La ricerca dicotomica - 1

- Per "cercare" un elemento in un vettore ordinato esiste un metodo detto ricerca binaria o dicotomica
  - Si confronta il valore **val** da ricercare con l'elemento centrale del vettore **A[length/2]**
  - Se **val** è minore dell'elemento mediano, si ripete la ricerca sulla metà sinistra del vettore, altrimenti si ricerca nella metà destra

La ricerca dicotomica - 2

- Esempio: ricerca del numero 23

Si confronta 23 con 13

Ci si concentra sulla metà destra (da ind. 8 a ind. 14): si confronta 23 con 27

Ci si concentra sulla metà sinistra (da ind. 8 a ind. 10): si confronta 23 con 20

Ci si concentra sulla metà destra (da ind. 9 a ind. 9): trovato!!

Implementazione della ricerca binaria ricorsiva

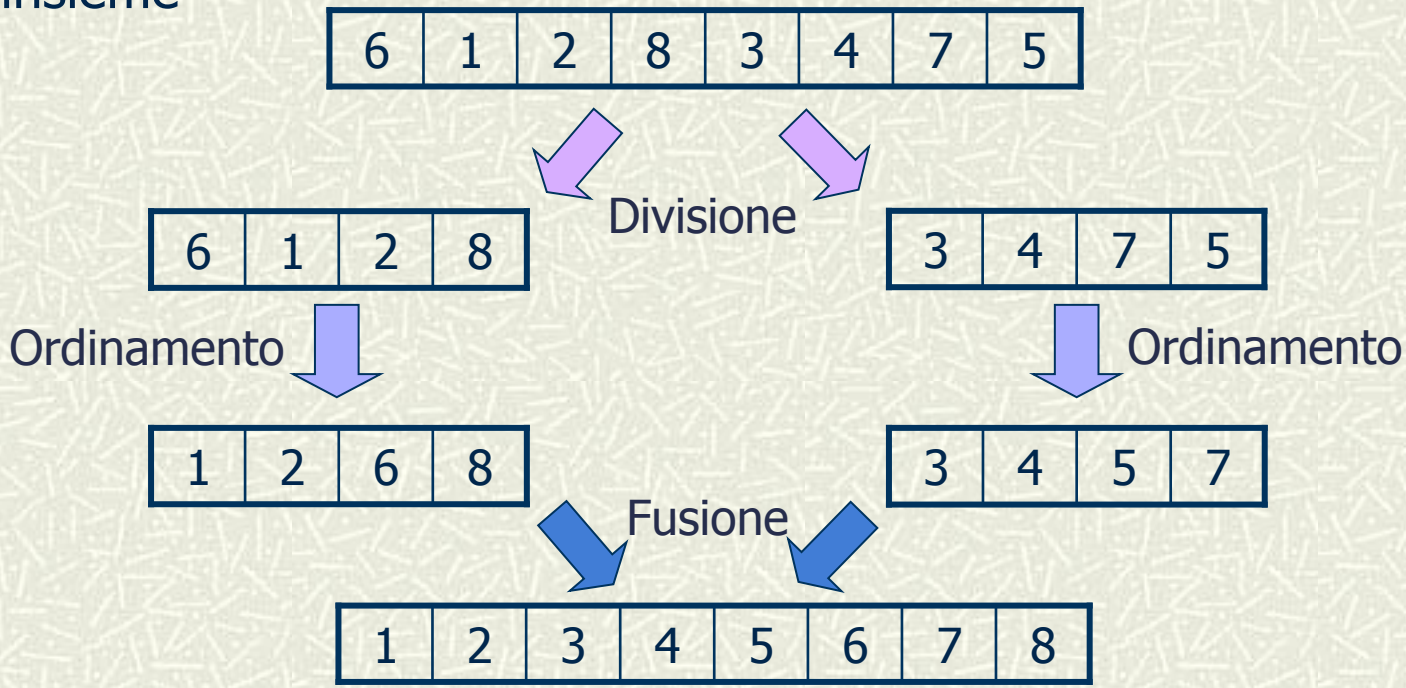
```
int binarySearch(int vet[], int dim, int el) {
int startPos;
int med = dim / 2;
if (vet[med] == el)
        return med;
if (med == 0)
        return -1;
if (el < vet[med])
{
        return binarySearch(vet, med, el);
}
else {
    startPos = med + 1;
    return startPos +
    binarySearch(&vet[startPos], dim - startPos, el);
 }
}
```

Mergesort - 1

- Il **Mergesort** è un algoritmo basato sul paradigma del *divide et impera*
- Una strategia ***divide et impera*** consiste nel suddividere un problema in sottoproblemi, nel risolvere i sottoproblemi, e nel ricomporli per ottenere la soluzione del problema originale
- Il Mergesort è composto da due fasi:
  - una fase di divisione del vettore da ordinare in sottovettori
  - una fase di ricomposizione dei risultati (merge)

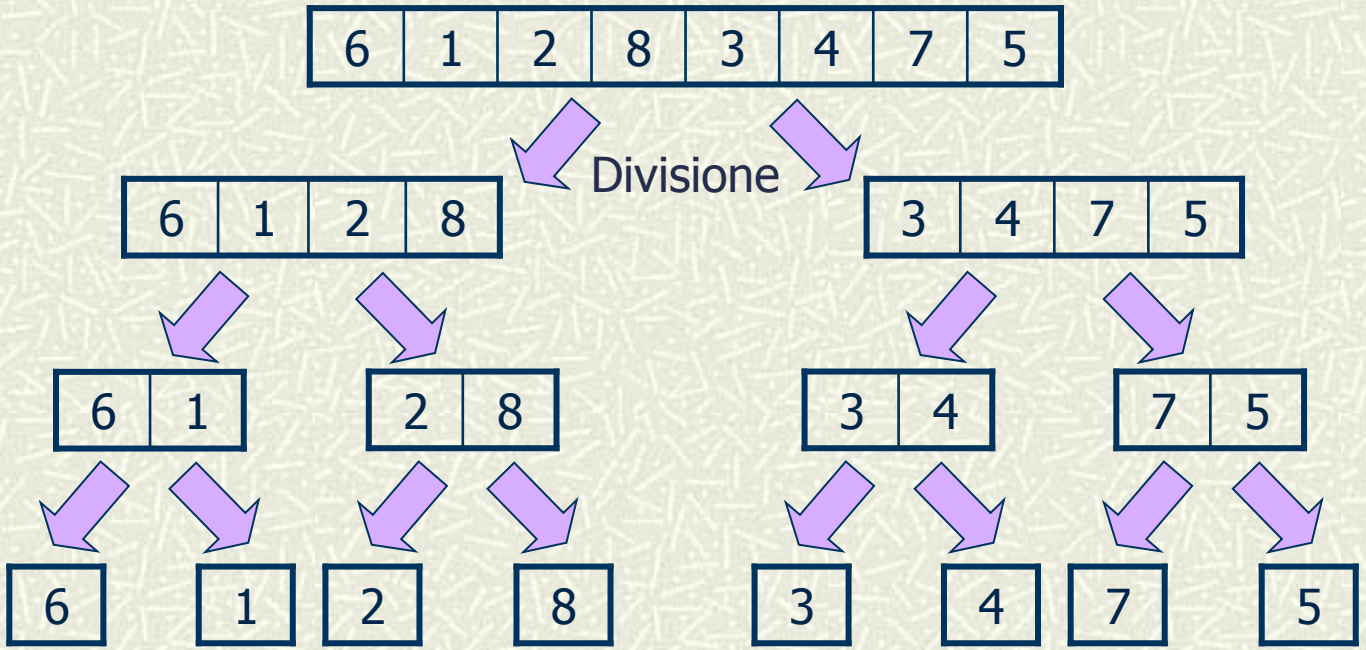
Mergesort - 2

- Idea
  - Dato un vettore da ordinare, lo si divide in due sottovettori di ugual dimensione, si ordinano i sottovettori e poi si "fondono" insieme



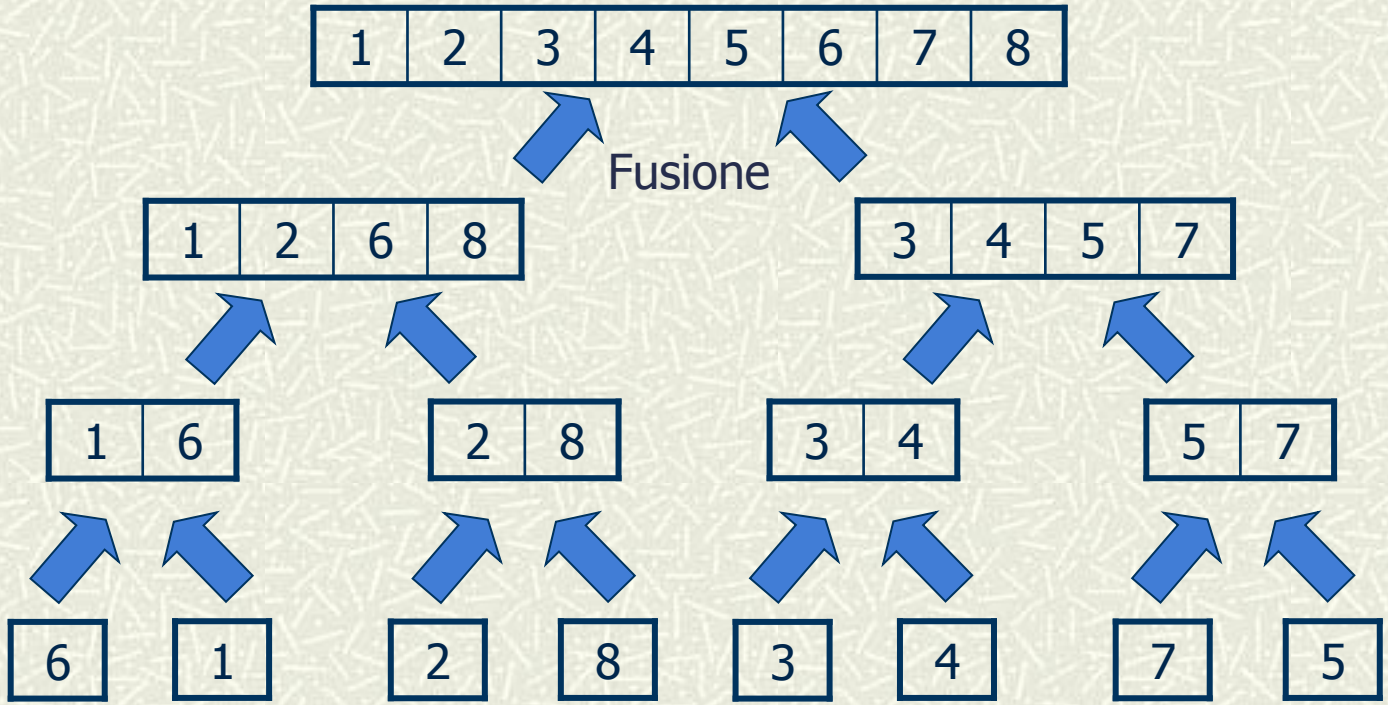
Mergesort: la divisione ricorsiva

- Come si ordinano i due sottovettori ?
  - Applicando **ricorsivamente** la divisione fino a quando il vettore contiene un solo elemento: in tal caso l'ordinamento è banale



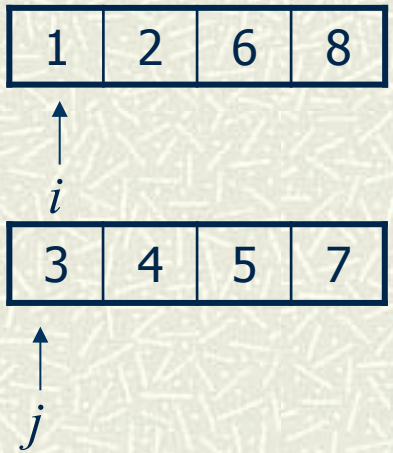
Mergesort: la fusione ricorsiva - 1

- I sottovettori ordinati verranno poi ricorsivamente fusi



Mergesort: la fusione ricorsiva - 2

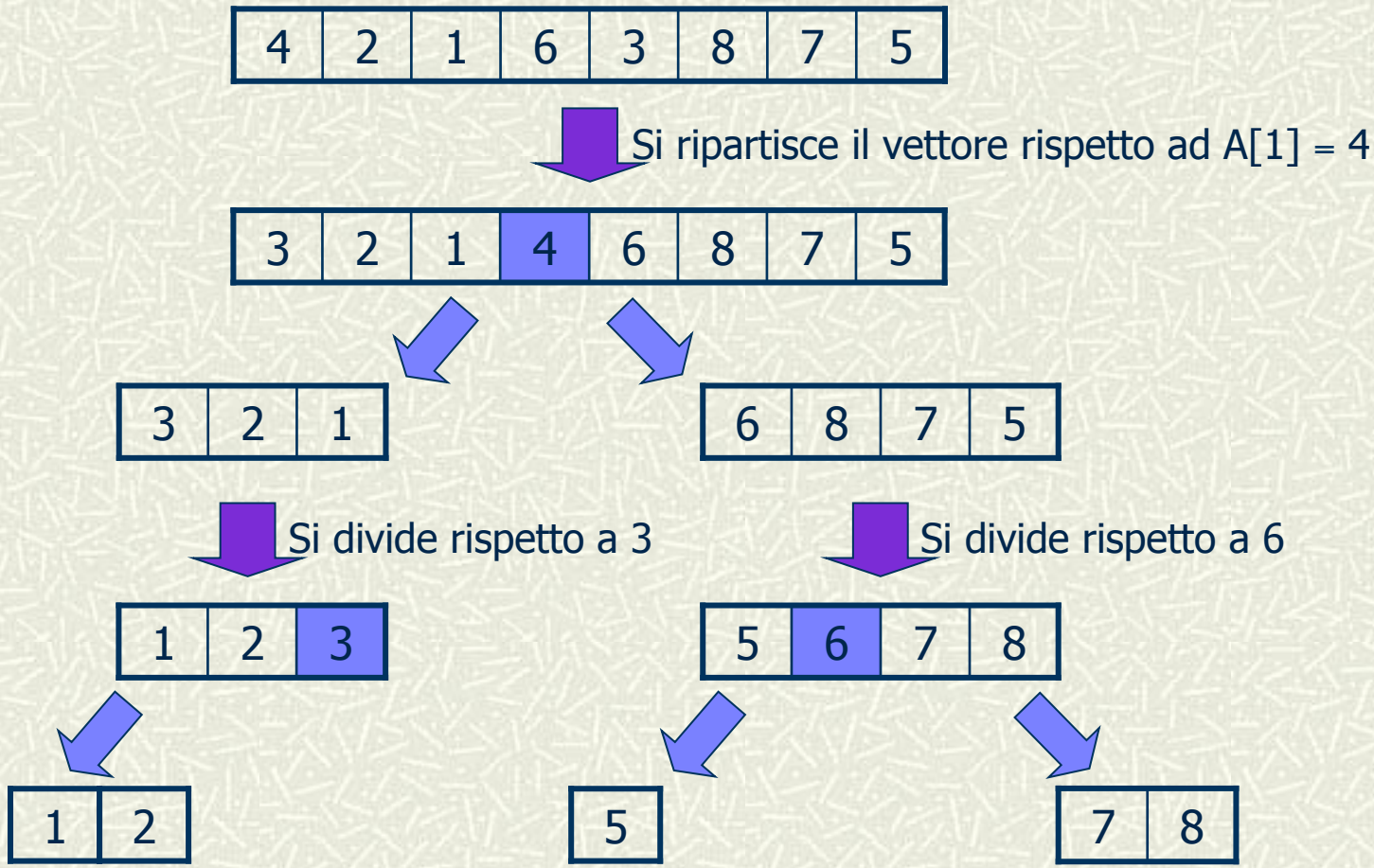
- La fusione viene realizzata utilizzando due indici che scorrono i due sottovettori da fondere:
  1. Ad ogni passo si confrontano i due elementi indicati dagli indici i e j, A[i], A[j]
  2. Si copia l'elemento minore in un vettore d'appoggio e si incrementa l'indice corrispondente
  3. Si torna al passo 1. fino a quando i due vettori non sono stati completamente visitati



Quicksort - 1

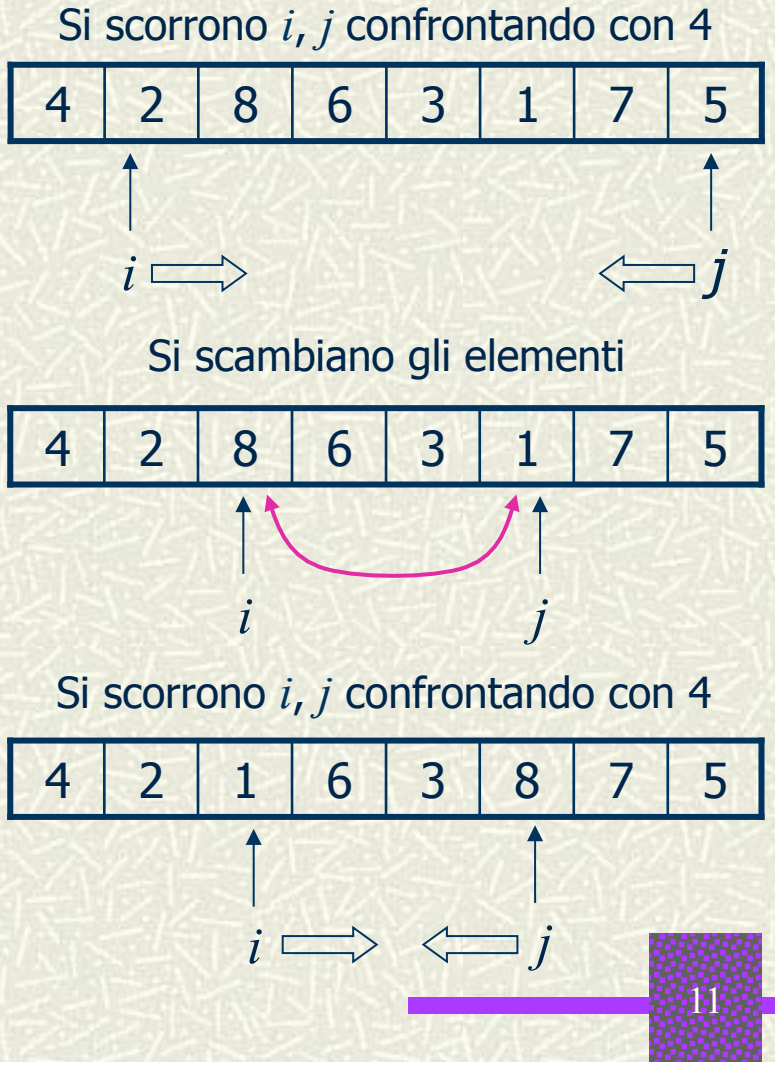
- **Quicksort**, come Mergesort, è un algoritmo ***divide et impera***
- **Idea**
  - Si partiziona il vettore A in due sottovettori, che contengono rispettivamente tutti gli elementi maggiori e minori di (per esempio) A[0], cioè il primo elemento del vettore - detto **pivot** o **perno**
  - Si **ripete** ricorsivamente la partizione…

Quicksort - 2


4 2 1 6 3 8 7 5
Si ripartisce il vettore rispetto ad A[1] = 4
3 2 1 4 6 8 7 5
3 2 1
6 8 7 5
Si divide rispetto a 3
Si divide rispetto a 6
1 2 3
5 6 7 8
1 2
5
7 8

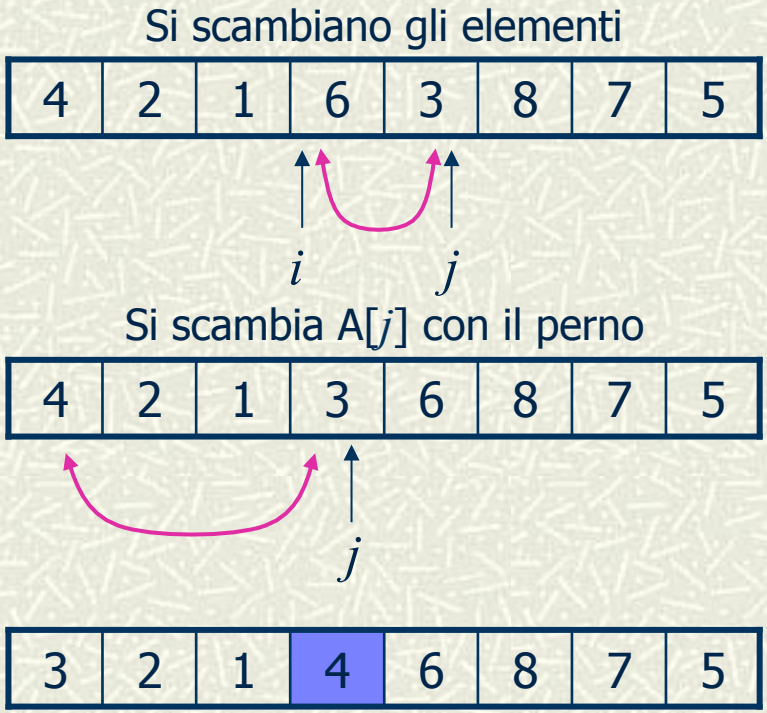
Quicksort: l'operazione pivot- 1

- Come si divide il vettore?
  - Si usano due indici i, j che scorrono il vettore da sinistra e da destra, rispettivamente
  - L'indice i scorre fino a quando A[i]<A[1]
  - L'indice j scorre fino a quando A[j]>A[1]
  - Si effettua lo scambio fra A[i] e A[j] e quindi si procede come sopra



Quicksort: l'operazione pivot- 2

- Alla fine si scambia il perno con l'elemento in posizione j



Implementazione

```
function quicksort('array')
    if length('array') ≤ 1
      return 'array'  // an array of zero or one elements is already sorted
    select and remove a pivot value 'pivot' from 'array'
    create empty lists 'less' and 'greater'
    for each 'x' in 'array'
      if 'x' ≤ 'pivot' then append 'x' to 'less'
      else append 'x' to 'greater'
    return concatenate(quicksort('less'), 'pivot', quicksort('greater'))
// two recursive calls
```

```
void quick_sort(int arr[20],int low,int high)
{
 int pivot,j,temp,i;
 if(low<high)
 {  pivot = low; i = low; j = high;
  while(i<j)
  {  while((arr[i]<=arr[pivot])&&(i<high))
   { i++;
   }
   while(arr[j]>arr[pivot])
   { j--;
   }
   if(i<j)
   {
         temp=arr[i];    arr[i]=arr[j];  arr[j]=temp;
   }
  }

  temp=arr[pivot];
  arr[pivot]=arr[j]; arr[j]=temp;
  quick_sort(arr,low,j-1);
  quick_sort(arr,j+1,high);
}
}
```

0	1	2	3	4	5	6
5	2	1	9	3	8	7

VETTORE DA ORDINARE
5 2 1 9 3 8 7

Pivot scelto 5

Partizione sinistra (elementi < 5) 2 1 3

Partizione destra (elementi > 5) 9 8 7

A sinistra
Pivot scelto 2

Partizione sinistra (elementi < 2) 1

Partizione destra (elementi > 2) 3

Partizione sinistra 1 2 3

A destra
Pivot scelto 9

Partizione sinistra (elementi < 9) 8 7

Partizione destra (elementi > 9)

Partizione destra 7 8 9

Partizione sinistra 1 2 3 **pivot** 5 **partizione destra** 7 8 9

Esercitazione

- Implementare un programma C che acquisica dall'utente il nome di un file di testo come argomento del main. Se il file non esiste, il programma emette un messaggio di errore e termina. Se il file esiste, ne legge il contenuto una riga alla volta, popolando un array di max 100 elementi. Se il file ha più di 100 linee, il programma termina con un messaggio di errore. Altrimenti, il programma ordina l'array in senso lessicografico (con il quicksort sviluppato da voi) e scrive su un nuovo file la sequenza ordinata delle stringhe dell'array, una linea per stringa. Il nome del nuovo file deve essere uguale al nome del file da ordinare con l'aggiunta del prefisso "sorted-".