La Ricorsione

Seconda Parte

Giselda De Vita

Definizione

Una funzione matematica è definita *ricorsivamente* quando nella sua definizione compare un riferimento a se stessa.

La *ricorsione* consiste nella possibilità di definire una funzione in termini di se stessa.

Problemi ricorsivi

Molti problemi, per loro natura, si prestano ad una definizione ricorsiva:

- Si definisce un metodo per risolvere dei sottoproblemi analoghi a quello di partenza, ma più piccoli
- Si definisce un metodo per combinare le soluzioni parziali nella soluzione del problema originario.

Divide et Impera

- (Divide) Dividere il problema in sottoproblemi più semplici
- (Impera) Risolvere i sottoproblemi separatamente ricorsivamente
- Ricombinare i sottoproblemi nella soluzione

Divide et Impera

Soluzione = Risolvi(Problema);

Risolvi(Problema):

Sottoproblema$_{1,2,3,\ldots,a}$ = Dividi(Problema);

Per ciascun Sottoproblema$_i$:

Sottosoluzione$_i$ = Risolvi (Sottoproblema$_i$);

Return Soluzione = Combina(Sottosoluzione$_{1,2,3,\ldots,a}$);

Divide et impera in pratica

Molti algoritmi importanti e universalmente utilizzati si basano sulla tecnica divide et impera.

Vedremo la ricerca binaria e l'algortimo di ordinamento Merge Sort.

La ricerca binaria

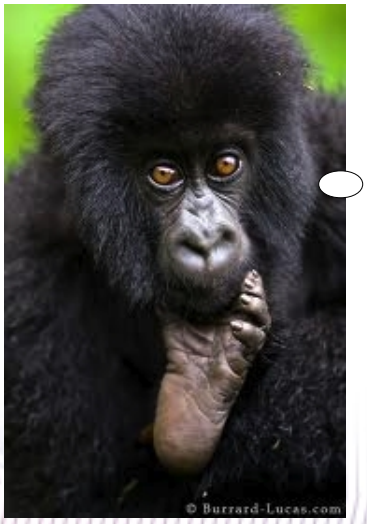
Problema: Trovare un elemento in un array ordinato

Ho la soluzione!
Scorro tutto l'array
finché non lo trovo!

E allora a che
serve il fatto che
sia ordinato?

La ricerca binaria

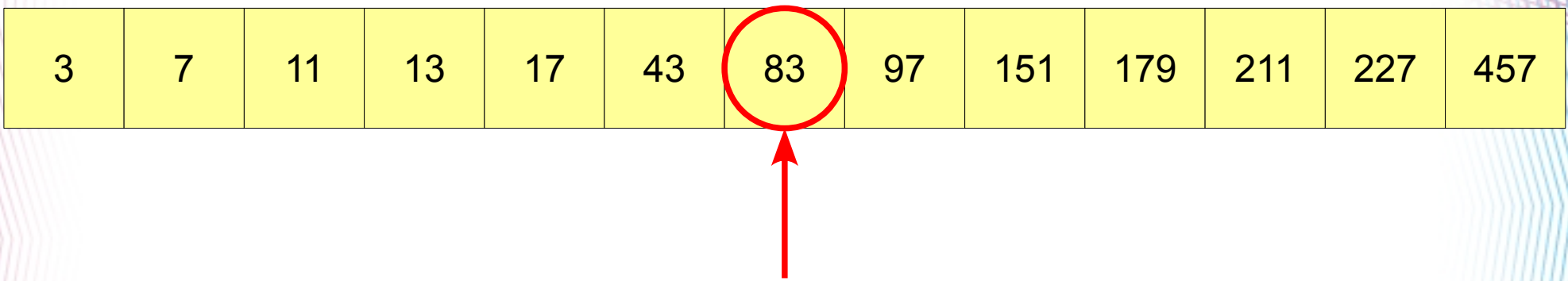
Idea: invece di procedere sequenzialmente, parto dalla metà.



Dalla metà?!?

La ricerca binaria

Ho un array ordinato e un elemento da trovare: **35**



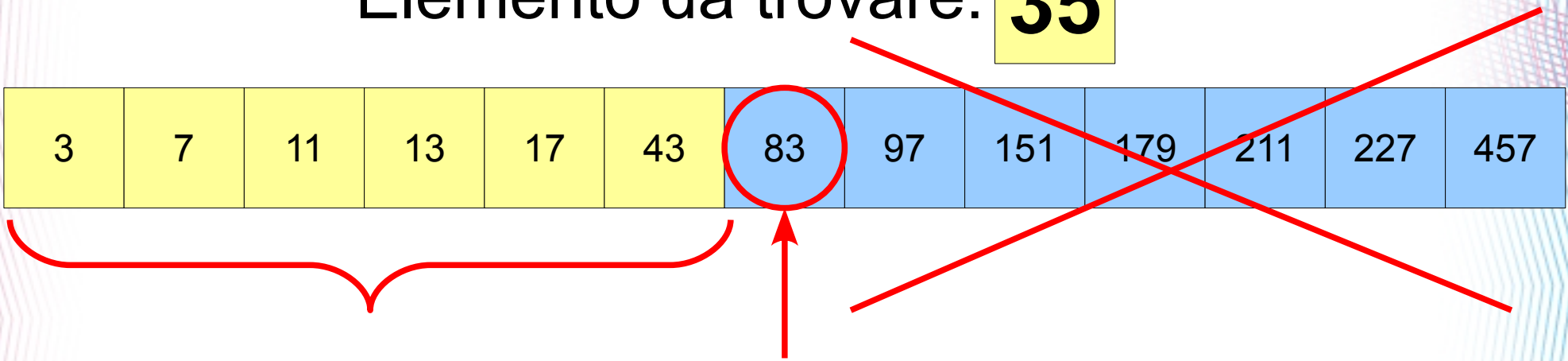
Parto da metà dell'array

83 è l'elemento che sto cercando?

NO! Sto cercando 35.

La ricerca binaria

Elemento da trovare: **35**

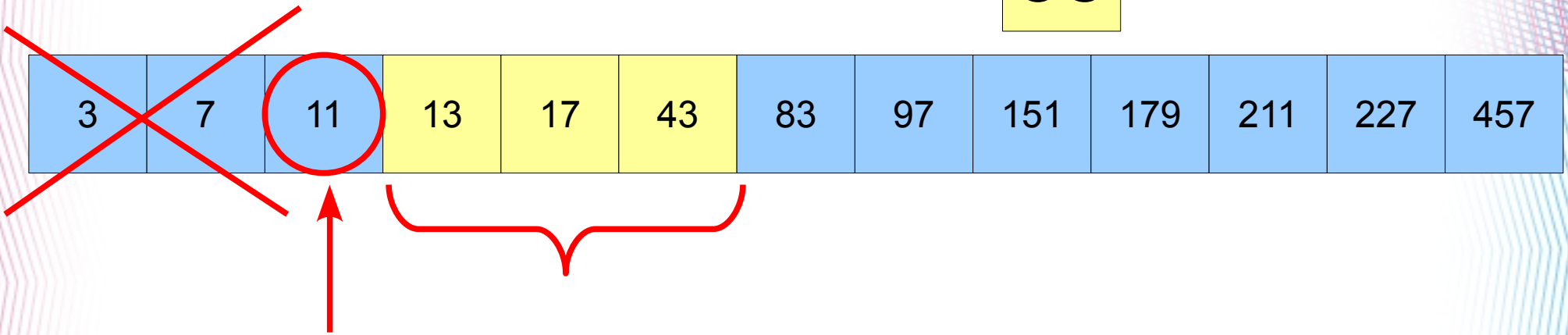


Elemento da cercare è < di 83

Non prendo più in considerazione la seconda metà dell'array ma ricerco solo nella prima metà.

La ricerca binaria

Elemento da trovare: **35**



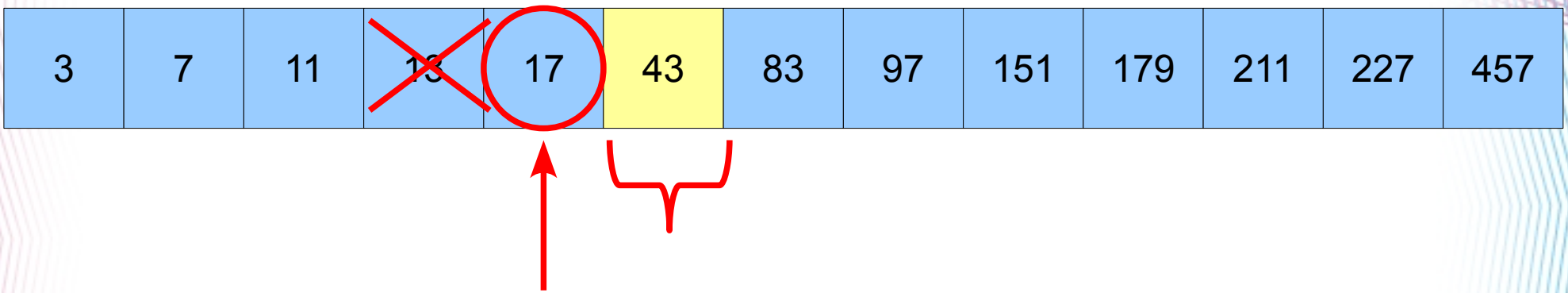
Parto da metà del vettore

35 > 11

L'elemento da cercare è maggiore di quello nella metà del vettore, ricerco nella seconda metà.

La ricerca binaria

Elemento da trovare: **35**



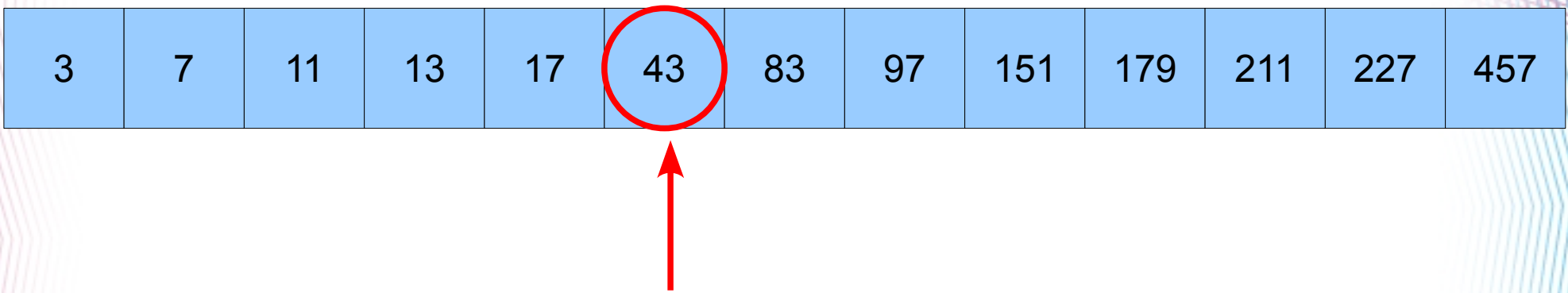
Parto da metà del vettore

35 > 17

Se l'elemento da cercare è maggiore di quello nella metà del vettore, ricerco nella seconda metà.

La ricerca binaria

Elemento da trovare: **35**



Parto da metà del vettore
L'elemento non è 43

Ricerca finita
Elemento non trovato

Implementazione

Richiamare la funzione di ricerca binaria:

```
int risultato = ricercaBinariaRicorsiva(A, 0, A.length,x);

if(risultato==-1)
    System.out.println("Elemento non trovato!"  );
else
    System.out.println("Elemento trovato in posizione: " + risultato);
```

Implementazione

```
public static int ricercaBinariaRicorsiva (int A[], int first, int last, int key) {
    if (first > last) return -1;
    else {
        int mid = (first + last) / 2;
        if (key == A[mid]) return mid;
        else
            if (key > A[mid])
                return ricercaBinariaRicorsiva (A, mid+1, last, key);
            else
                return ricercaBinariaRicorsiva (A, first, mid-1, key);
    }
}
```

Ordinamento

Esistono molti algoritmi di ordinamento. Tutti ricevono in input una sequenza non ordinata di elementi e restituiscono la sequenza ordinata.

MergeSort

Il merge sort è un algoritmo di ordinamento che utilizza un processo di risoluzione ricorsivo.

L'idea alla base del merge sort è il procedimento Divide et Impera.

Il merge sort opera dividendo l'insieme da ordinare in due metà e procede all'ordinamento delle medesime ricorsivamente.

Quando si sono divise tutte le metà si procede alla loro fusione (merge) costruendo un insieme ordinato.

MergeSort

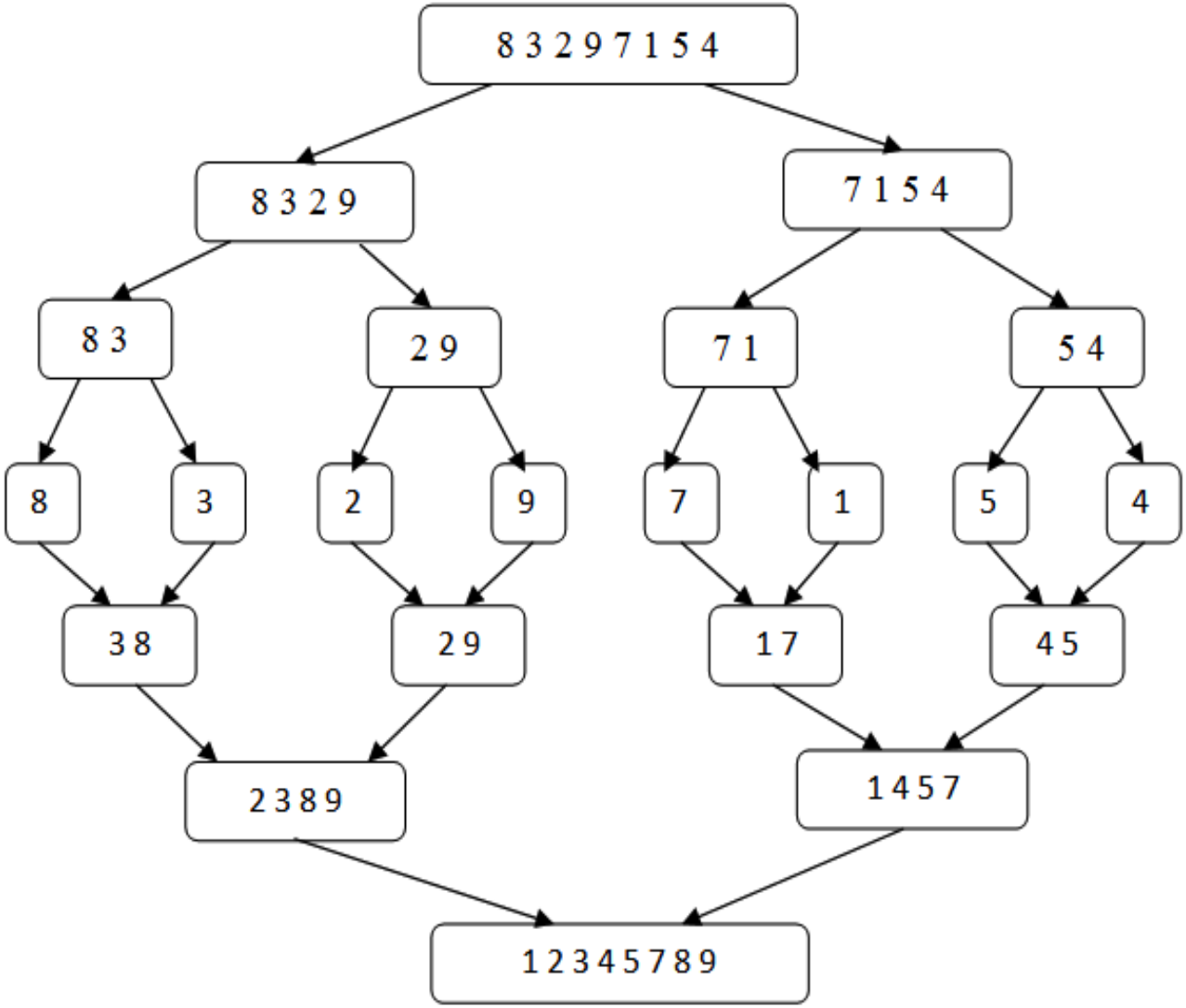
Prima fase: divide

L'insieme di elementi viene diviso in 2 metà ricorsivamente.

Seconda fase: impera

Supponendo di avere due sequenze già ordinate, per unirle, l'algoritmo mergesort estrae ripetutamente il minimo delle due sequenze in ingresso e lo pone in una sequenza in uscita.

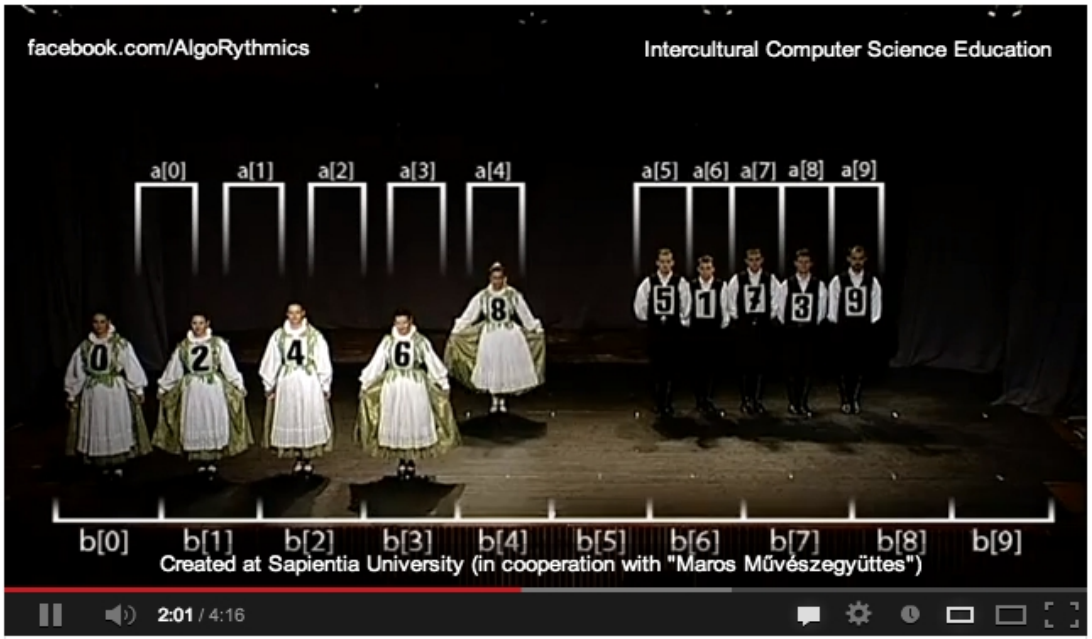
MergeSort



Le danze ungheresi

Video per capire il funzionamento del Merge Sort:

Video Danze Ungheresi



Implementazione

```
static void mergesort(int A[],int inizio, int fine) {

    if(inizio<fine) {
        int medio = inizio + (fine-inizio)/2;
        mergesort(A,inizio,medio);
        mergesort(A,medio+1,fine);
        merge( A, inizio, fine, medio);
    }

}
```

Implementazione

```
private static void merge(int[] A, int inizio, int fine, int medio) {
    int[] temp = new int[fine-inizio+1];
    int j = inizio;
    int k = medio+1;
    int cont = 0;
    while ((j <= medio) && (k <= fine)) {
        //caso in cui entrambi i vettori contengono elementi
        if (A[j] <= A[k]) {
            temp[cont] = A[j];
            j++;
        }
        else {
            temp[cont] = A[k];
            k++;
        }
        cont++;
    }

    if (k > fine) //il secondo pezzo del vettore è finito prima
        for (int i = j; i <= medio; i++) {
            temp[cont] = A[i];
            cont++;
        }
    else //il primo pezzo del vettore è finito prima
        for (int i = k; i <= fine; i++) {
            temp[cont] = A[i];
            cont++;
        }

    //si copia il vettore ausiliario nel vettore originario
    for(int i = 0; i < temp.length; i++)
        A[inizio+i] = temp[i];
}
```