.

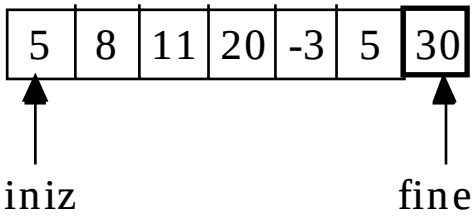
```
do
    per tutte le coppie di elementi adiacenti del
    vettore A esegui:
        se  A[i] > A[i+1] allora scambiali;
while   il vettore A e' ordinato.
```

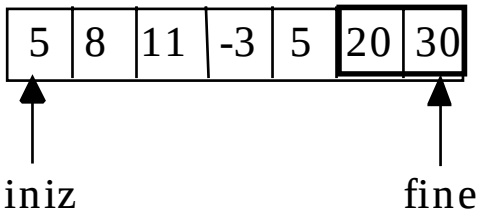
- Il vettore e` ordinato quando non ci sono più scambi.

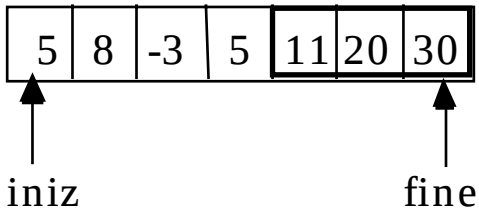
- Si chiama ***ordinamento a bolla*** perché dopo la prima scansione del vettore, l'elemento massimo si porta in ultima posizione (gli elementi più piccoli "salgono" verso le posizioni iniziali del vettore).

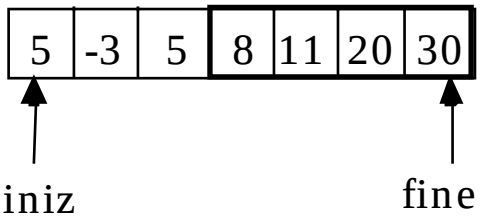
Bubble Sort: Esempio

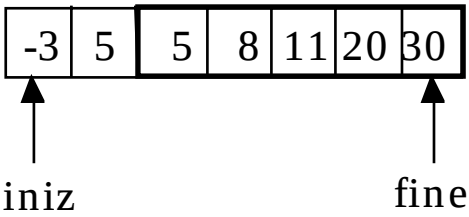












Bubble sort: realizzazione

```
void bubblesort (vettore v, int iniz, int
fine)
{ int SCAMBIO, I;
  int temp;
do
   {SCAMBIO = false;
   for (I = iniz; I < fine; I++)
  {if (v[I] > v[I+1])
    { SCAMBIO = true;
      temp = v[I];
      v[I] = v[I+1];
      v[I+1] = temp;
    }
  }
   }
while (SCAMBIO);
}
```

☞ ogni “passata” ha come effetto la collocazione nella sua posizione definitiva di un elemento:

- la prima scansione pone il valore massimo in ultima posizione
- la seconda colloca il massimo tra gli elementi rimanenti nella penultima posizione
- etc.

➠ ad ogni scansione e` possibile ridurre il vettore alla parte non ancora ordinata.

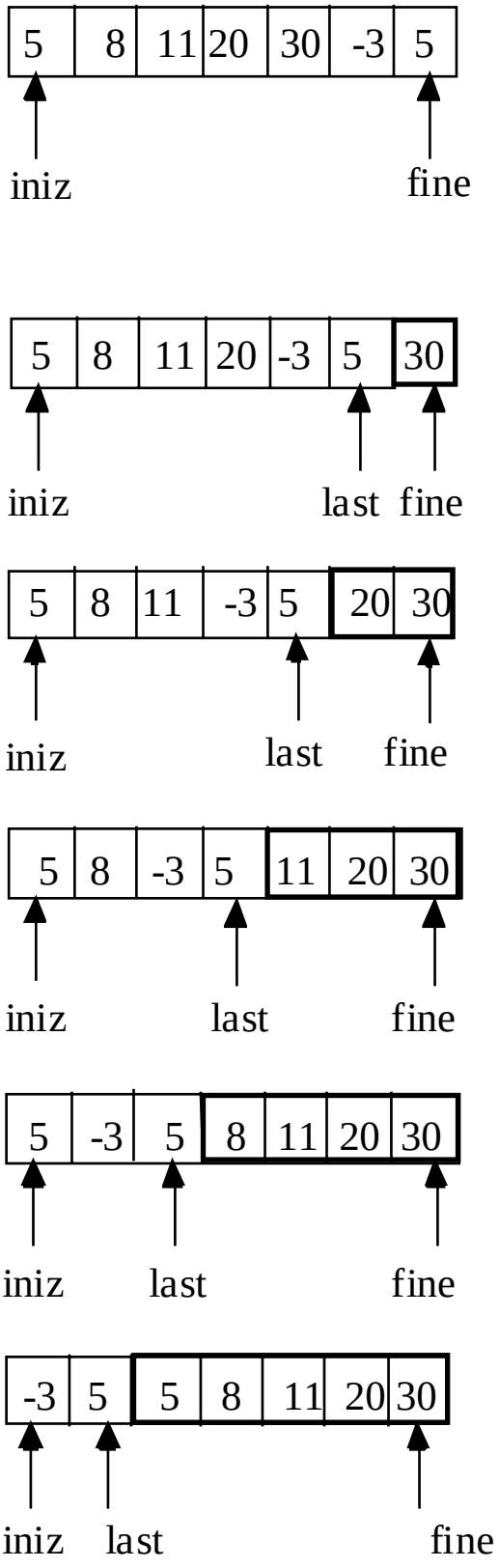
Bubble sort ottimizzato:

Utilizza una variabile ausiliaria per tenere traccia della posizione in cui è stato effettuato l'ultimo scambio.

Ad ogni iterazione si esclude la parte finale del vettore già ordinata.

```
void bubble_opt(vettore v, int iniz, int
fine)
{ int I, limit, last;
  float     temp;
 fine--;
 limit=fine;
 while (limit>iniz)
  {last=iniz;
   for (I = iniz; I <= limit; I++)
   {if (v[I] > v[I+1])
      { temp = v[I];
        v[I] = v[I+1];
        v[I+1] = temp;
        last = I;
      }
  }
   limit=last;
}
}
```

Se non ci sono stati scambi, limit==iniz alla fine dell'esecuzione del corpo del ciclo.


5 8 11 20 30 -3 5
iniz
fine
5 8 11 20 -3 5 30
iniz
last fine
5 8 11 -3 5 20 30
iniz
last fine
5 8 -3 5 11 20 30
iniz
last fine
5 -3 5 8 11 20 30
iniz
last fine
-3 5 5 8 11 20 30
iniz
last fine

Osservazioni sull'algoritmo bubble sort:

Non sono sempre necessarie n-1 iterazioni. Se non avviene alcuno scambio, l'algoritmo termina.

dipende dai valori dei dati di ingresso.

Caso migliore: Vettore gia` ordinato.
Una sola iterazione, con (n-1) confronti e nessuno scambio.

Caso peggiore: Vettore ordinato in senso decrescente.
Al passo di iterazione i, (n-i) confronti ed (n-i) scambi.

✐ Esercizio:

Scrivere una versione ricorsiva dell'algoritmo di ordinamento a bolla.

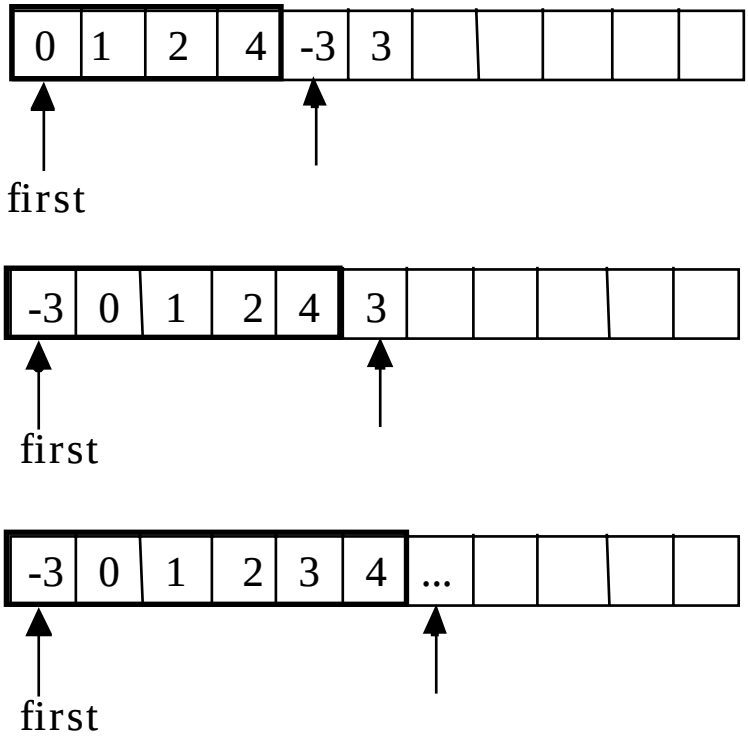
Bubble Sort Ricorsivo

Soluzione:

```
void bubble_ric(vettore v, int iniz, int
fine)
{
int i, last, temp;
last=iniz;
for (i=iniz; i<fine; i++)
  if (v[i] > v[i+1])
      { temp = v[i];
       v[i] = v[i+1];
       v[i+1] = temp;
       last=i;
       }
if (last>iniz)
  bubble_ric(v, iniz, last);
  else return;
}
```

Insert sort

L'ordinamento e` ottenuto costruendo un sotto-vettore ordinato, a partire dalla prima componente. In questo sotto-vettore gli elementi sono inseriti ordinatamente e l'inserimento e` ottenuto attraverso "shift" a destra dei restanti elementi del vettore.



Insert Sort

Realizzazione:

```
void insert_sort (vettore v)
{  int i, j, el, pos;
   boolean trovato;

   for (i=1; i<N; i++)
    {  trovato=false;
       el=v[i];
       for (j=0; (j<=i)&&!trovato; j++)
          if (el<=v[j])
             {trovato=true;
               pos=j;
              }
       if (trovato) /* shift */
          for(j=i; j>pos; j--)
             v[j]=v[j-1];
       v[pos]=el;
    } /* for i*/
}
```

Osservazioni sull'insert sort:

Caso migliore: Vettore già ordinato (n-1 confronti)

Shell Sort

Vengono esaminate coppie di elementi (*prec* e *succ*) situati a distanza prefissata (*gap*):

- quando non e` rispettata la relazione d'ordine (*prec* > *succ*), i due elementi vengono **scambiati**.

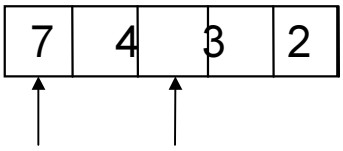
- se viene effettuato uno scambio, e` necessario ricontrollare la coppia (gia` esaminata) di cui *prec* fa parte (**retropropagazione**) ed eventualmente provvedere ad ulteriori scambi.

- al termine di ogni scansione il *gap* viene ridotto (ad esempio, dimezzato) e si ripete l'analisi.

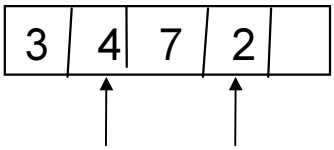
- l'algoritmo si ferma quando si raggiunge un valore di gap=1.

Esempio:

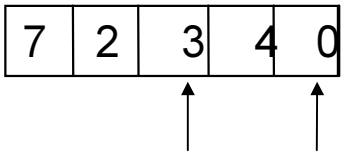
Su un vettore di 5 elementi:
Gap=2



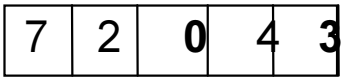
Scambio:



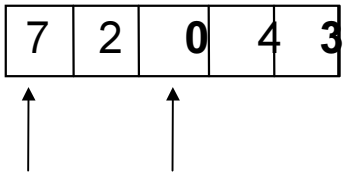
Scambio:

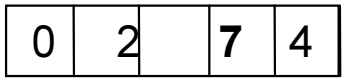


Scambio:

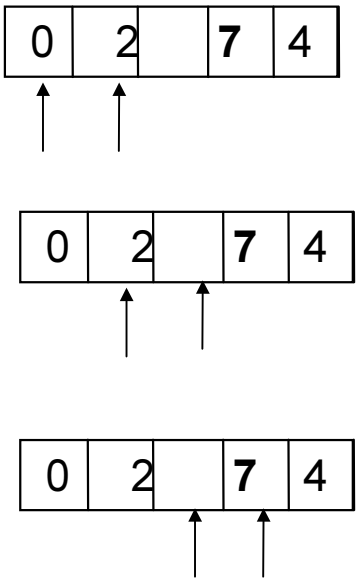


Retropropagazione e scambio:

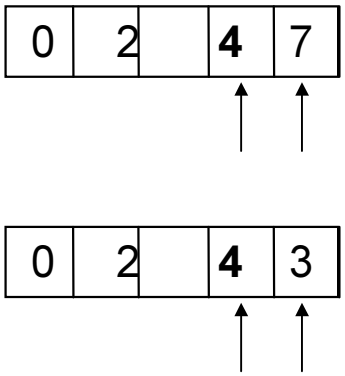




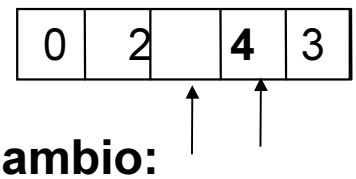
Gap=2



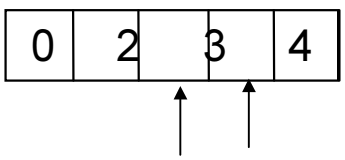
Scambio:



Scambio e retropropagazione:



Scambio:



Fine

Shell Sort

Realizzazione:

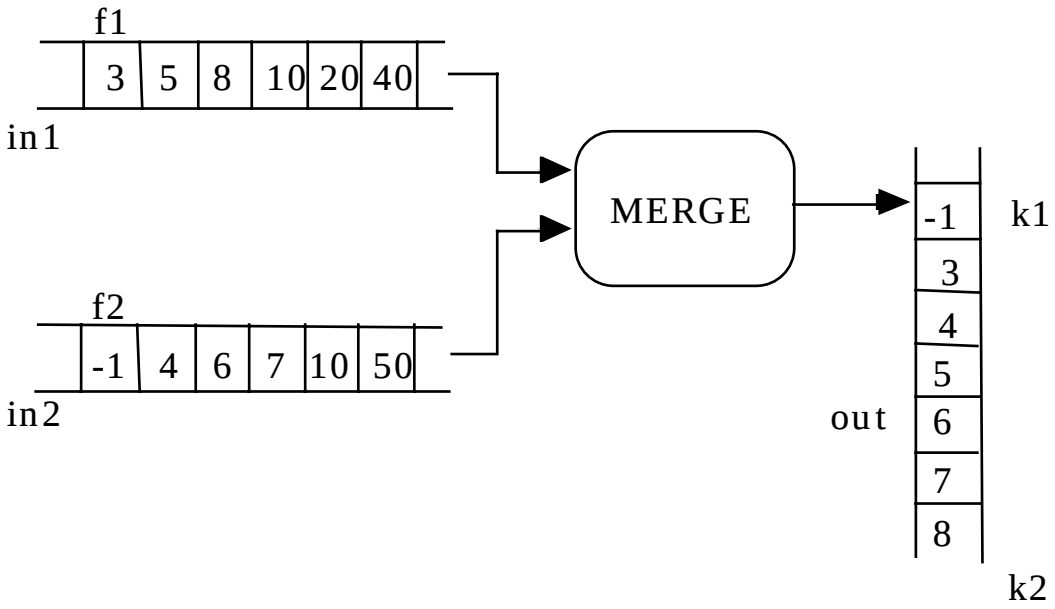
```
void shell_sort(vettore v, int iniz, int
fin)
{
  int i,prec, succ, temp,
gap=(iniz+fin)/2;
  boolean fine;
  fin--;
  while (gap>0)
    {
    for (i=gap; i<N; i++)
       {prec=i-gap; fine=false;
        do
          { succ=prec+gap;
            if (v[prec]>v[succ])
               {temp=v[prec];
                v[prec]=v[succ];
                v[succ]=temp;
          /*retropropagazione*/
                prec=prec-gap;
                }
              else fine=true;
           }while((!fine) &&(prec>=0));
          }
       gap/=2;
    }
}
```

Merge sort (ordinamento per fusione)

Utilizza, al proprio interno, l'algoritmo di ***fusione*** (o ***merge*****)**.

Merge:

Dati due vettori x, y ordinati in ordine crescente, con m componenti ciascuno, produrre un unico vettore z, di 2*m componenti, ordinato.



- Si scandiscono i due vettori di ingresso, confrontandone le componenti a coppie.
- Se in1[i] <= in2[j], out[k]=in1[i] (scrivi nella componente corrente del vettore out in1[i]); altrimenti, out[k]=in2[j].

Merge

Indici i,j per scandire in1 e in2, indice k per scrivere su out.

Si confrontano in1[i] e in2[j]:

- se in1[i] <= in2[j], scrive in1[i] nella componente k-esima di out (incrementa i, k);
- altrimenti, scrive in2[j] nella componente k-esima di out (incrementa j, k).

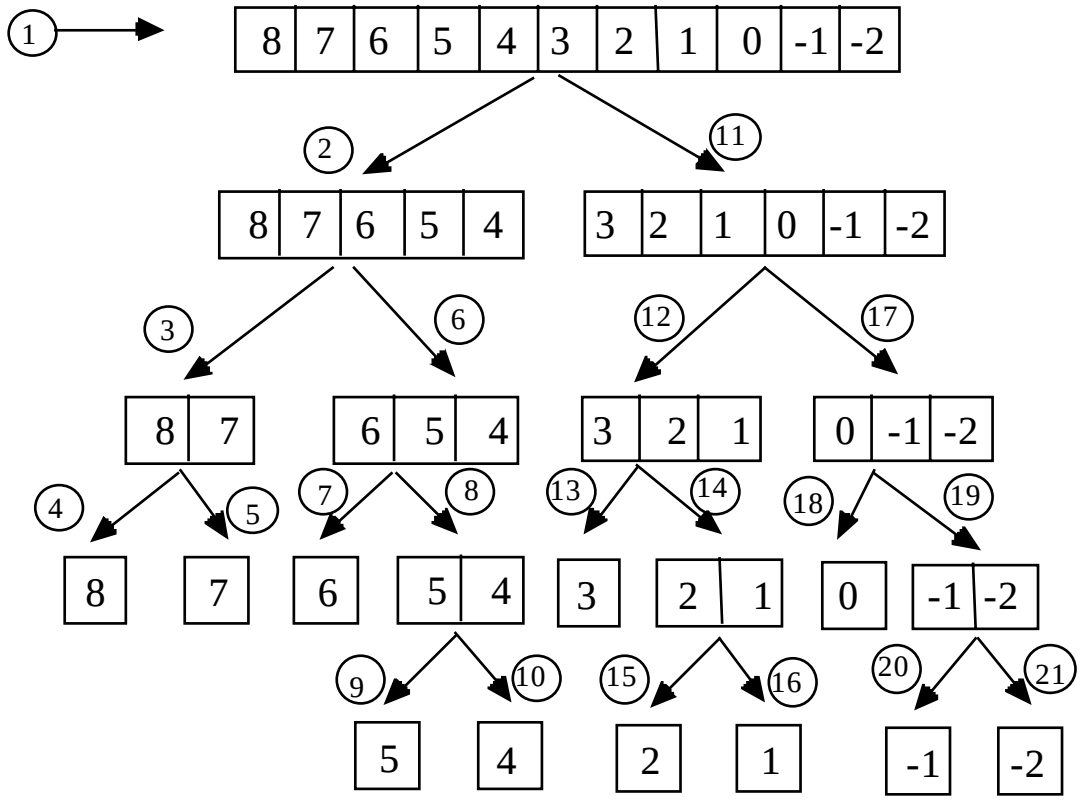
Se la scansione di uno dei vettori e` arrivata all'ultima componente, si copiano i rimanenti elementi dell'altro nel vettore out.

Merge sort (ordinamento per fusione):

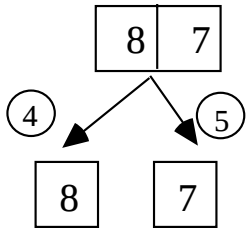
E` un algoritmo ***ricorsivo***.

Il vettore di ingresso viene diviso in due sotto-vettori sui quali si richiama il merge sort.

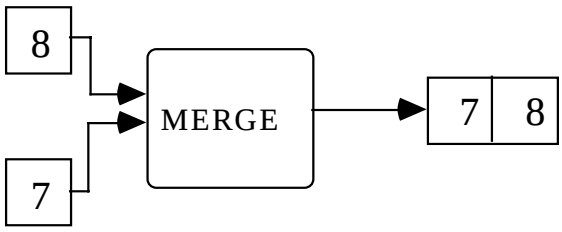
Quando ciascun sotto-vettore e` ordinato, i due vengono “fusi” attraverso la procedura di merge.



Dopo:



merge dei due sotto-vettori ad una componente:



In teoria e` il “migliore” algoritmo di ordinamento (abbiamo assunto pero` nullo il costo di attivazione di una procedura).

Mergesort

Realizzazione:

```
void merge (vettore v, int iniz1, int
iniz2, int fine);

void mergesort (vettore v, int iniz, int
fine)
{
  int m;
  if ( iniz < fine)
   {  m = (fine + iniz) / 2;
      mergesort  (v, iniz , m);
      mergesort  (v, m +1, fine);
      merge (v, iniz, m + 1,fine);
   }
}
```

```
   void merge (vettore v, int iniz1, int
   iniz2, int fine)
   /* fusione di due vettori */
   {  vettore vout; /*vett. temporaneo*/

   int i, j, k;

   i = iniz1; j = iniz2; k = iniz1;
   /*confronto: */
   while (( i <= iniz2 -1) && ( j <= fine ))
     { if (v [i] < v [j])
         {  vout [k] = v[i];
            i= i + 1; }
       else
         {  vout [k] = v[j];
            j= j + 1; }
      k = k + 1;
     }

   /* fasi di trattamento del vettore non
   terminato */
   while ( i <= iniz2 -1)
     {   vout [k] = v[i];
      i= i + 1;
      k = k + 1; }
   while ( j <= fine )
     {   vout [k] = v[j];
      j= j + 1; k = k + 1;}

   /* copia da vout in uscita */
   for (i = iniz1; i<= fine; i=i+1)
      v[i] = vout [i];
   }
}
```

Quick sort

Come merge-sort, suddivide il vettore in due sotto-vettori, delimitati da un elemento "sentinella" (***pivot***).

L'**obiettivo** e` di avere nel primo sotto-vettore solo elementi minori o uguali al pivot, nel secondo sotto-vettore solo elementi maggiori.

Per raggiungere l'obiettivo:

Si determina arbitrariamente un pivot (ad esempio pivot=V[N-1])

Si scandisce il vettore dato mediante due indici:

- i, che parte da 0 e procede in **avanti**
- j, che parte da N-1 (N=dimensione del vettore) e procede all'**indietro**

Scansione in avanti:

ogni elemento V[i] viene confrontato con il pivot; se V[i]> pivot, la scansione in avanti si ferma e si passa alla

Scansione all'indietro:

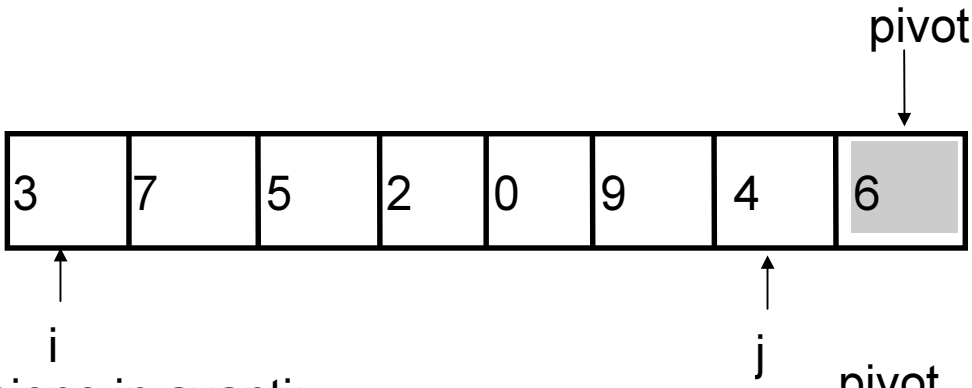
ogni elemento V[j] vene confrontato con il pivot; se V[j]< pivot, la scansione in avanti si ferma e l'elemento V[j]viene scambiato con V[i]

Poi si riprende con la scansione avanti, indietro, etc.; Il tutto si ferma quando **i==j**. A questo punto si scambia V[i] con il pivot

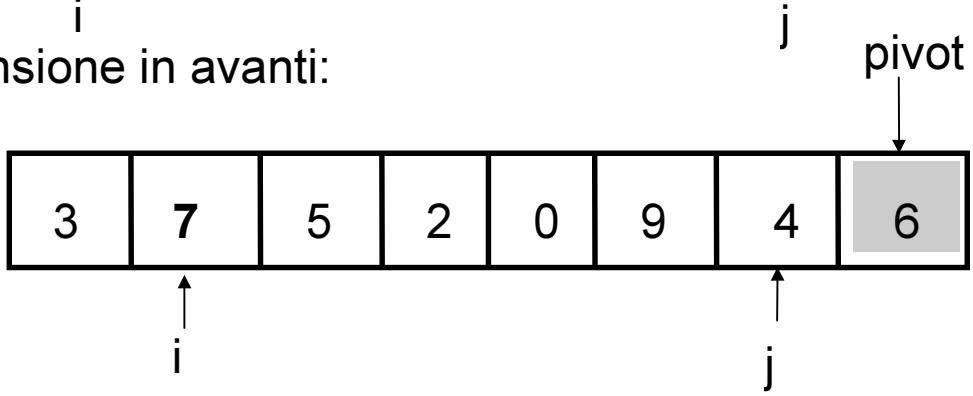
Alla fine della scansione il pivot e` collocato nella sua posizione definitiva.

L'algoritmo e` ***ricorsivo***: si richiama su ciascun sotto-vettore fino a quando non si ottengono sotto-vettori con un solo elemento.

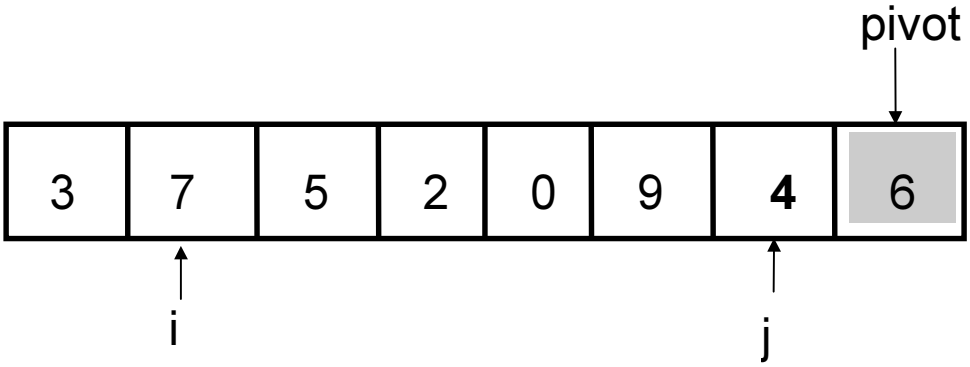
A questo punto il vettore iniziale risulta ordinato.



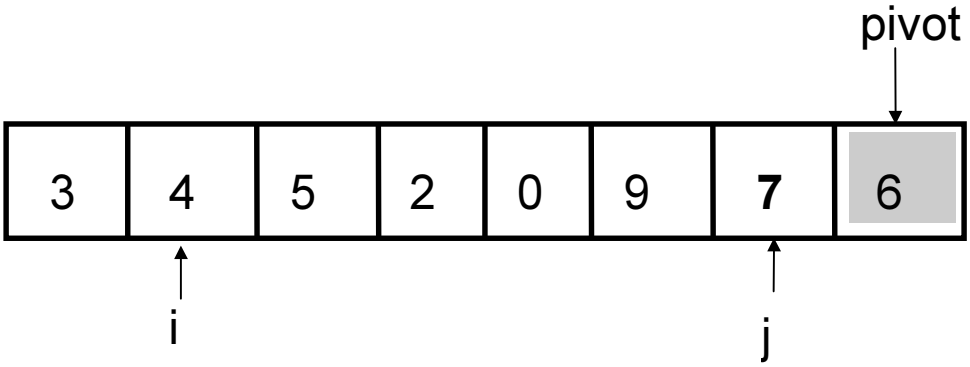
Scansione in avanti:



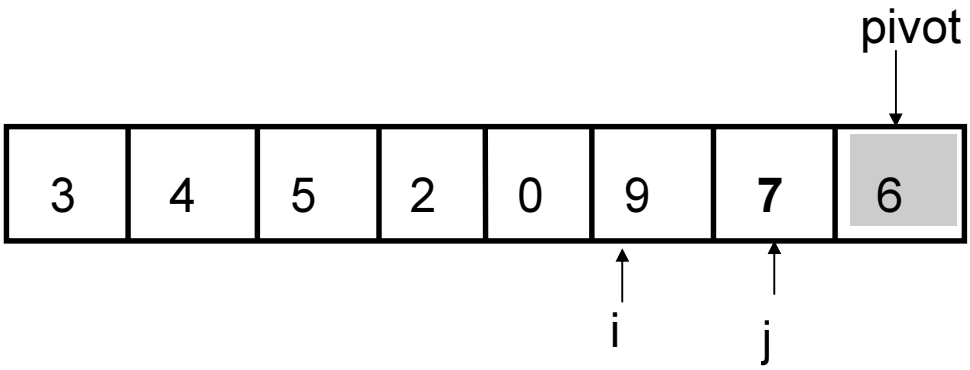
Scansione all'indietro:



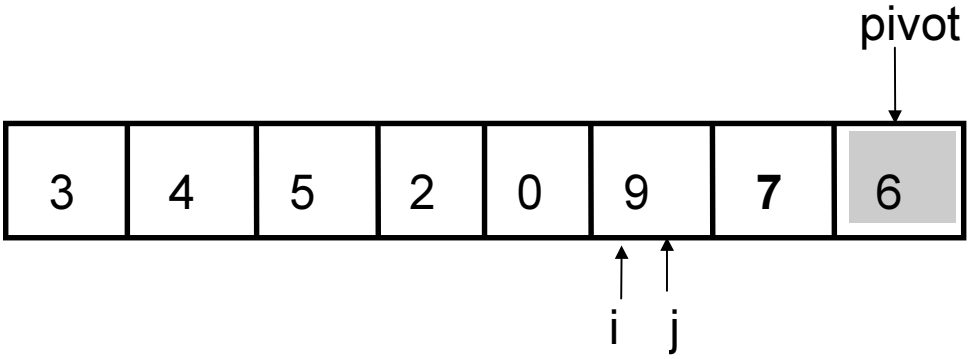
Scambio:



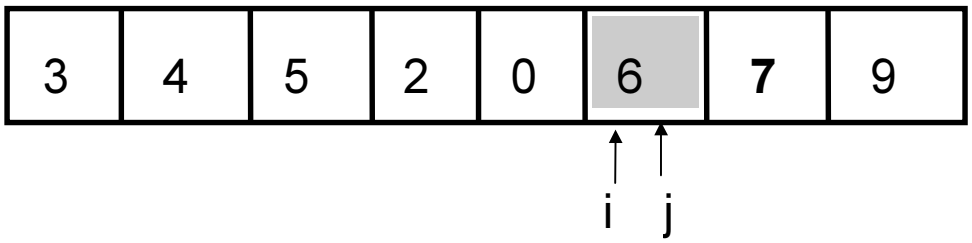
Scansione all'avanti:



Scansione all'indietro:



Fine scansione: sacmbio il pivot con V[i]:



Il pivot e` nella posizione definitiva: ripeto il procedimento sui due sottovettori:

- V[0, i-1]
- V[i+1, N-1]

Quick Sort

Realizzazione:

```
void quicksort (vettore v, int iniz, int
fine)
{int i, j, ipivot, pivot, temp;
 if ( iniz < fine )
  {  i=iniz; j=fine; ipivot=fine;
   pivot = v[ipivot];
   do  /* trova il pivot */
   {while ((i < j)&&(v[i]<=pivot))
         i = i + 1;
   while ((j > i)&&(v[j]>=pivot))
         j = j - 1;
   if (i<j) { temp = v[i];
         v[i] = v[j];
         v[j] = temp; }
  }
  while (i < j);
/* determinati i due sottoinsiemi */
/* posiziona il pivot */
   if ((i!=ipivot)&&(v[i]!=v[ipivot]))
   {  temp = v[i];
      v[i] = v[ipivot];
      v[ipivot] = temp;
      ipivot=i;
   }
  /* ricorsione sulle sottoparti*/
  if (iniz < ipivot - 1)
   quicksort (v, iniz, ipivot - 1);
  if (ipivot + 1< fine )
   quicksort (v, ipivot + 1, fine); }
```

Esercizio:

1) Scrivere una procedura che esegua l'ordinamento per righe di una matrice quadrata. Realizzare la procedura di ordinamento per colonne (si puo` ottenere dalla precedente sulla matrice trasposta). Chiamare le procedure con un programma di prova.

2) Scrivere una procedura che esegua il merge di due file di interi ordinati su un terzo file.

3) Realizzare un programma per ordinare un file di interi. Utilizzare una variabile buffer (vettore) di appoggio in memoria centrale.

Merge di file ordinati: soluzione

```
void mergefile (FILE *f1, FILE *f2, FILE
*f3)
{ int x1, x2;

fread(&x1, sizeof(int), 1, f1);
fread(&x2, sizeof(int), 1, f2);
while (( !feof(f1)) && ( !feof(f2)))
  {
   if (x1 < x2)
   {  fwrite(&x1, sizeof(int), 1, f3);
      fread(&x1, sizeof(int), 1, f1);
   }
   else
   {
      fwrite(&x2, sizeof(int), 1, f3);
      fread(&x2, sizeof(int), 1,f2);
   }
  }

 while (!feof(f1))
  {
   fwrite(&x1, sizeof(int), 1, f3);
   fread(&x1, sizeof(int), 1, f1);
  }

 while (!feof(f2))
  {fwrite(&x1, sizeof(int), 1, f3);
   fread(&x2, sizeof(int), 1, f2);
  }
}
```

Uso di mergefile:

```
void mergefile (FILE *f1, FILE *f2, FILE
*f3);

main()
{ FILE *f1, *f2, *f3;

f1=fopen("file1.dat", "rb");
f2=fopen("file2.dat", "rb");
f3=fopen("fileris.dat", "wb");

mergefile (f1, f2, f3);

fclose(f1);
fclose(f2);
fclose(f3);
}
```