Algoritmi di ordinamento

Violetta Lonati

Università degli studi di Milano
Dipartimento di Informatica

Laboratorio di algoritmi e strutture dati
Corso di laurea in Informatica

10 novembre 2016

Argomenti

Alcuni algoritmi di ordinamento ricorsivi

- Selectionsort
- Mergesort

Algoritmi di ordinamento

- Abbiamo già implementato l'ordinamento per inserimento, utile per riempire un array mantenendolo ordinato ad ogni passo.
- Ora vediamo alcuni algoritmi di ordinamento classici, usati per ordinare un array già riempito ma in disordine.
  - selection sort (in italiano ordinamento per selezione);
  - mergesort (in italiano ordinamento per immersione);
- Tutti e due gli algoritmi sono basati sui confronti tra elementi e sono ricorsivi: la base della ricorsione è data dagli array di lunghezza 0 o 1, che sono sempre ordinati.
- Per semplicità faremo riferimento soltanto ad array di interi.

selectionsort (int a[ ], int n)

Ordina i primi n elementi del vettore a, procedendo come segue:

1. cerca in a l'elemento massimo e lo scambia con l'elemento nell'ultima posizione dell'array;
2. richiama ricorsivamente se stessa per ordinare i primi $n - 1$ elementi dell'array.

selectionsort (int a[ ], int n)

Ordina i primi n elementi del vettore a, procedendo come segue:

1. cerca in a l'elemento massimo e lo scambia con l'elemento nell'ultima posizione dell'array;
2. richiama ricorsivamente se stessa per ordinare i primi $n - 1$ elementi dell'array.

Esempio su input a = 15, 29, 11, 7

selectionsort (int a[ ], int n)

Ordina i primi n elementi del vettore a, procedendo come segue:

1. cerca in a l'elemento massimo e lo scambia con l'elemento nell'ultima posizione dell'array;
2. richiama ricorsivamente se stessa per ordinare i primi $n - 1$ elementi dell'array.

Esempio su input a = 15, 29, 11, 7

```
selectionsort( a, 4 )
```

15 29 11 7

selectionsort (int a[ ], int n)

Ordina i primi n elementi del vettore a, procedendo come segue:

1. cerca in a l'elemento massimo e lo scambia con l'elemento nell'ultima posizione dell'array;
2. richiama ricorsivamente se stessa per ordinare i primi $n-1$ elementi dell'array.

Esempio su input a = 15, 29, 11, 7

```
selectionsort( a, 4 )
```

15	29	11	7
15	7	11	29

selectionsort (int a[ ], int n)

Ordina i primi n elementi del vettore a, procedendo come segue:

1. cerca in a l'elemento massimo e lo scambia con l'elemento nell'ultima posizione dell'array;
2. richiama ricorsivamente se stessa per ordinare i primi $n-1$ elementi dell'array.

Esempio su input a = 15, 29, 11, 7

```
selectionsort( a, 4 )
  selectionsort( a, 3 )
```

15	29	11	7
15	7	11	29
15	7	11	29

selectionsort (int a[ ], int n)

Ordina i primi n elementi del vettore a, procedendo come segue:

1. cerca in a l'elemento massimo e lo scambia con l'elemento nell'ultima posizione dell'array;
2. richiama ricorsivamente se stessa per ordinare i primi $n-1$ elementi dell'array.

Esempio su input a = 15, 29, 11, 7

```
selectionsort( a, 4 )
  selectionsort( a, 3 )
```

15	29	11	7
15	7	11	29
15	7	11	29
11	7	15	29

selectionsort (int a[ ], int n)

Ordina i primi n elementi del vettore a, procedendo come segue:

1. cerca in a l'elemento massimo e lo scambia con l'elemento nell'ultima posizione dell'array;
2. richiama ricorsivamente se stessa per ordinare i primi $n-1$ elementi dell'array.

Esempio su input a = 15, 29, 11, 7

```
selectionsort( a, 4 )
  selectionsort( a, 3 )
    selectionsort( a, 2 )
```

15	29	11	7
15	7	11	29
15	7	11	29
11	7	15	29
11	7	15	29

selectionsort (int a[ ], int n)

Ordina i primi n elementi del vettore a, procedendo come segue:

1. cerca in a l'elemento massimo e lo scambia con l'elemento nell'ultima posizione dell'array;
2. richiama ricorsivamente se stessa per ordinare i primi $n-1$ elementi dell'array.

Esempio su input a = 15, 29, 11, 7

```
selectionsort( a, 4 )
  selectionsort( a, 3 )
    selectionsort( a, 2 )
```

15	29	11	7
15	7	11	29
15	7	11	29
11	7	15	29
11	7	15	29
7	11	15	29

selectionsort (int a[ ], int n)

Ordina i primi n elementi del vettore a, procedendo come segue:

1. cerca in a l'elemento massimo e lo scambia con l'elemento nell'ultima posizione dell'array;
2. richiama ricorsivamente se stessa per ordinare i primi $n-1$ elementi dell'array.

Esempio su input a = 15, 29, 11, 7

```
selectionsort( a, 4 )
  selectionsort( a, 3 )
    selectionsort( a, 2 )
      selectionsort( a, 1 )
```

15	29	11	7
15	7	11	29
15	7	11	29
11	7	15	29
11	7	15	29
7	11	15	29
7	11	15	29

selectionsort (int a[ ], int n)

Ordina i primi n elementi del vettore a, procedendo come segue:

1. cerca in a l'elemento massimo e lo scambia con l'elemento nell'ultima posizione dell'array;
2. richiama ricorsivamente se stessa per ordinare i primi $n-1$ elementi dell'array.

Esempio su input a = 15, 29, 11, 7

```
selectionsort( a, 4 )
  selectionsort( a, 3 )
    selectionsort( a, 2 )
      selectionsort( a, 1 )
```

15	29	11	7
15	7	11	29
15	7	11	29
11	7	15	29
11	7	15	29
7	11	15	29
7	11	15	29
7	11	15	29

mergesort (int a[ ], int sx, int dx)

Ordina la parte dell'array *a* compresa tra gli indici *sx* e *dx*, come segue:

1. divide l'array in due sotto-array di dimensione circa uguale;
2. ordina il sotto-array di sinistra richiamando se stessa;
3. ordina il sotto-array di destra richiamando se stessa;
4. integra (merge) i due sotto-array in un unico array ordinato.

La base della ricorsione è data dagli array di lunghezza 0 o 1, che sono sempre ordinati.

merging

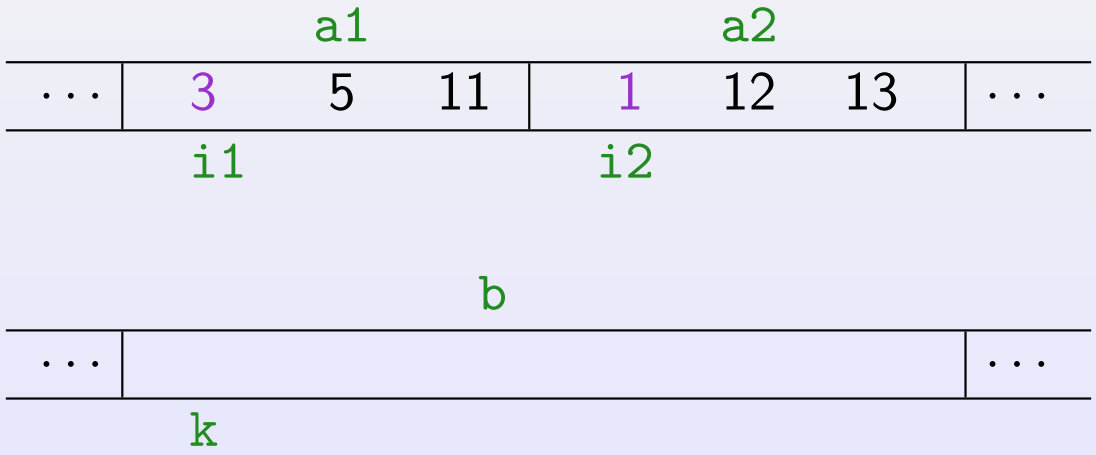
La parte di integrazione (merge) di due array ordinati `a1` e `a2` funziona con un vettore di supporto `b`:

- si scorrono entrambi gli array da sinistra a destra usando due indicatori `i1` e `i2` rispettivamente;
- ad ogni passo si confronta `a1[i1]` con `a2[i2]` e si sceglie l'elemento più piccolo, lo si copia nell'array di supporto `b` (nella prima posizione libera) e si incrementa l'indicatore relativo ad esso;
- quando `i1` esce da `a1` oppure `i2` esce da `a2`, la parte rimanente dell'altro array viene copiata in `b`;
- alla fine si copia il contenuto dell'array `b` nell'array originale.

merging

La parte di integrazione (merge) di due array ordinati `a1` e `a2` funziona con un vettore di supporto `b`:

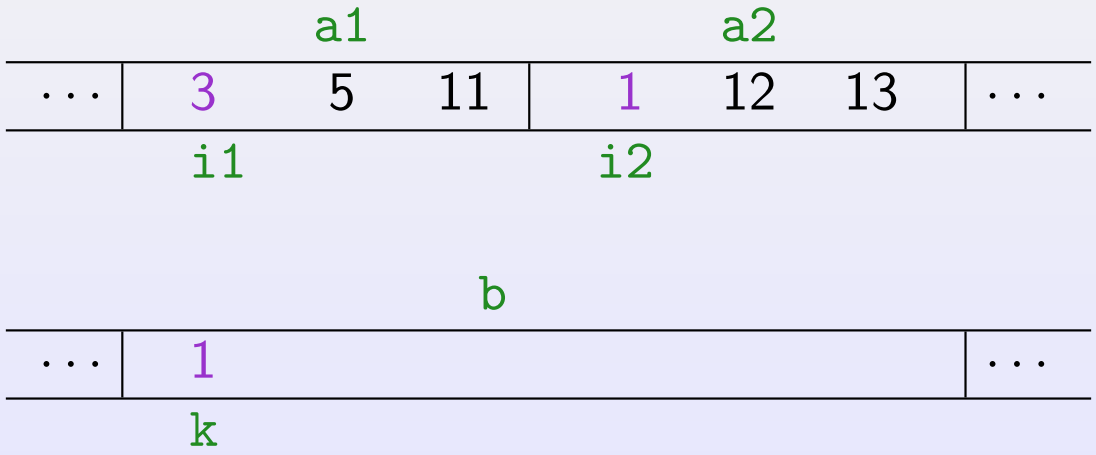
- si scorrono entrambi gli array da sinistra a destra usando due indicatori `i1` e `i2` rispettivamente;
- ad ogni passo si confronta `a1[i1]` con `a2[i2]` e si sceglie l'elemento più piccolo, lo si copia nell'array di supporto `b` (nella prima posizione libera) e si incrementa l'indicatore relativo ad esso;
- quando `i1` esce da `a1` oppure `i2` esce da `a2`, la parte rimanente dell'altro array viene copiata in `b`;
- alla fine si copia il contenuto dell'array `b` nell'array originale.



merging

La parte di integrazione (merge) di due array ordinati `a1` e `a2` funziona con un vettore di supporto `b`:

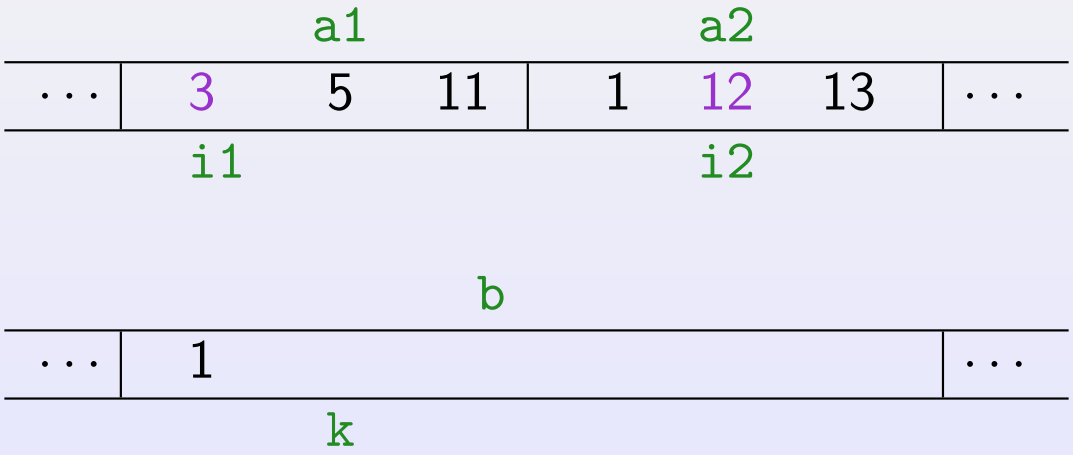
- si scorrono entrambi gli array da sinistra a destra usando due indicatori `i1` e `i2` rispettivamente;
- ad ogni passo si confronta `a1[i1]` con `a2[i2]` e si sceglie l'elemento più piccolo, lo si copia nell'array di supporto `b` (nella prima posizione libera) e si incrementa l'indicatore relativo ad esso;
- quando `i1` esce da `a1` oppure `i2` esce da `a2`, la parte rimanente dell'altro array viene copiata in `b`;
- alla fine si copia il contenuto dell'array `b` nell'array originale.



merging

La parte di integrazione (merge) di due array ordinati `a1` e `a2` funziona con un vettore di supporto `b`:

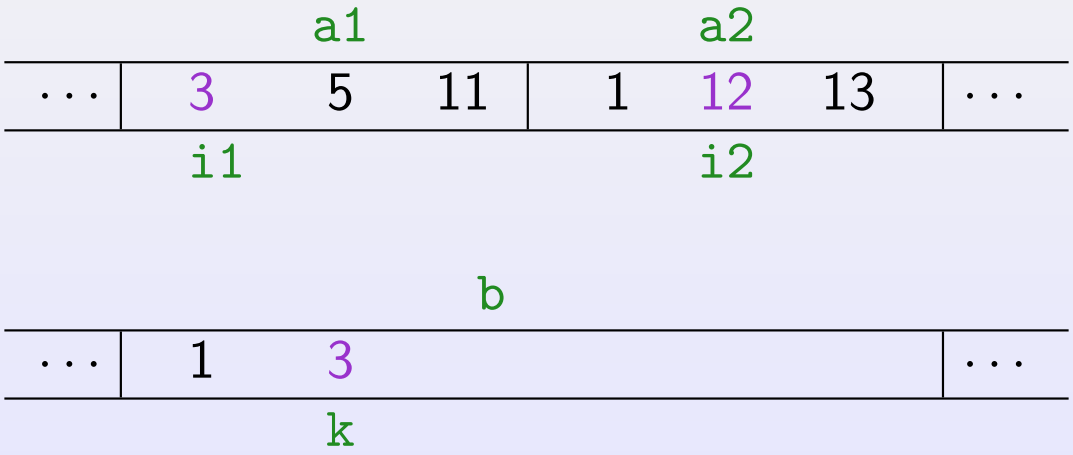
- si scorrono entrambi gli array da sinistra a destra usando due indicatori `i1` e `i2` rispettivamente;
- ad ogni passo si confronta `a1[i1]` con `a2[i2]` e si sceglie l'elemento più piccolo, lo si copia nell'array di supporto `b` (nella prima posizione libera) e si incrementa l'indicatore relativo ad esso;
- quando `i1` esce da `a1` oppure `i2` esce da `a2`, la parte rimanente dell'altro array viene copiata in `b`;
- alla fine si copia il contenuto dell'array `b` nell'array originale.



merging

La parte di integrazione (merge) di due array ordinati `a1` e `a2` funziona con un vettore di supporto `b`:

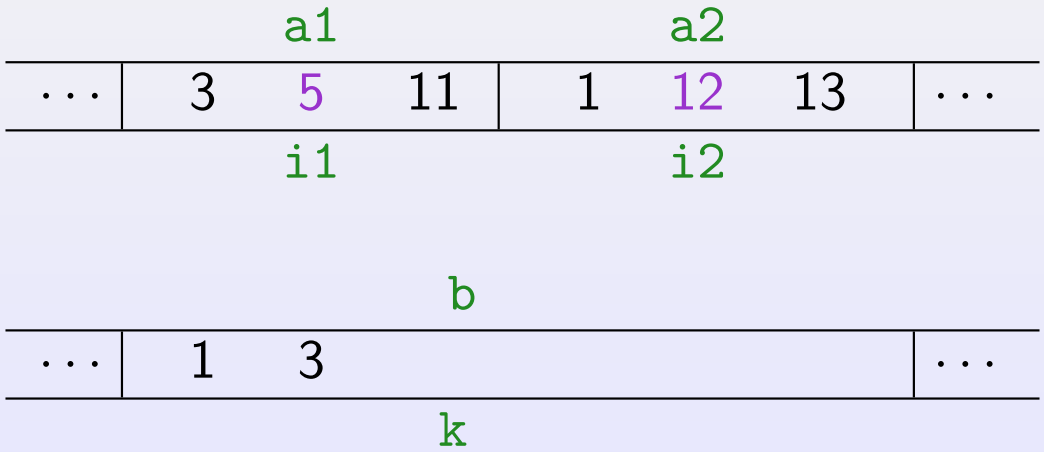
- si scorrono entrambi gli array da sinistra a destra usando due indicatori `i1` e `i2` rispettivamente;
- ad ogni passo si confronta `a1[i1]` con `a2[i2]` e si sceglie l'elemento più piccolo, lo si copia nell'array di supporto `b` (nella prima posizione libera) e si incrementa l'indicatore relativo ad esso;
- quando `i1` esce da `a1` oppure `i2` esce da `a2`, la parte rimanente dell'altro array viene copiata in `b`;
- alla fine si copia il contenuto dell'array `b` nell'array originale.



merging

La parte di integrazione (merge) di due array ordinati `a1` e `a2` funziona con un vettore di supporto `b`:

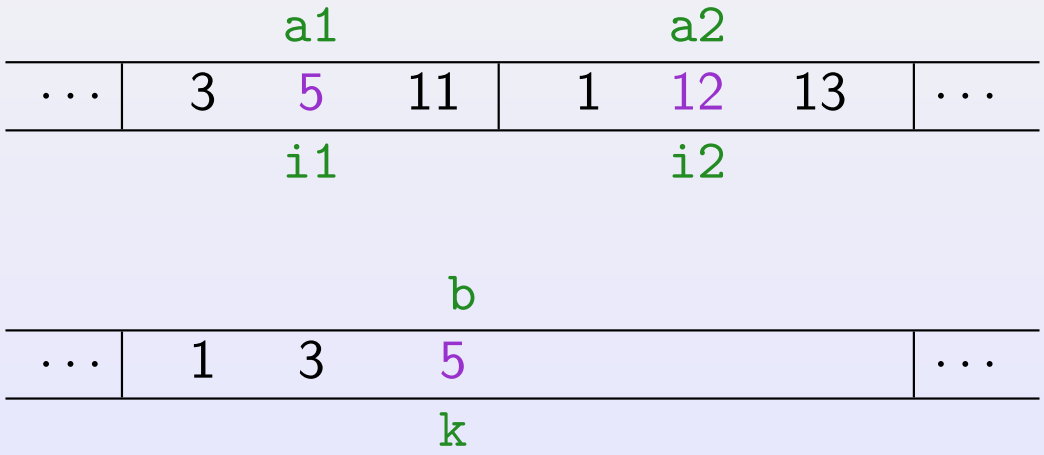
- si scorrono entrambi gli array da sinistra a destra usando due indicatori `i1` e `i2` rispettivamente;
- ad ogni passo si confronta `a1[i1]` con `a2[i2]` e si sceglie l'elemento più piccolo, lo si copia nell'array di supporto `b` (nella prima posizione libera) e si incrementa l'indicatore relativo ad esso;
- quando `i1` esce da `a1` oppure `i2` esce da `a2`, la parte rimanente dell'altro array viene copiata in `b`;
- alla fine si copia il contenuto dell'array `b` nell'array originale.



merging

La parte di integrazione (merge) di due array ordinati `a1` e `a2` funziona con un vettore di supporto `b`:

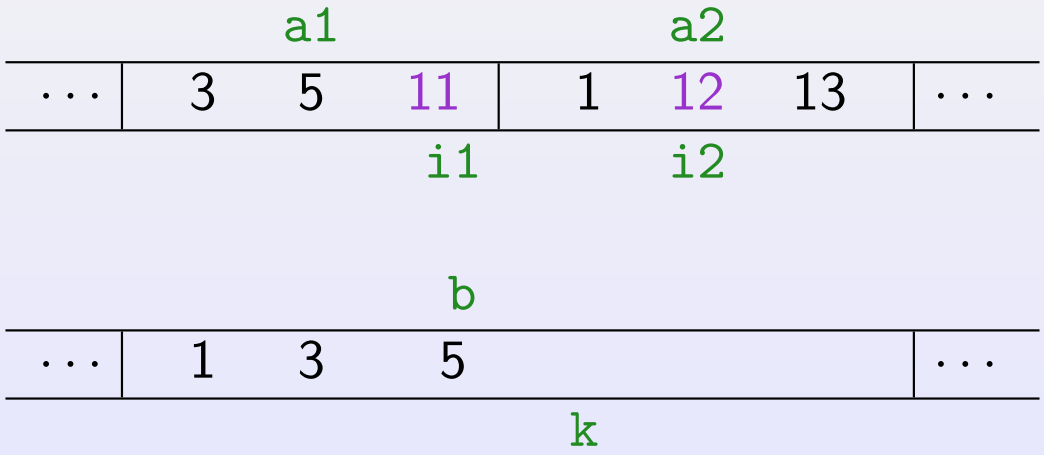
- si scorrono entrambi gli array da sinistra a destra usando due indicatori `i1` e `i2` rispettivamente;
- ad ogni passo si confronta `a1[i1]` con `a2[i2]` e si sceglie l'elemento più piccolo, lo si copia nell'array di supporto `b` (nella prima posizione libera) e si incrementa l'indicatore relativo ad esso;
- quando `i1` esce da `a1` oppure `i2` esce da `a2`, la parte rimanente dell'altro array viene copiata in `b`;
- alla fine si copia il contenuto dell'array `b` nell'array originale.



merging

La parte di integrazione (merge) di due array ordinati `a1` e `a2` funziona con un vettore di supporto `b`:

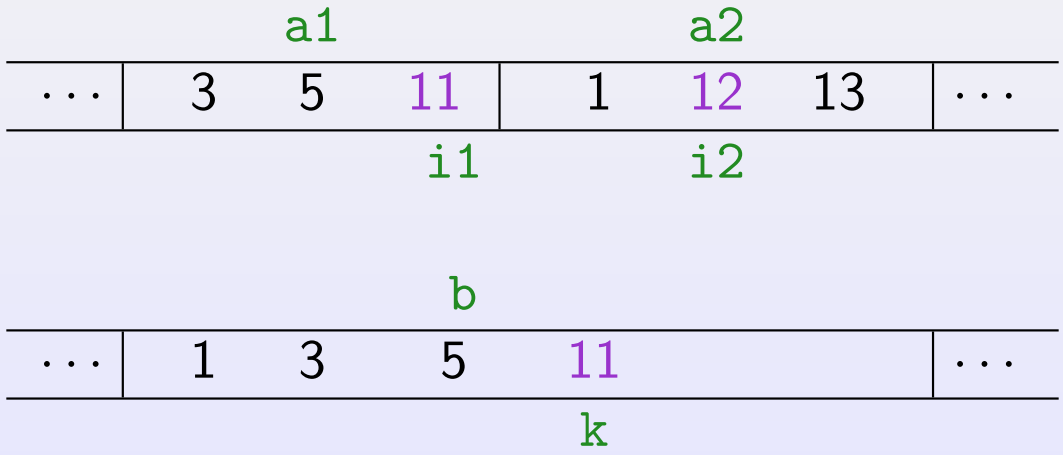
- si scorrono entrambi gli array da sinistra a destra usando due indicatori `i1` e `i2` rispettivamente;
- ad ogni passo si confronta `a1[i1]` con `a2[i2]` e si sceglie l'elemento più piccolo, lo si copia nell'array di supporto `b` (nella prima posizione libera) e si incrementa l'indicatore relativo ad esso;
- quando `i1` esce da `a1` oppure `i2` esce da `a2`, la parte rimanente dell'altro array viene copiata in `b`;
- alla fine si copia il contenuto dell'array `b` nell'array originale.



merging

La parte di integrazione (merge) di due array ordinati `a1` e `a2` funziona con un vettore di supporto `b`:

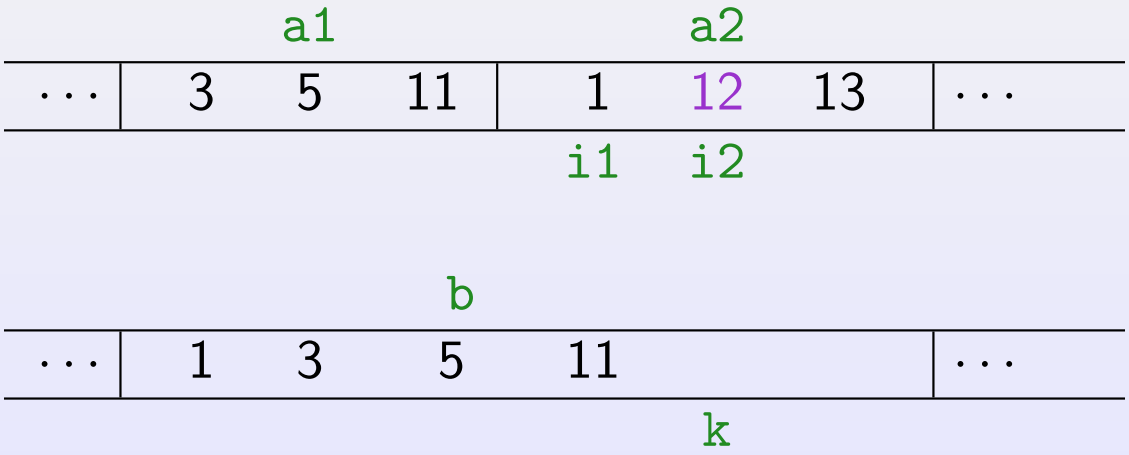
- si scorrono entrambi gli array da sinistra a destra usando due indicatori `i1` e `i2` rispettivamente;
- ad ogni passo si confronta `a1[i1]` con `a2[i2]` e si sceglie l'elemento più piccolo, lo si copia nell'array di supporto `b` (nella prima posizione libera) e si incrementa l'indicatore relativo ad esso;
- quando `i1` esce da `a1` oppure `i2` esce da `a2`, la parte rimanente dell'altro array viene copiata in `b`;
- alla fine si copia il contenuto dell'array `b` nell'array originale.



merging

La parte di integrazione (merge) di due array ordinati `a1` e `a2` funziona con un vettore di supporto `b`:

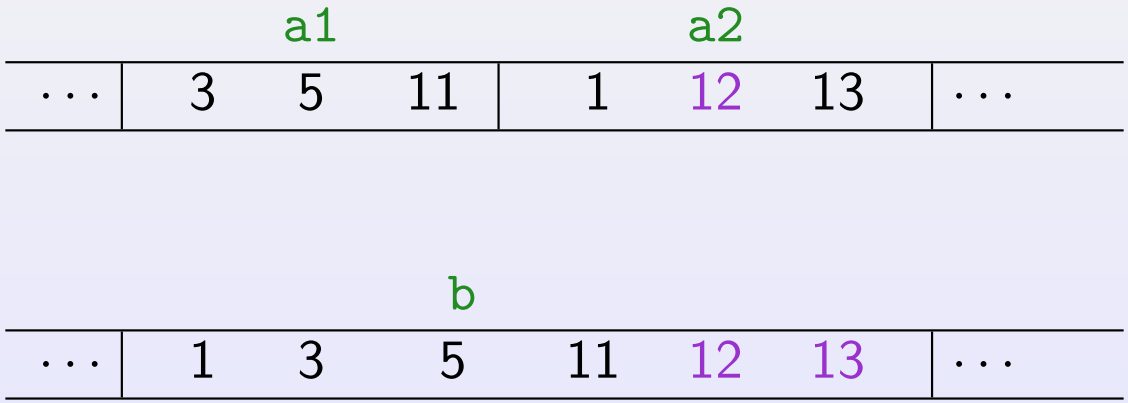
- si scorrono entrambi gli array da sinistra a destra usando due indicatori `i1` e `i2` rispettivamente;
- ad ogni passo si confronta `a1[i1]` con `a2[i2]` e si sceglie l'elemento più piccolo, lo si copia nell'array di supporto `b` (nella prima posizione libera) e si incrementa l'indicatore relativo ad esso;
- quando `i1` esce da `a1` oppure `i2` esce da `a2`, la parte rimanente dell'altro array viene copiata in `b`;
- alla fine si copia il contenuto dell'array `b` nell'array originale.



merging

La parte di integrazione (merge) di due array ordinati `a1` e `a2` funziona con un vettore di supporto `b`:

- si scorrono entrambi gli array da sinistra a destra usando due indicatori `i1` e `i2` rispettivamente;
- ad ogni passo si confronta `a1[i1]` con `a2[i2]` e si sceglie l'elemento più piccolo, lo si copia nell'array di supporto `b` (nella prima posizione libera) e si incrementa l'indicatore relativo ad esso;
- quando `i1` esce da `a1` oppure `i2` esce da `a2`, la parte rimanente dell'altro array viene copiata in `b`;
- alla fine si copia il contenuto dell'array `b` nell'array originale.



mergesort (int a[ ], int sx, int dx)

Ordina la parte dell'array *a* compresa tra gli indici *sx* e *dx*, come segue:

1. divide l'array in due sotto-array di dimensione circa uguale;
2. ordina il sotto-array di sinistra richiamando se stessa;
3. ordina il sotto-array di destra richiamando se stessa;
4. integra (merge) i due sotto-array in un unico array ordinato.

La base della ricorsione è data dagli array di lunghezza 0 o 1, che sono sempre ordinati.

mergesort - esempio

`mergesort( a, 0, 4 )` 29 15 11 7 13

mergesort - esempio

```
mergesort( a, 0, 4 )
  mergesort( a, 0, 2 )
```

29 15 11 7 13

29 15 11 7 13

mergesort - esempio

```
mergesort( a, 0, 4 )
  mergesort( a, 0, 2 )
    mergesort( a, 0, 1 )
```

29	15	11	7	13
29	15	11	7	13
29	15	11	7	13

mergesort - esempio

```
mergesort( a, 0, 4 )
  mergesort( a, 0, 2 )
    mergesort( a, 0, 1 )
      mergesort( a, 0, 0 )
```

29	15	11	7	13
29	15	11	7	13
29	15	11	7	13
29	15	11	7	13

mergesort - esempio

```
mergesort( a, 0, 4 )
  mergesort( a, 0, 2 )
    mergesort( a, 0, 1 )
      mergesort( a, 0, 0 )
      mergesort( a, 1, 1 )
```

29	15	11	7	13
29	15	11	7	13
29	15	11	7	13
29	15	11	7	13
29	15	11	7	13

mergesort - esempio

`mergesort( a, 0, 4 )`	29	15	11	7	13
`  mergesort( a, 0, 2 )`	29	15	11	7	13
`    mergesort( a, 0, 1 )`	29	15	11	7	13
`      mergesort( a, 0, 0 )`	29	15	11	7	13
`      mergesort( a, 1, 1 )`	29	15	11	7	13
`      merge( a, 0, 1, 0 )`	15	29	11	7	13

mergesort - esempio

`mergesort( a, 0, 4 )`	29	15	11	7	13
`  mergesort( a, 0, 2 )`	29	15	11	7	13
`    mergesort( a, 0, 1 )`	29	15	11	7	13
`      mergesort( a, 0, 0 )`	29	15	11	7	13
`      mergesort( a, 1, 1 )`	29	15	11	7	13
`      merge( a, 0, 1, 0 )`	15	29	11	7	13
`    mergesort( a, 2, 2 )`	15	29	11	7	13

mergesort - esempio

`mergesort( a, 0, 4 )`	29	15	11	7	13
`  mergesort( a, 0, 2 )`	29	15	11	7	13
`    mergesort( a, 0, 1 )`	29	15	11	7	13
`      mergesort( a, 0, 0 )`	29	15	11	7	13
`      mergesort( a, 1, 1 )`	29	15	11	7	13
`      merge( a, 0, 1, 0 )`	15	29	11	7	13
`    mergesort( a, 2, 2 )`	15	29	11	7	13
`    merge( a, 0, 2, 1 )`	11	15	29	7	13

mergesort - esempio

`mergesort( a, 0, 4 )`	29	15	11	7	13
`  mergesort( a, 0, 2 )`	29	15	11	7	13
`    mergesort( a, 0, 1 )`	29	15	11	7	13
`      mergesort( a, 0, 0 )`	29	15	11	7	13
`      mergesort( a, 1, 1 )`	29	15	11	7	13
`      merge( a, 0, 1, 0 )`	15	29	11	7	13
`    mergesort( a, 2, 2 )`	15	29	11	7	13
`    merge( a, 0, 2, 1 )`	11	15	29	7	13
`  mergesort( a, 3, 4 )`	11	15	29	7	13

mergesort - esempio

`mergesort( a, 0, 4 )`	29	15	11	7	13
`  mergesort( a, 0, 2 )`	29	15	11	7	13
`    mergesort( a, 0, 1 )`	29	15	11	7	13
`      mergesort( a, 0, 0 )`	29	15	11	7	13
`      mergesort( a, 1, 1 )`	29	15	11	7	13
`      merge( a, 0, 1, 0 )`	15	29	11	7	13
`    mergesort( a, 2, 2 )`	15	29	11	7	13
`    merge( a, 0, 2, 1 )`	11	15	29	7	13
`  mergesort( a, 3, 4 )`	11	15	29	7	13
`    mergesort( a, 3, 3 )`	11	15	29	7	13

mergesort - esempio

`mergesort( a, 0, 4 )`	29	15	11	7	13
`  mergesort( a, 0, 2 )`	29	15	11	7	13
`    mergesort( a, 0, 1 )`	29	15	11	7	13
`      mergesort( a, 0, 0 )`	29	15	11	7	13
`      mergesort( a, 1, 1 )`	29	15	11	7	13
`      merge( a, 0, 1, 0 )`	15	29	11	7	13
`    mergesort( a, 2, 2 )`	15	29	11	7	13
`    merge( a, 0, 2, 1 )`	11	15	29	7	13
`  mergesort( a, 3, 4 )`	11	15	29	7	13
`    mergesort( a, 3, 3 )`	11	15	29	7	13
`    mergesort( a, 4, 4 )`	11	15	29	7	13

mergesort - esempio

`mergesort( a, 0, 4 )`	29	15	11	7	13
`  mergesort( a, 0, 2 )`	29	15	11	7	13
`    mergesort( a, 0, 1 )`	29	15	11	7	13
`      mergesort( a, 0, 0 )`	29	15	11	7	13
`      mergesort( a, 1, 1 )`	29	15	11	7	13
`      merge( a, 0, 1, 0 )`	15	29	11	7	13
`    mergesort( a, 2, 2 )`	15	29	11	7	13
`    merge( a, 0, 2, 1 )`	11	15	29	7	13
`  mergesort( a, 3, 4 )`	11	15	29	7	13
`    mergesort( a, 3, 3 )`	11	15	29	7	13
`    mergesort( a, 4, 4 )`	11	15	29	7	13
`    merge( a, 3, 4, 3 )`	11	15	29	7	13

mergesort - esempio

`mergesort( a, 0, 4 )`	29	15	11	7	13
`  mergesort( a, 0, 2 )`	29	15	11	7	13
`    mergesort( a, 0, 1 )`	29	15	11	7	13
`      mergesort( a, 0, 0 )`	29	15	11	7	13
`      mergesort( a, 1, 1 )`	29	15	11	7	13
`      merge( a, 0, 1, 0 )`	15	29	11	7	13
`    mergesort( a, 2, 2 )`	15	29	11	7	13
`    merge( a, 0, 2, 1 )`	11	15	29	7	13
`  mergesort( a, 3, 4 )`	11	15	29	7	13
`    mergesort( a, 3, 3 )`	11	15	29	7	13
`    mergesort( a, 4, 4 )`	11	15	29	7	13
`    merge( a, 3, 4, 3 )`	11	15	29	7	13
`  merge( a, 0, 4, 2 )`	7	11	13	15	29