Merge Sort di un array

Angelo Gargantini

4 giugno 2015

Idea di base

- Fino ad ora quando ordinavamo un array con il bubble sort, consideravamo sempre tutto l'array.
- Quante volte devo scandire l'array? Il numero di scambi nel caso peggiore è proporzionale a n^2
- E se se invece dividessimo il problema in sotto problemi?
- Cerchiamo di applicare le idee della ricerca dicotomica

Merge Sort

- Il meccanismo di ordinamento di questo algoritmo fa uso della tecnica **Divide et Impera**.
- Consiste nel suddividere *ricorsivamente* un problema complesso in due o più sotto-problemi più semplici e poi si ricombinano le soluzioni trovate per ricostruire la soluzione del problema complessivo.
- per ordinare un array
  - lo dividiamo in due parti
  - ordiniamo le due parti
  - fondiamo le due parti ordinate

metodi ricorsivi

- Diciamo che un metodo è **ricorsivo** quando nel suo corpo chiama se stesso
- Alcuni algoritmi si possono scrivere in modo compatto usando la ricorsione
- Esempio: il fattoriale

```
int fattoriale(int n){ return n * fattoriale(n-1)}
```

- Attenzione: posso avere facilmente computazioni infinite. Devo prevedere un caso di uscita (base della ricorsione).

```
int fattoriale(int n){
    if (n ==0) return 1;
    else return n * fattoriale(n-1)
}
```

metodi ricorsivi

- Diciamo che un metodo è **ricorsivo** quando nel suo corpo chiama se stesso
- Alcuni algoritmi si possono scrivere in modo compatto usando la ricorsione
- Esempio: il fattoriale

```
int fattoriale(int n){ return n * fattoriale(n-1)}
```

- Attenzione: posso avere facilmente computazioni infinite. Devo prevedere un caso di uscita (base della ricorsione).

```
int fattoriale(int n){
    if (n ==0) return 1;
    else return n * fattoriale(n-1)
}
```

Esercizio Fibonacci

Un altra funzione molto facile da implementare in modo ricorsivo è Fibonacci.

Fibonacci La successione di Fibonacci è una successione in sequenza di numeri interi naturali ciascun numero della quale è il risultato della somma dei due precedenti. La successione si definisce matematicamente assegnando i valori dei due primi termini, F(0):= 0 ed F(1):= 1, e chiedendo che per ogni successivo sia F(n) := F(n-1) + F(n-2) con n>1.

Attenzione: la versione semplice quante volte chiamerà F(0)

Ricerca ricorsiva

Anche metodi di ricerca si possono scrivere in modo ricorsivo.

La ricerca di un numero y in un array di interi A dalla posizione x si può fare facilmente in modo ricorsivo

```
//
// @return true  se y è in A a partire da pos
//
boolean cerca(int y, int[] A, int pos)
```

Come posso fare in modo ricorsivo? Il caso base (quando interrompo la ricerca?)

Esercizio

La conta di quanti elementi ci sono positivi in un array di interi
Alcune volte scrivo dei metodi helper.

```
int conta(int y, int[] A)
int conta(int y, int[] A, int pos)
```

Come posso fare in modo ricorsivo?

MergeSort

- Proviamo ad ordinare con il merge sort
- In pseudo codice:

```
merge_sort(array m)
    if length(m) ≤ 1 return m
    var integer middle = length(m) / 2
    merge_sort m fino a middle
    merge_sort m da middle+1 alla fine
    merge the two subarrays
```

- il metodo che mi serve prende l'array che deve ordinare e le posizioni

Ordinamento di porzione di array

Se vogliamo ordinare solo una porzione di un array, da min a max

- min: prima posizione che devo ordinare
- max: ultima posizione che devo ordinare

```
merge_sort(array m, int min, int max)
      if min == max return
      var integer middle = (max - min)  / 2
      merge_sort m da min fino a middle
      merge_sort m da middle+1 a max
      merge the two subarrays
```

Codice - metodi

Ecco i metodi che ci servono e che implementiamo (come metodi statici)

1. Per ordinare un intero array:
   `static void mergeSort(int[] array)`
   - mi basta fare mergesort(m,0,m.length-1);
2. Ordina l'array da min a max (compreso)
   `static void mergeSort(int[] array, int min, int max)`
3. Fa il merge di due porzioni dell'array da min a med e da med +1 a max
   `static void merge(int[] array, int min, int med, int max)`

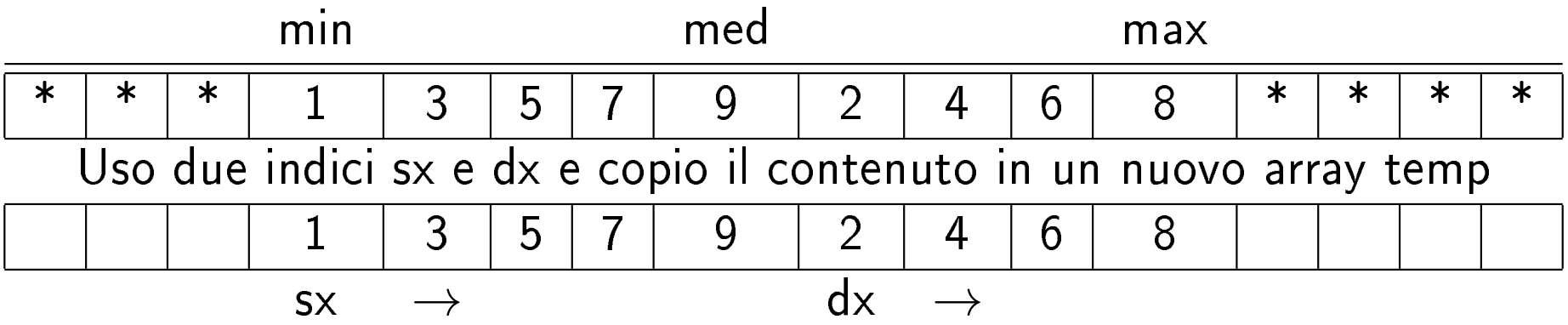
codice di merge_sort

```
// sort di m dalla posizione min a max (compresi)
static void mergesort(int[] m, int min, int max){
  // definizione ricorsiva
  // base della ricorsione (quando finisco?)
  // la parte da ordinare ha un solo elemento
  if (max==min) return;
  // altrimenti max > min
  // 1. calcolo la metà
  int med = (min + max) /2;
  // 2. ordino da min a med
  mergesort(m,min,med);
  // 3. ordino da med +1 a max
  mergesort(m,med+1,max);
  // merge delle due parti
  merge(m,min,med,max);
}
```

Fare il merge tra due porzioni di array

`merge(int[] array, int min, int med, int max)`

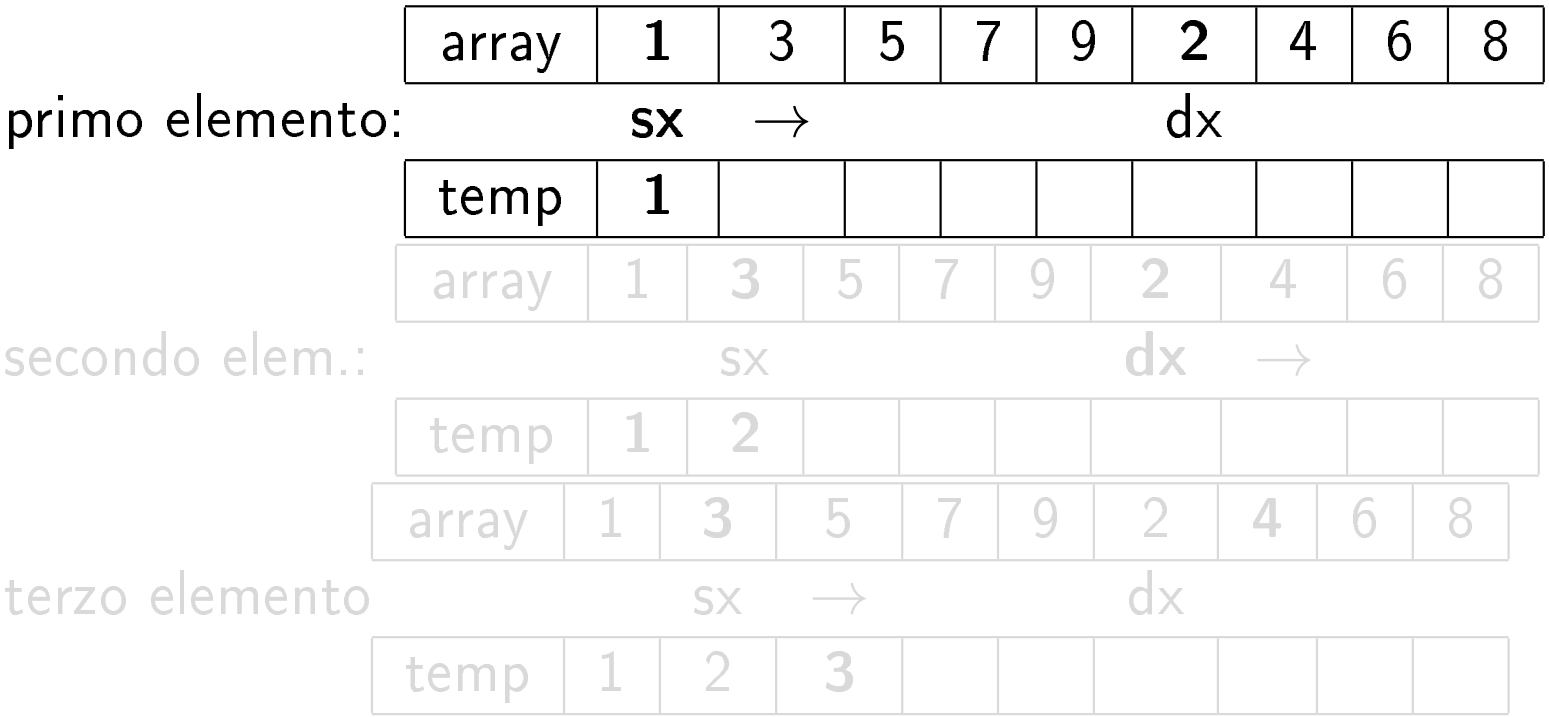
- Assumo che l'array da min a med sia ordinato e così anche da med +1 a max



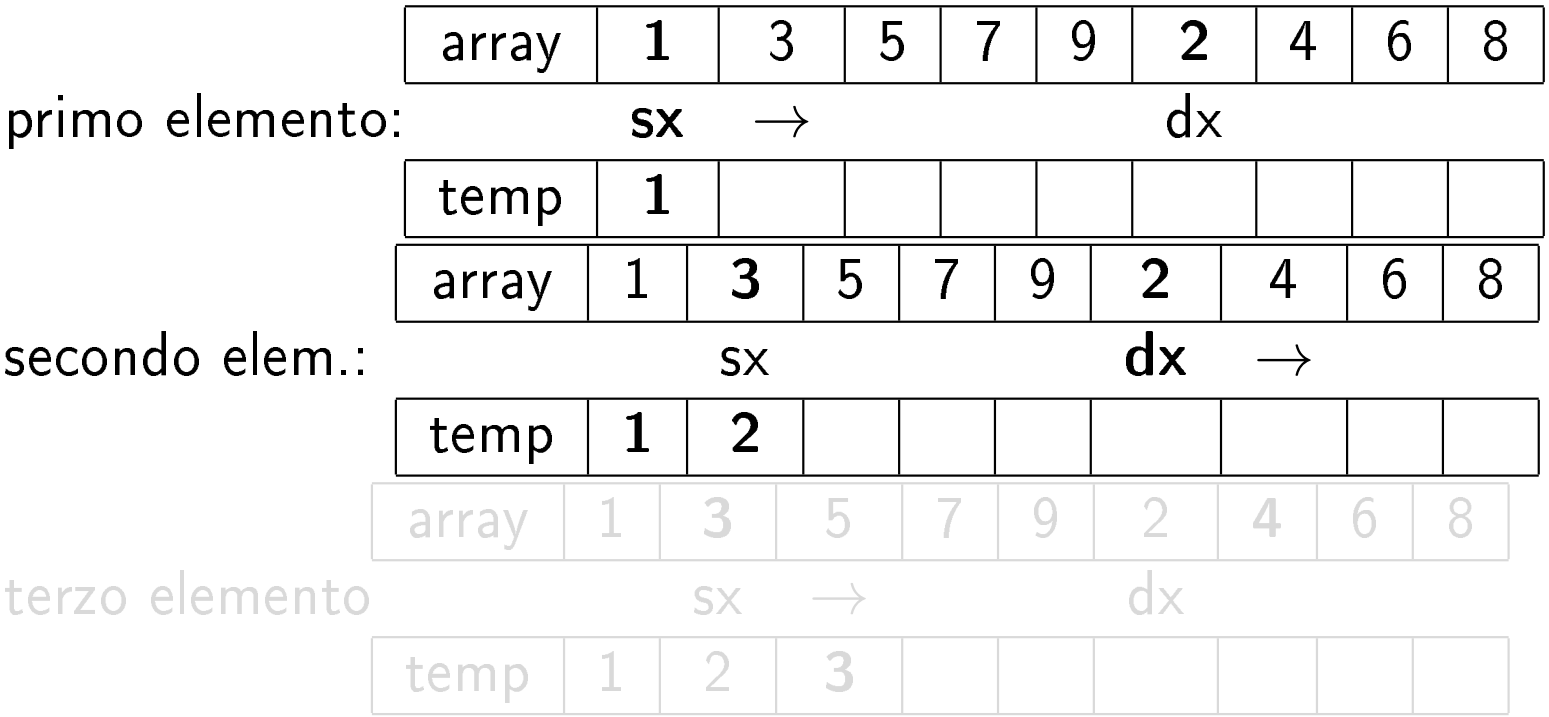
Nuovo array temp con indice i

Copio da array l'elemento minore tra quelli di sx e dx

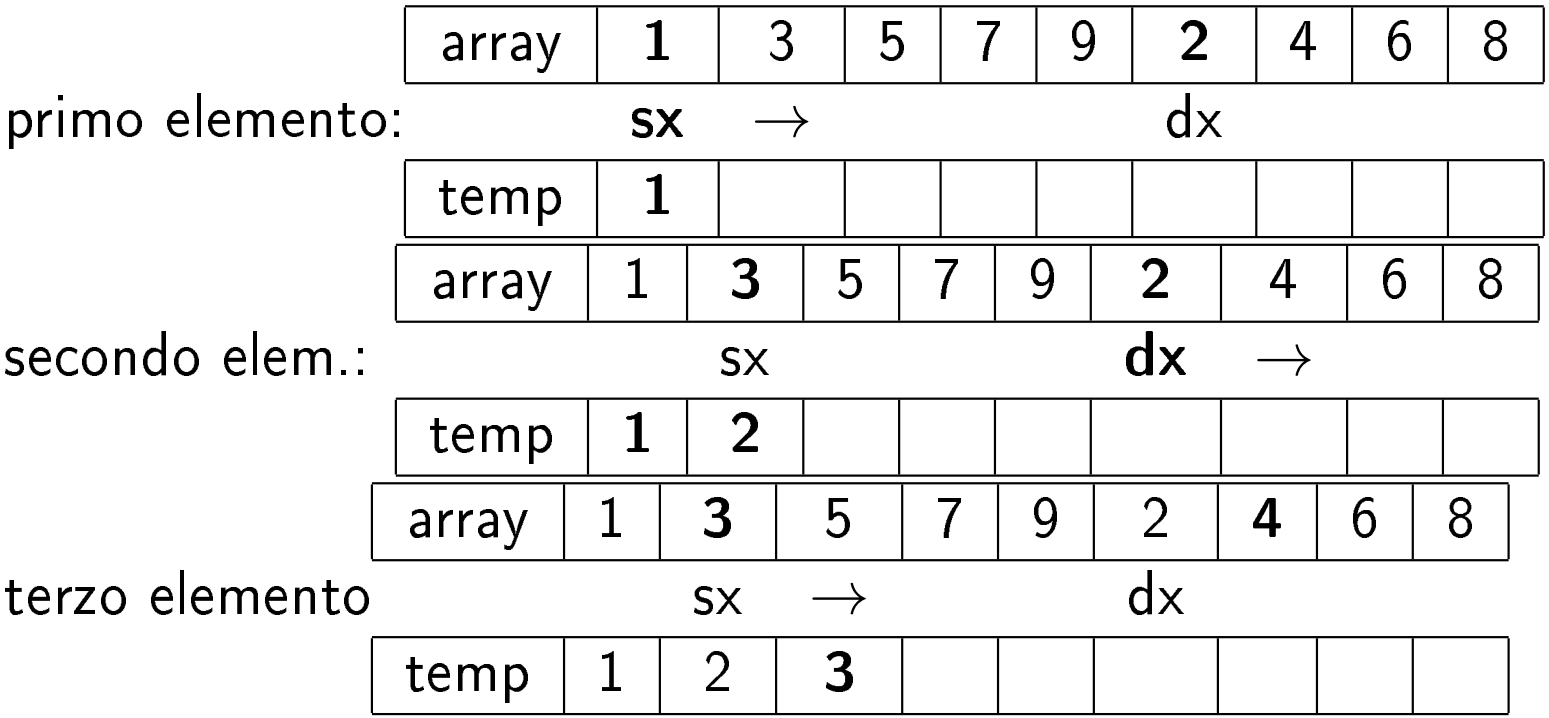
Fare il merge tra due porzioni di array



Fare il merge tra due porzioni di array



Fare il merge tra due porzioni di array



Copia nel nuovo array temp

- Copio dalla parte sinistra in due casi
  1. se la parte destra è stata copiata tutta:
     dx $>$ max
  2. oppure l'elemento a sx è minore di quello di dx e sx è ancora valido
     A[sx] $<$ A[dx] && sx $<=$ middle
  3. in totale ho:

```
if (dx > max || (A[sx] < A[dx] && sx <= middle)) {
```

- Poi l'array temp va ricopiato su A
  for (int k = 0; k $<$ temp.length; k++) A[min + k] = temp[k];

Merge

```
static void merge(int[] A, int min, int middle, int max) {
 // creo un array temporaneo per ospitare il merge da min a max
 int[] temp = new int[max - min + 1];
 int sx = min, dx = middle + 1;
 int i = 0; // indice per temp
 while (i < temp.length) {
  if (dx > max || (A[sx] < A[dx] && sx <= middle)) {
   // copio in temp l'elemento da sinistra
   temp[i] = A[sx];
   sx++;
   i++;
  } else {
   // copio elemento di destra
   temp[i] = A[dx];
   dx++;
   i++;
  }
 }
 // copia temp in A
 for (int k = 0; k < temp.length; k++)
   A[min + k] = temp[k];
}
```

Codice completo

```
static void mergesort(int[] m){
   mergesort(m,0,m.length-1);
}
static void mergesort(int[] m,
                      int min, int max){
   if (max==min) return;
   int med = (min + max) /2;
   mergesort(m,min,med);
   mergesort(m,med+1,max);
   merge(m,min,med,max);
}
```