Ordinamenti ricorsivi
Selection Sort ricorsivo

- Il metodo del selection sort può essere facilmente realizzato in modo ricorsivo
- si definisce una procedura che ordina (ricorsivamente) la porzione di array individuata da due indici `from` e `to`
- il minimo elemento della porzione viene messo in posizione `from` per poi ordinare ricorsivamente la porzione tra `from+1` e `to`
- Il caso base corrisponde all'ordinamento di una porzione fatta da un solo elemento (è già ordinata)

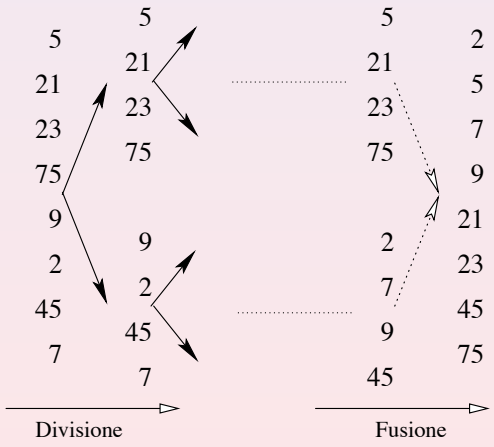
```
void SelectionSort(int v[], int from, int to){
  if (from < to) {
    int min = minPos(v,from,to);
    swap(v+from, v+min);
    SelectionSort(v, from+1, to);
  } }

void sort(int v[], int dim) {
    SelectionSort(v,0,dim-1);
}
```

Merge sort

Si divide il vettore da ordinare in due parti:

- si ordina ricorsivamente la prima parte
- si ordina ricorsivamente la seconda parte
- si combinano (operazione di fusione, merge) le due parti ordinate

Esempio:



Esprimiamo il procedimento in uno pseudo-linguaggio

```
ordina per fusione gli elementi di A da from a to

IF from < to (c'è più di un elemento tra from e to)

THEN
     mid =  (from + to) /2
     ordina per fusione gli elementi di A da from a mid
     ordina per fusione gli elementi di A da mid +1 a to
     fondi
        gli elementi di A da from a mid con
        gli elementi di A da mid+1 a to
        restituendo il risultato nel sottovettore
         di A da from a to
```

Implementiamo l'algoritmo in C, definendo una procedura ricorsiva

`void mergeRicorsivo(int A[], int from, int to)`

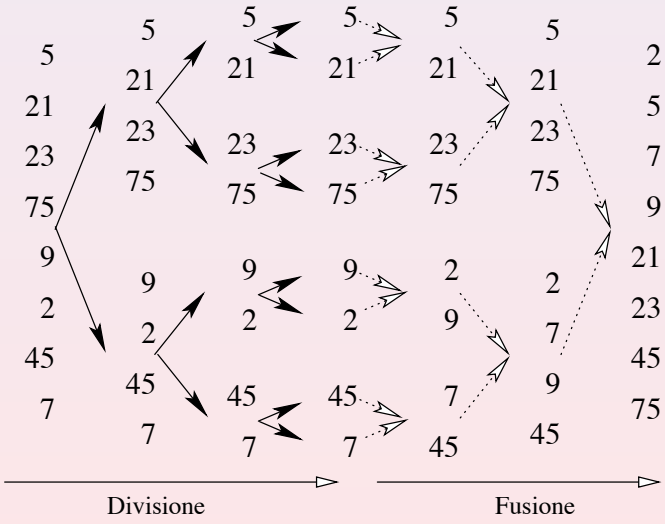
che ordina la porzione dell'array `A` individuata dagli indici `from` e `to`.

```
void mergeRicorsivo(int A[], int from, int to)

{
  int mid;

  if (from < to) {              /* l'intervallo da mid a to, estremi
                                   inclusi, comprende almeno due elementi */
    mid = (from + to) / 2;

    mergeRicorsivo(A, from, mid);
    mergeRicorsivo(A, mid+1, to);

    merge(A, from, mid, to);    /* fonde le due porzioni ordinate [from, mid],
                                   [mid+1, to] nel sottovettore [from, to]  */
  }
}
```

La procedura `mergeSort` che ordina un array di interi è semplicemente

```
void sort(int v[], int dim)
{
mergeRicorsivo(v, 0, dim-1);
}
```

Esempio:



- Vediamo l'operazione di *fusione*, definendo la procedura

  `void merge(int A[], int from, int mid, int to)`

  che fonde le due porzioni dell'array `A` con indici compresi tra `from` e `mid` e tra `mid+1` e `to`.
- La procedura utilizza un array di supporto `B`: per semplicità, supponiamo di avere una costante `LUNG` che definisce la lunghezza degli array che stiamo trattando.

```
void merge(int A[], int from, int mid, int to)
 {
  int B[LUNG];                              /* vettore di appoggio */
  int primo, secondo, appoggio, da_copiare;

  primo = from;
  secondo = mid + 1;
  appoggio = from;

  while (primo <= mid && secondo <= to) { /* copia in modo ordinato      */
    if (A[primo] <= A[secondo]) {         /* gli elementi della prima e  */
      B[appoggio] = A[primo];             /* della seconda porzione in B */
      primo++;                            /* fino ad esaurire una delle due */
    }
    else {
      B[appoggio] = A[secondo];
      secondo++;
    }
    appoggio++;
  }
```

```
 if (secondo > to)              /* e' finita prima la seconda porzione */
 /* copia da A in B tutti gli elementi della
    prima porzione fino a mid */

    for (da_copiare = primo; da_copiare <= mid; da_copiare++) {
       B[appoggio] = A[da_copiare];
       appoggio++;
    }
  else                           /* e' finita prima la prima porzione */

    for (da_copiare = secondo; da_copiare <= to; da_copiare++) {
    /* copia da A in B tutti gli elementi della
    /* seconda porzione fino a to */

       B[appoggio] = A[da_copiare];
       appoggio++;
    }

/* ricopia tutti gli elementi da from a to da B ad A */
  for (da_copiare = from; da_copiare <= to; da_copiare++)
    A[da_copiare] = B[da_copiare];
}
```

Ricerca binaria in un vettore ordinato

Per cercare un elemento in un vettore ordinato, è possibile sfruttare l'ordinamento dimezzando ad ogni passo l'intervallo di elementi ancora da considerare per la ricerca.

Cerchiamo un elemento el in un vettore ordinato A:

- usiamo due indici from e to per delimitare l'intervallo di ricerca
- confrontiamo el con l'elemento di A al centro dell'intervallo, ovvero con A[mid], dove mid = (from+to)/2.
  Distinguiamo 3 casi:
  el = A[mid] $\Longrightarrow$ l'elemento è stato trovato
  el < A[mid] $\Longrightarrow$ cerchiamo nell'intervallo con indici da from a mid−1
  el > A[mid] $\Longrightarrow$ cerchiamo nell'intervallo con indici da mid+1 a to
- se non troviamo el ci fermiamo quando l'intervallo di ricerca si è annullato (ovvero, from > to)

Esempio: Ricerca di 24 nel seguente vettore ordinato.

14	19	24	31	33	40	45	52	60
from								*to*
				mid				
from			*to*					
	mid							
		from	*to*					
		mid						

ricerca el nella parte di A compresa tra from e to
if to $<$ from
then l'elemento non compare in A
else mid $=$ (from+to)/2 (divisione intera)
if el $=$ A[mid]
then l'elemento è stato trovato
el $<$ A[mid]
then ricerca el nella parte di A tra from e mid-1
else ricerca el nella parte di A tra mid+1 e to

```
int RicercaBinariaRic(int A[], , int el, int from, int to)

/* Funzione ricorsiva che restituisce la posizione di el nella parte di A
    compresa fra from e to; restituisce -1 se el non sta nella porzione*/

{ int mid, posizione;

  if (from > to)
   posizione = -1;          /* la parte del vettore tra from e to e' vuota */
 else {
   mid = (from + to) / 2;
   if (el == A[mid]) {                    /* l'elemento e' stato trovato */
     posizione = mid;
   }
   else
     if (el < A[mid])                     /* cerca nella parte inferiore */
       posizione = RicercaBinariaRic(A, el, from, mid-1);
     else                                 /* cerca nella parte superiore */
       posizione = RicercaBinariaRic(A, el, mid+1, to);
 }
 return posizione;
}
```

```
int RicercaBinaria(int A[], int el, int n)
 /* Effettua la ricerca binaria di el nei primi n elementi di A, usando la
    funzione ricorsiva RicercaBinariaRic.  Restituisce la posizione di el se
    questo e' presente in A, -1 altrimenti.  */
{
 return RicercaBinariaRic(A, el, 0, n-1);
 }
```