Problema:

a partire da due sequenze ordinate v1 e v2 di elementi vogliamo costruirne una ordinata v con tutti gli elementi di v1 e v2

Algoritmo ricorsivo:

Se le due sequenze contengono elementi
confronta i primi due elementi delle sequenze, metti il più piccolo all'inizio delle nuova sequenza e richiama la funzione sul resto della seguenza dalla quale abbiamo scelto l'elemento e l'altra.
Se una delle due sequenze è terminata copia gli elementi di quella rimasta nella nuova

Esempio:

v1= 1 5 10

v2 = 2 3 7 9 20

Allora v = 1 2 3 5 7 9 10 20

Implementazione su array (vettore) in C

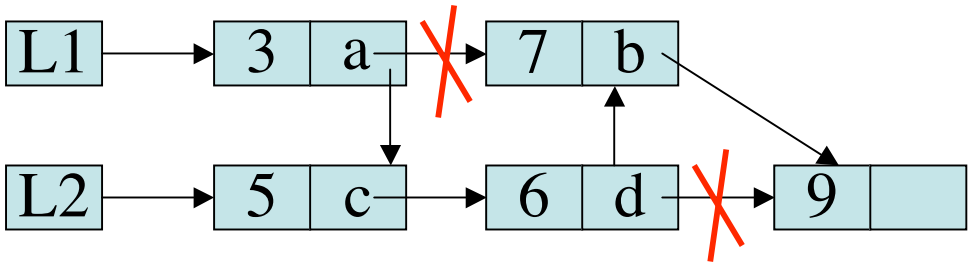
```
void merge(int *a,int*b,int *c, int n, int m)
/*  costruisce un vettore ordinato a partire da
due vettori ordinati a, con n elementi e b con m.
prec: a!= NULL && b != NULL && n >= 0 && m >= 0 &&
a[i] <= a[i+1], per 0<=i<n && b[i] <= b[i+1], 0<=i<m.
postc: return  in c  gli n+m elementi di a e b, in modo tale che
c[i] <= c[i+1], 0<=i<n.*/
{if (n == 0 && m == 0) return;
 if (n == 0 && m > 0 ) {*c = *b; merge(a,++b,++c,n,m-1);}
 else
 if (n > 0 && m == 0 )  {*c = *a; merge(++a,b,++c,n-1,m);}
 else
 if (*a < *b ) {*c = *a;  merge(++a,b,++c,n-1,m);}
 else {*c = *b; merge(a,++b,++c,n,m-1);}
}
```

N.B. la memoria per il vettore c deve essere allocata prima della chiamata

Questa funzione è tail recursive!!

Implementazione su lista concatenata in C

```
ListaPtr merge (ListaPtr L1, ListaPtr L2)
/*fonde  due liste ordinate L1 e L2 (modificando entrambe)
in un'unica lista  ordinata con gli elementi di entrambe
prec: gli elementi sono in ordine crescente nelle due liste
postc: restituisce una lista ordinata, in ordine crescente,
 che contiene  tutti gli elementi di L1 e di L2  */
{if (!L2) return L1;
if (!L1) return L2;
if (L2 -> elem < L1 -> elem)
        {L2 -> next = merge(L1,L2 ->next);return L2;}
else
        {L1 -> next = merge(L1->next,L2);return L1;}
}
```



merge(L1,L2)

3<5 : L1 -> next che è a = merge(a,L2); a diventa L2, return L1

5<7 : L2 -> next che è c = merge(a,c); il valore non cambia, return L2

6<7 : L2 -> next che è d = merge(a,d); d diventa a, return c

7<9 : L1 -> next che è b = merge(NULL,d); b diventa d, return a

merge(NULL,d); return d

Problema:

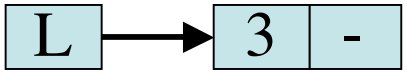
a partire da una lista concatenata v vogliamo costruire due liste concatenate ciascuna con circa la metà degli elementi di v.

Idea 1:

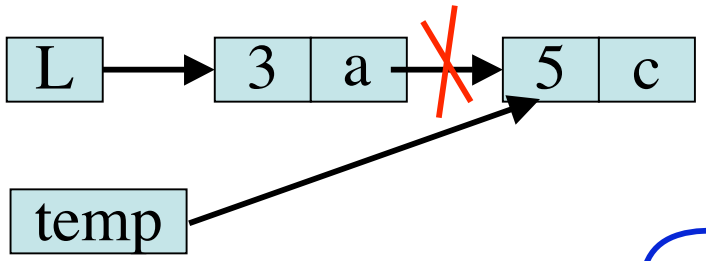
Scorro la lista costruendone un'altra con solo gli elementi di posto pari e cancellando da quella data gli elementi di posto dispari

Casi base:

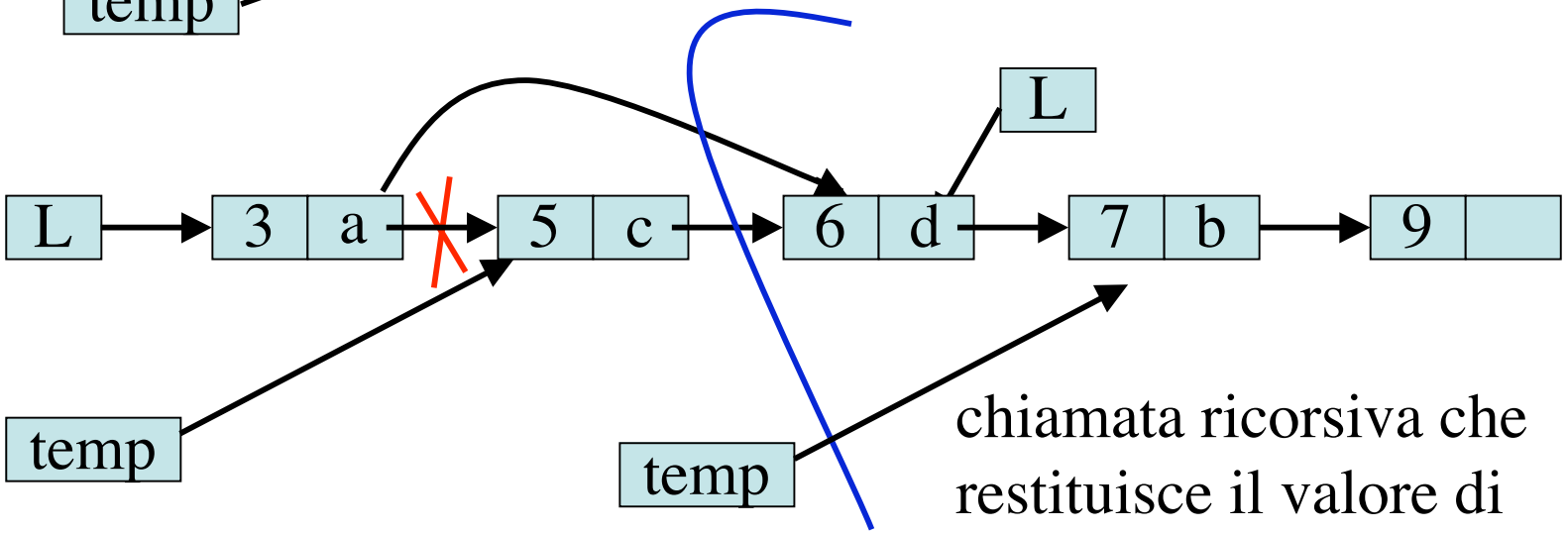
L = NULL



restituisci NULL



restituisci temp



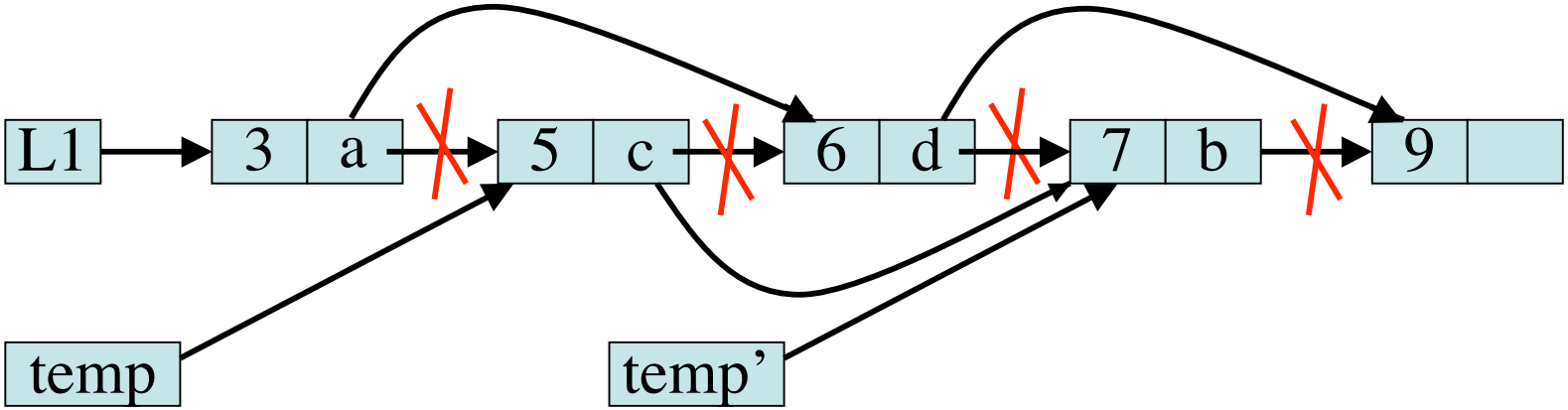
chiamata ricorsiva che restituisce il valore di temp-> next

Implementazione:

```
ListaPtr dividi(ListaPtr L)
/* divide una lista in due metà
postc: se L è non vuota  conterrà gli
elementi di posto dispari e la lista
restituita quelli di posto pari*/
{ListaPtr temp;
if (!L || !L-> next) return NULL;
temp = L-> next;
L->next = temp -> next;
temp -> next = dividi(temp -> next);
return temp;
}
```

Esempio di esecuzione:

```
ListaPtr dividi(ListaPtr L)
{ListaPtr temp;
if (!L) return L;
if (!L-> next) return NULL;
temp = L-> next;
L->next = temp -> next;
temp -> next = dividi(temp -> next);
return temp;
}
```



dividi(L1)

restituisce temp

c = dividi(c)

restituisce temp'=d quindi c = d

b = dividi(b)

restituisce NULL quindi b = NULL

Problema:

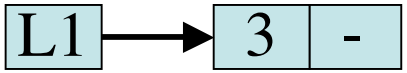
a partire da una lista concatenata v vogliamo costruire due liste concatenate ciascuna con circa la metà degli elementi di v.

Idea 2:

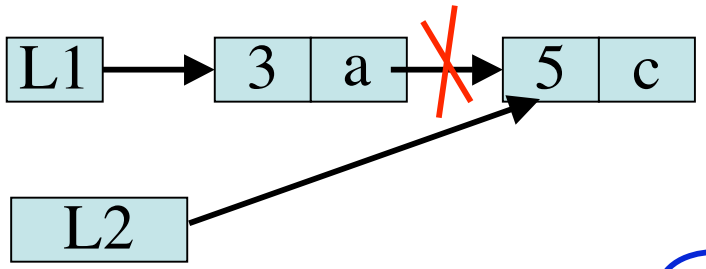
Scorri la lista con due puntatori d'appoggio, di cui uno scorre saltando un elemento, quando questo ha raggiunto la fine della lista l'altro è situato al centro della lista quindi si "stacca" la lista ottenendone una con la prima metà degli elementi e la seconda i rimanenti.

Casi base:

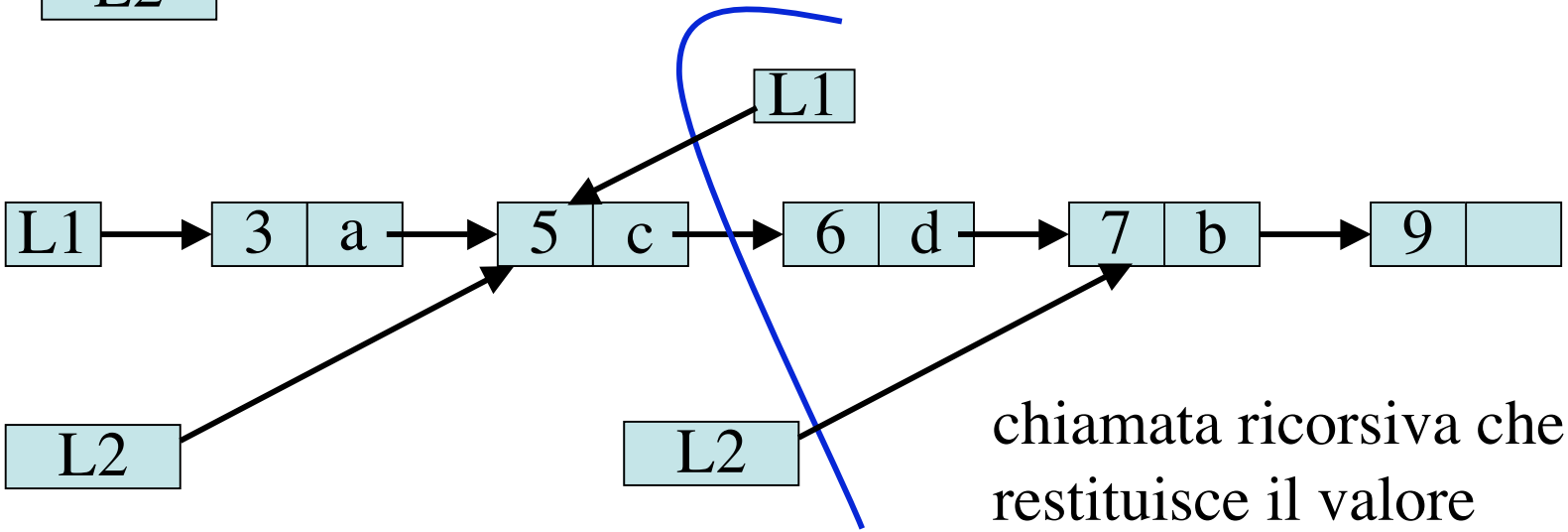
L1 = NULL



restituisci NULL



restituisci L2



chiamata ricorsiva che
restituisce il valore
L1->next salvato in L2

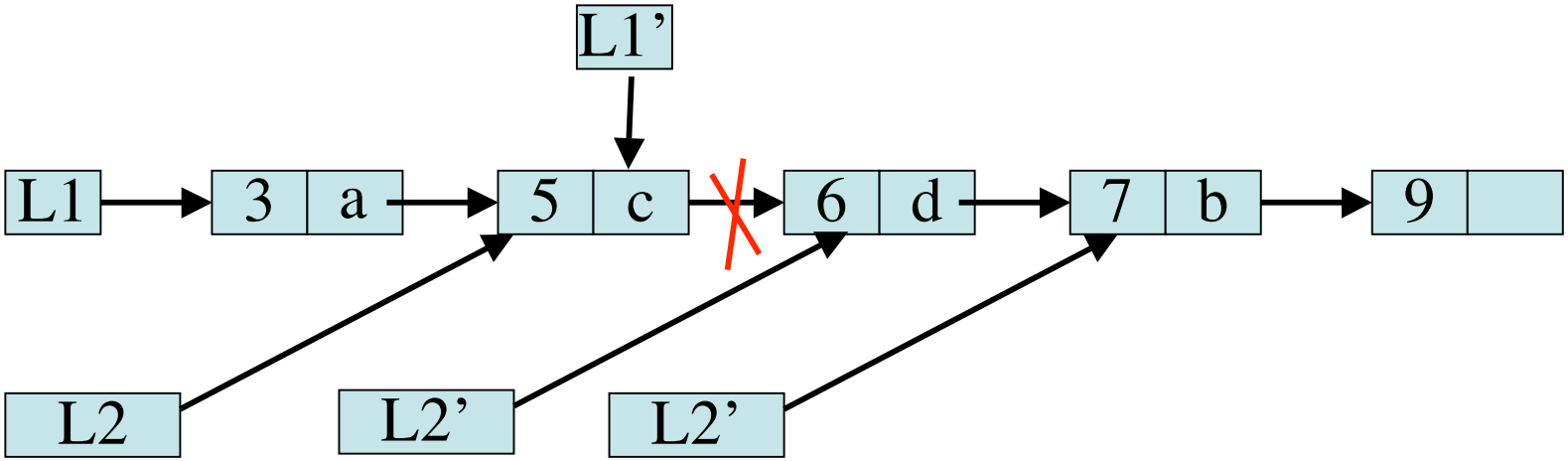
Implementazione

```
ListaPtr  dividi2( ListaPtr L1, ListaPtr L2)
/* divide la lista L in due, L2 va
inizializzato a L1->next;
prec: L1!= NULL
postc: L1 conterrà la prima metà circa degli
elementi e quella restituita la seconda.*/
{if (!L2 || !L2 -> next )
 {L2 = L1->next;
  L1 -> next = NULL;
  return L2;
 }
return  dividi2(L1->next,L2 -> next -> next );}
```

E’ tail recursive

Esempio di esecuzione:

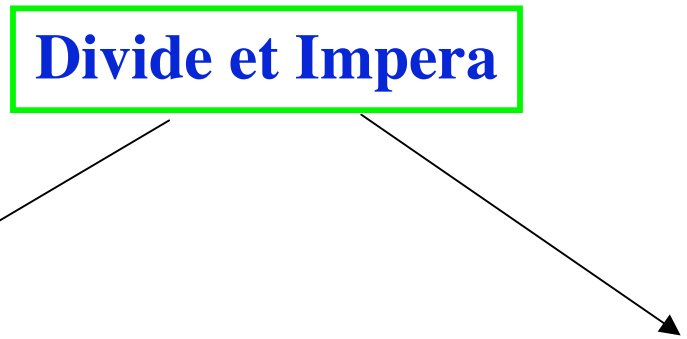
```
ListaPtr  dividi2( ListaPtr L1, ListaPtr L2)
{if (!L2 || !L2 -> next )
 {L2 = L1->next;
  L1 -> next = NULL;
  return L2;
 }
return  dividi2(L1->next,L2 -> next -> next );}
```



dividi2(L1,L2) restituisce L2

L2 = dividi2(a,d) restituisce L2

L2 -> next = NULL L2 = c, L1 - > next = NULL e restituisce L2



Quicksort

Charles Antony Richard Hoare
(Prof. Emerito alla Oxford University)
Computer Jurnal 5,1,1962

Mergesort

John von Neumann(1903-1957)
Nel 1944, il suo rapporto interno "First Draft of a Report on the EDVAC" contiene tra l'altro, il mergesort come programma di ordinamento

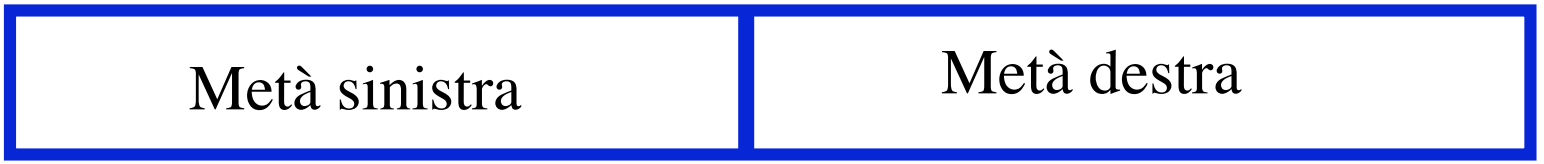
MergeSort

- Esempio di algoritmo basato su Divide et Impera

- Due fasi:
  - Fase di suddivisione
    - Divide il lavoro a metà
  - Fase di ordinamento (fusione)
    - Impera sulle due metà!

MergeSort

- Dividi
  - Dividi la lista in due metà



MergeSort

- Impera

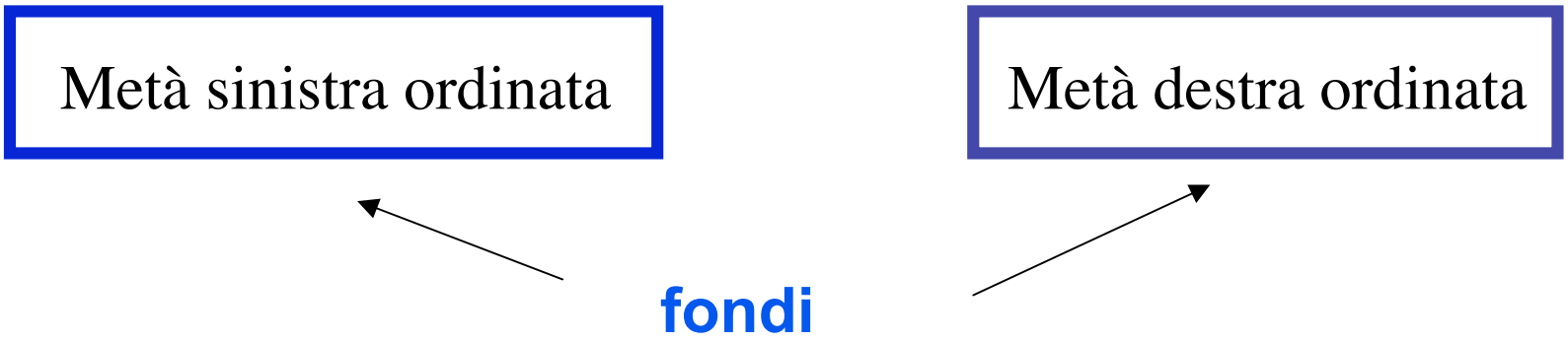
  1 Applica lo stesso algoritmo a ogni metà

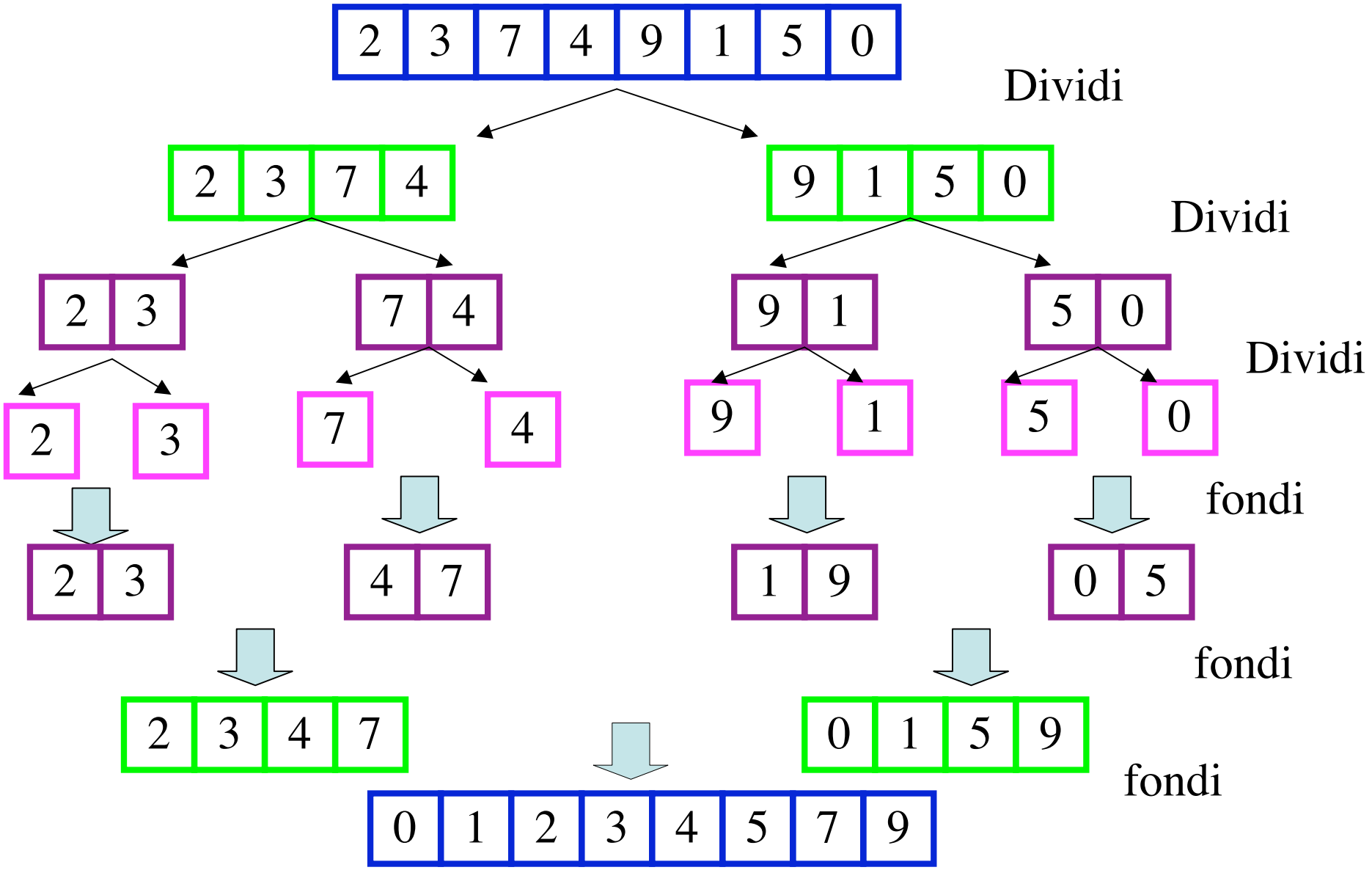
MergeSort

- Impera

  2 (a partire da quando si ha un solo elemento o nessuno)

  fondi




2 3 7 4 9 1 5 0
Dividi
2 3 7 4
9 1 5 0
Dividi
2 3
7 4
9 1
5 0
Dividi
2
3
7
4
9
1
5
0
fondi
2 3
4 7
1 9
0 5
fondi
2 3 4 7
0 1 5 9
fondi
0 1 2 3 4 5 7 9