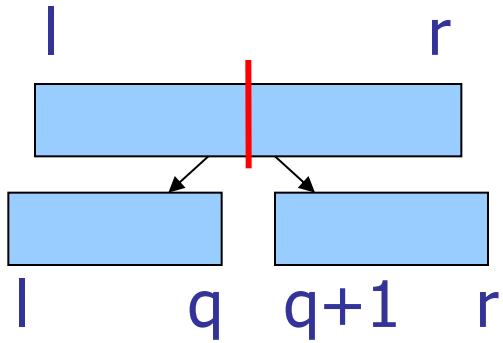
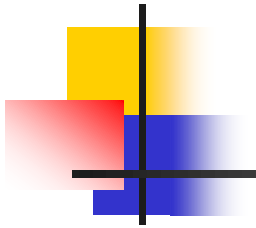
Gli algoritmi ricorsivi di ordinamento

Paolo Camurati
Dip. Automatica e Informatica
Politecnico di Torino

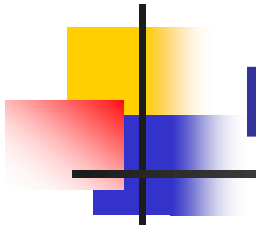
Merge Sort (von Neumann, 1945)

- Divisione:
  - due sottovettori SX e DX rispetto al centro del vettore.





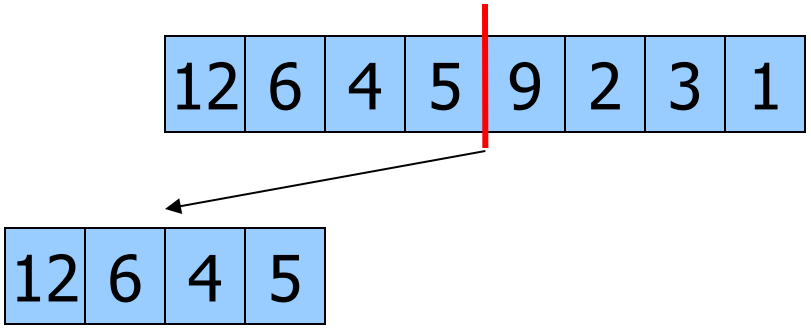
- Ricorsione
  - merge sort su sottovettore SX
  - merge sort su sottovettore DX
  - condizione di terminazione: con 1 (l=r) o 0 (l>r) elementi è ordinato
- Ricombinazione:
  - fondi i due sottovettori ordinati in un vettore ordinato.



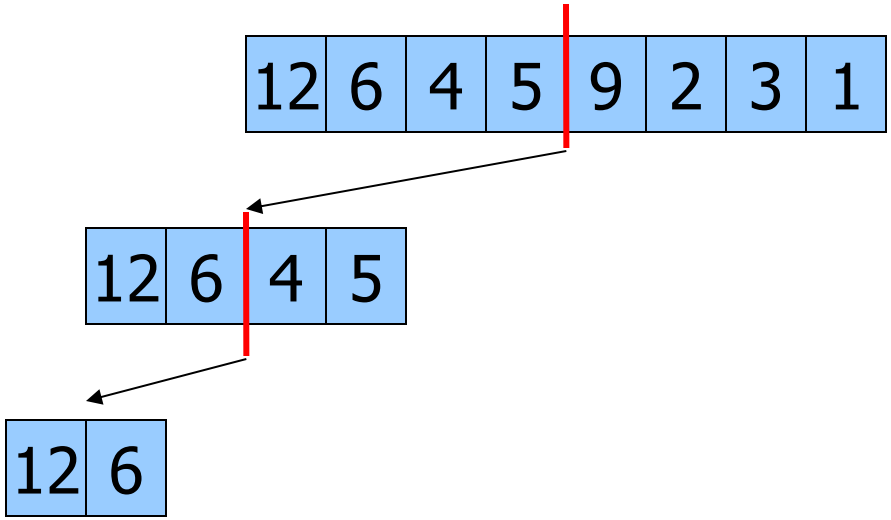
Esempio

12	6	4	5	9	2	3	1

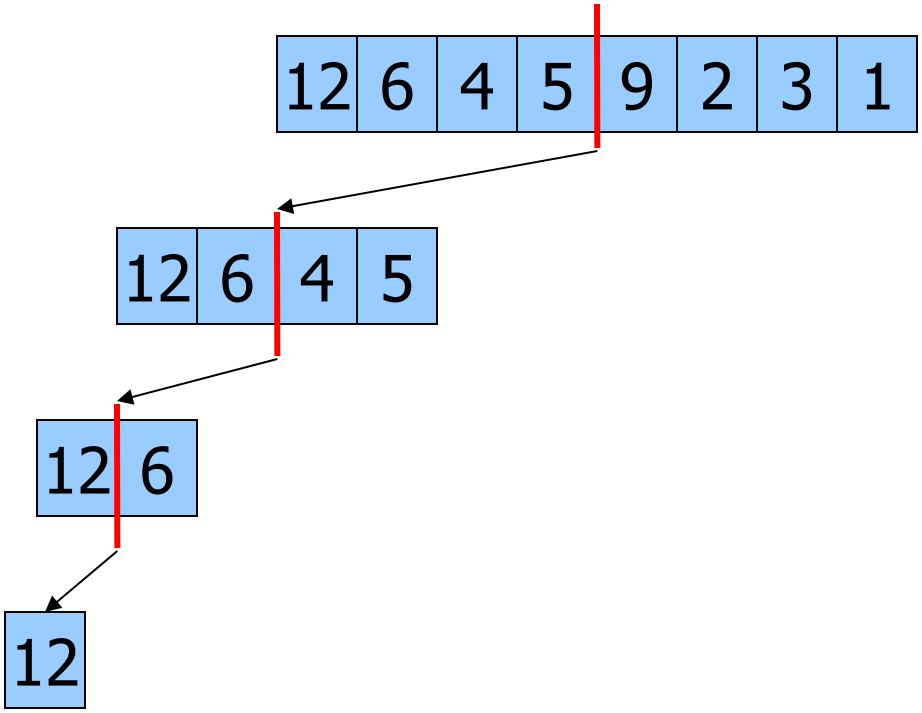
Divisione ricorsiva:



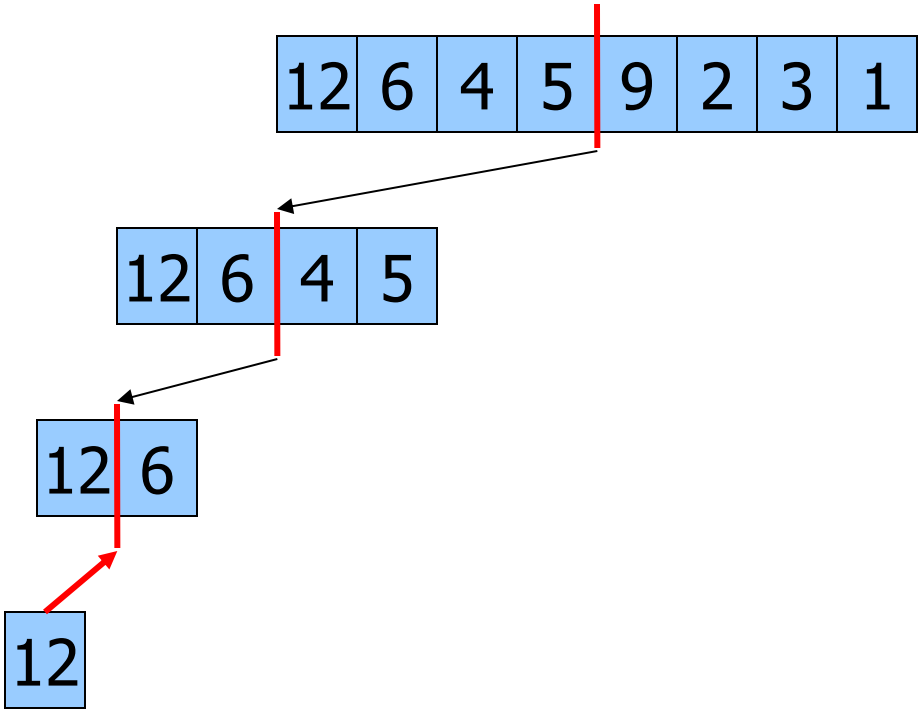
Divisione ricorsiva:



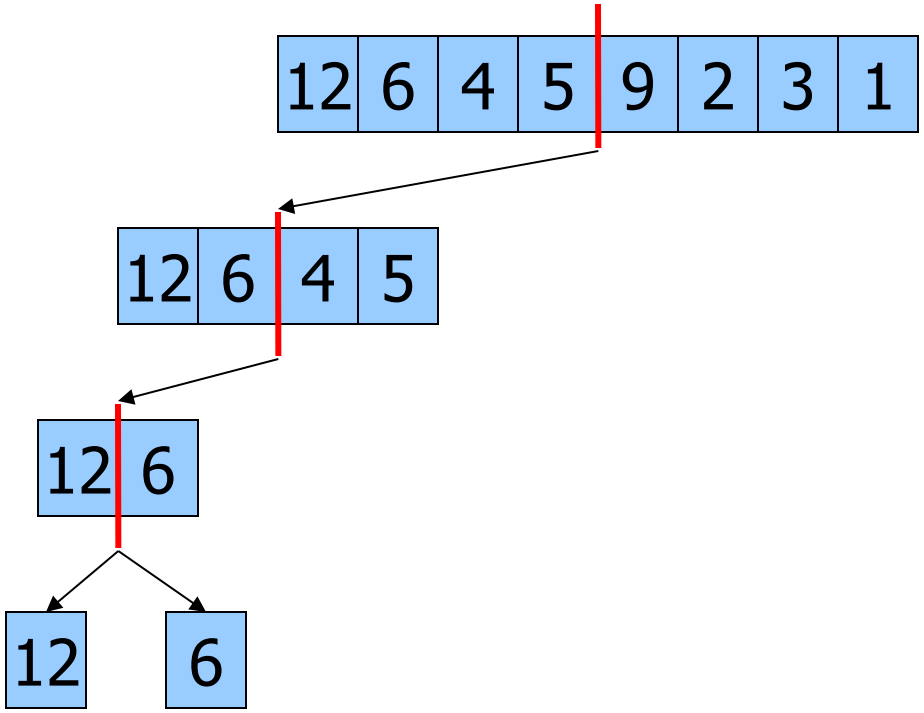
Divisione ricorsiva:



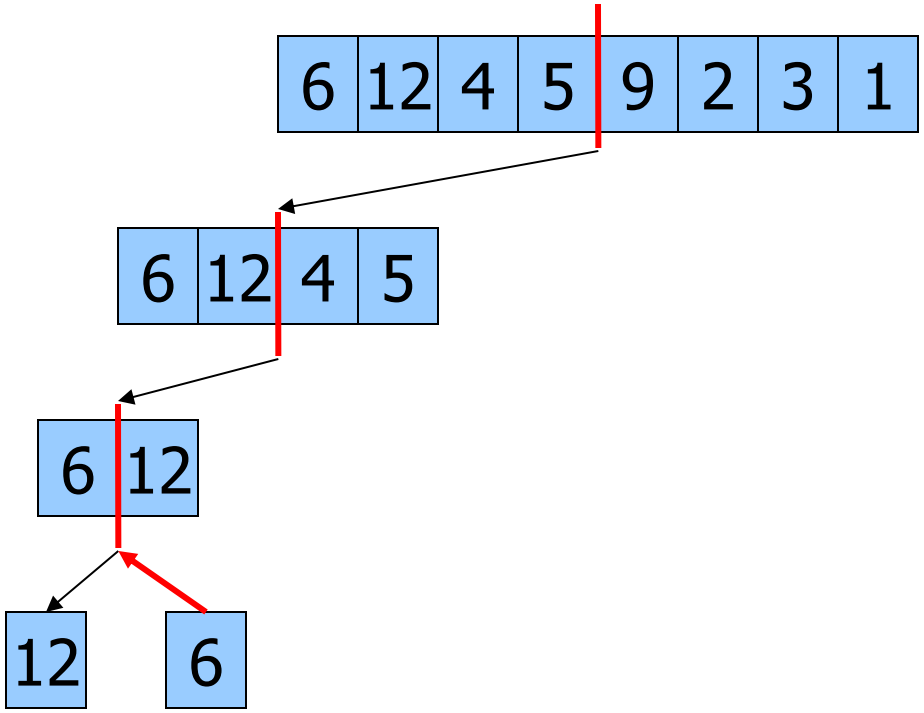
Ricombinazione:



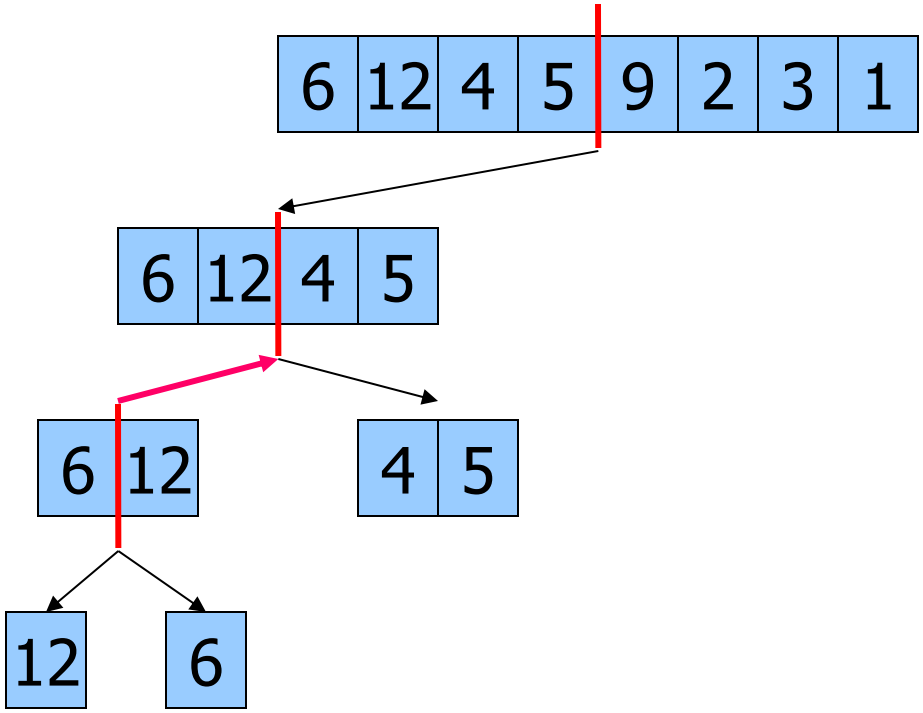
Divisione ricorsiva:



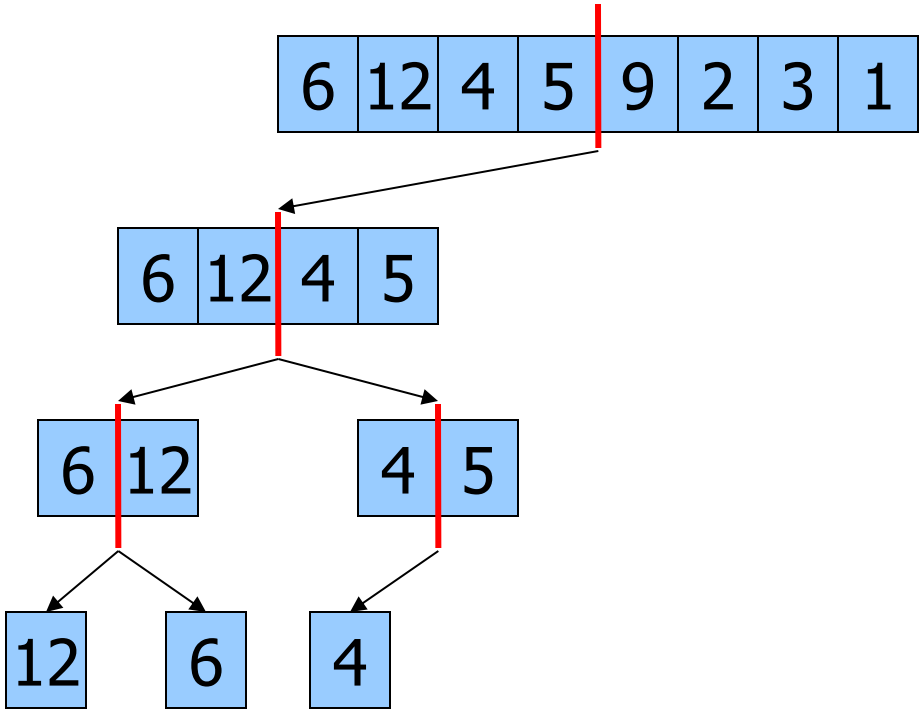
Ricombinazione:



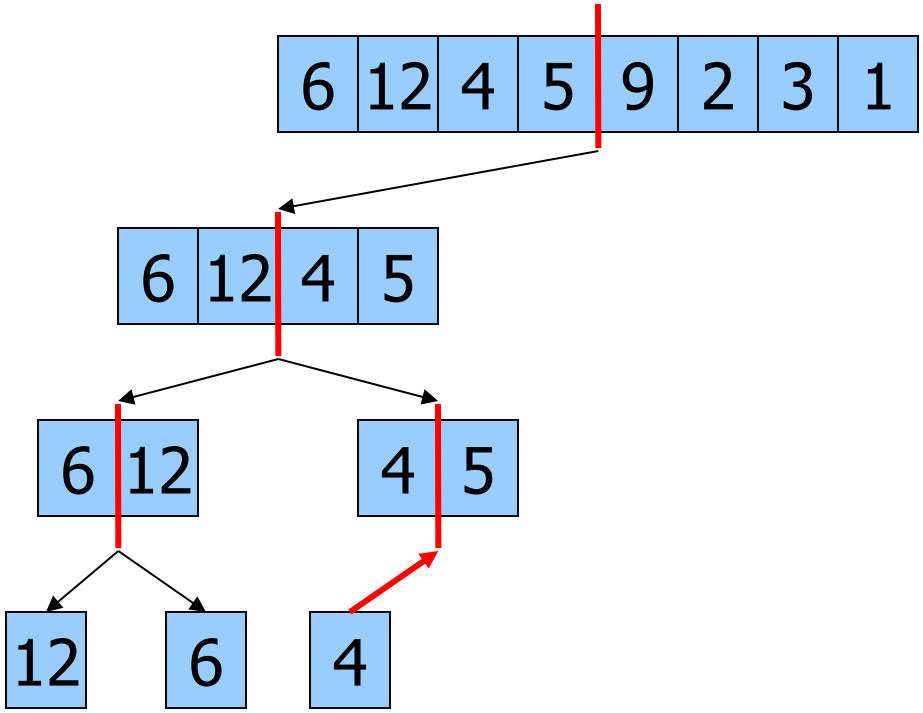
Divisione ricorsiva:



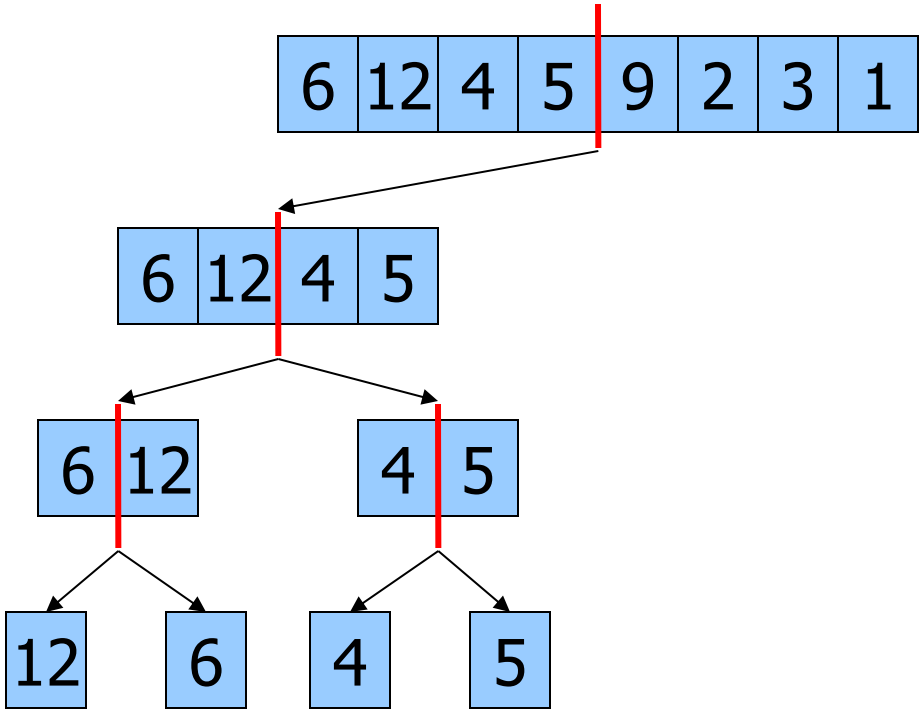
Divisione ricorsiva:



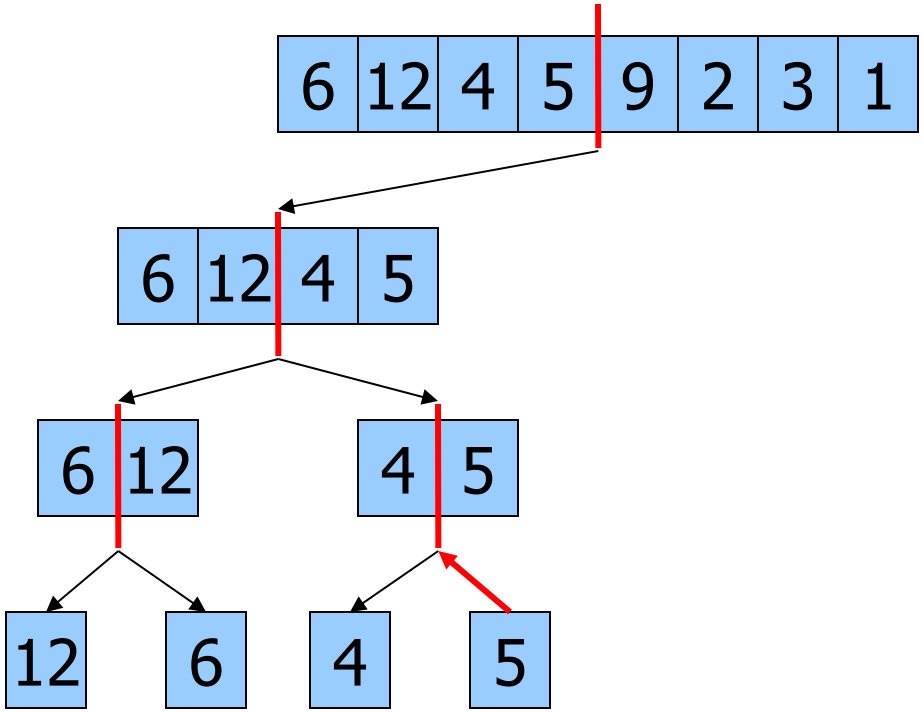
Ricombinazione:



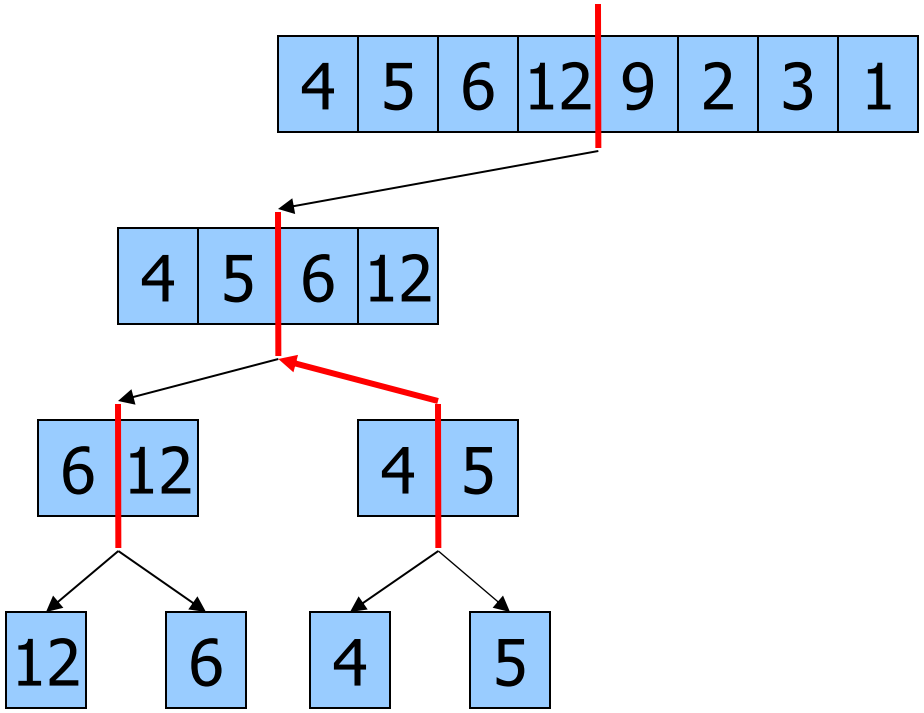
Divisione ricorsiva:



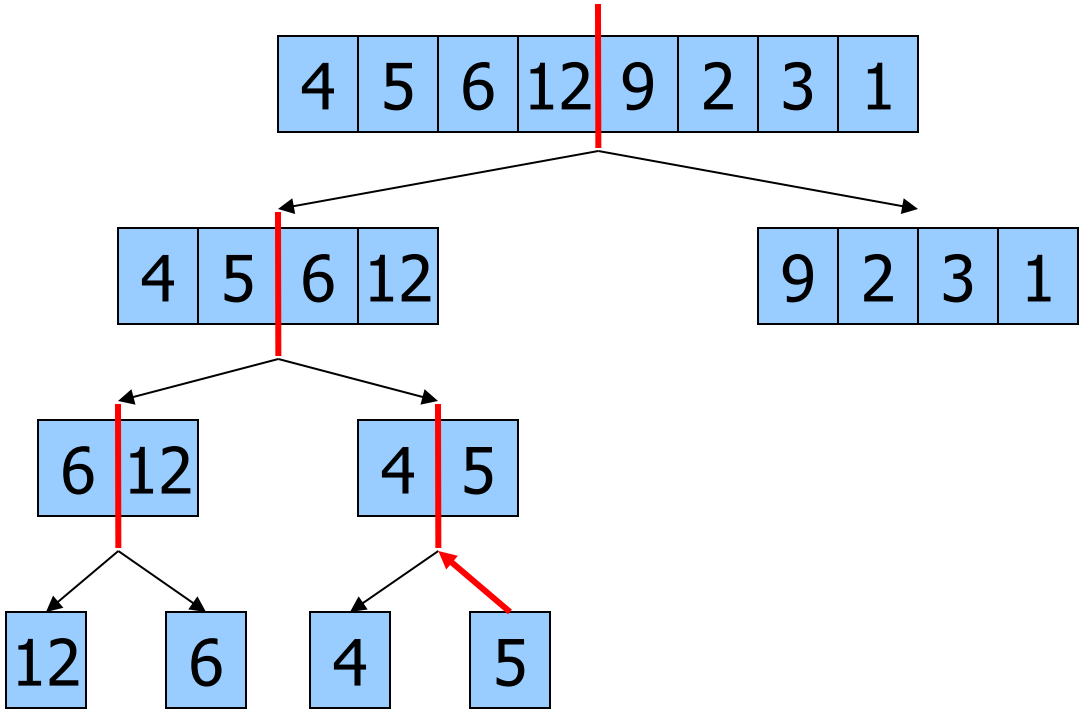
Ricombinazione:



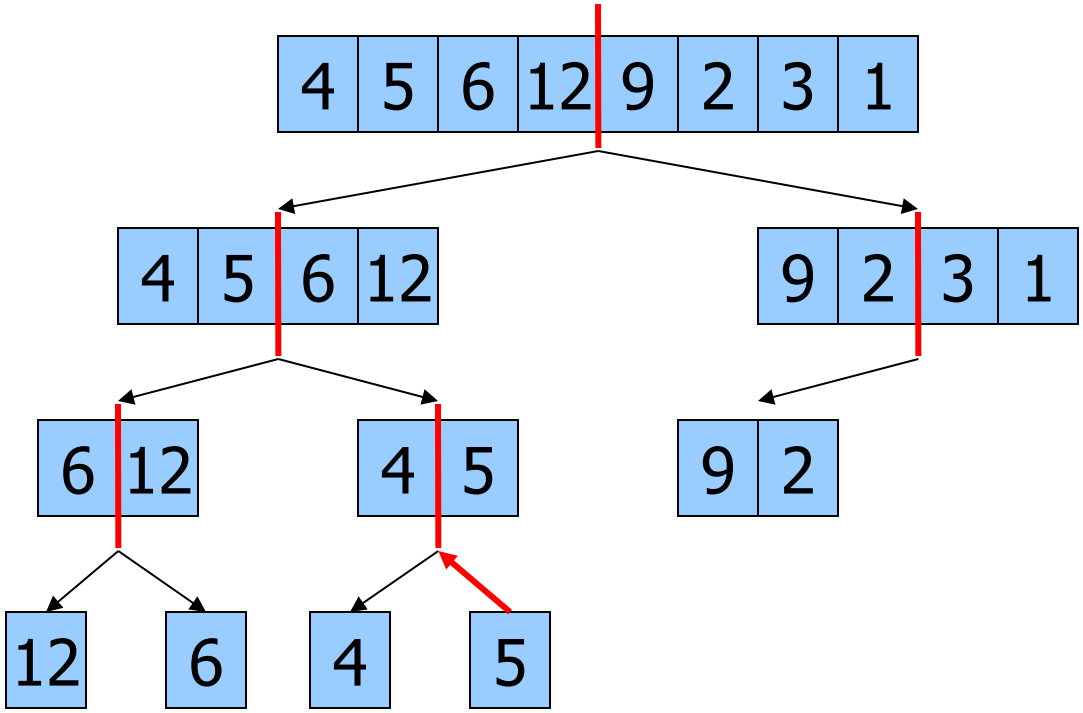
Ricombinazione:



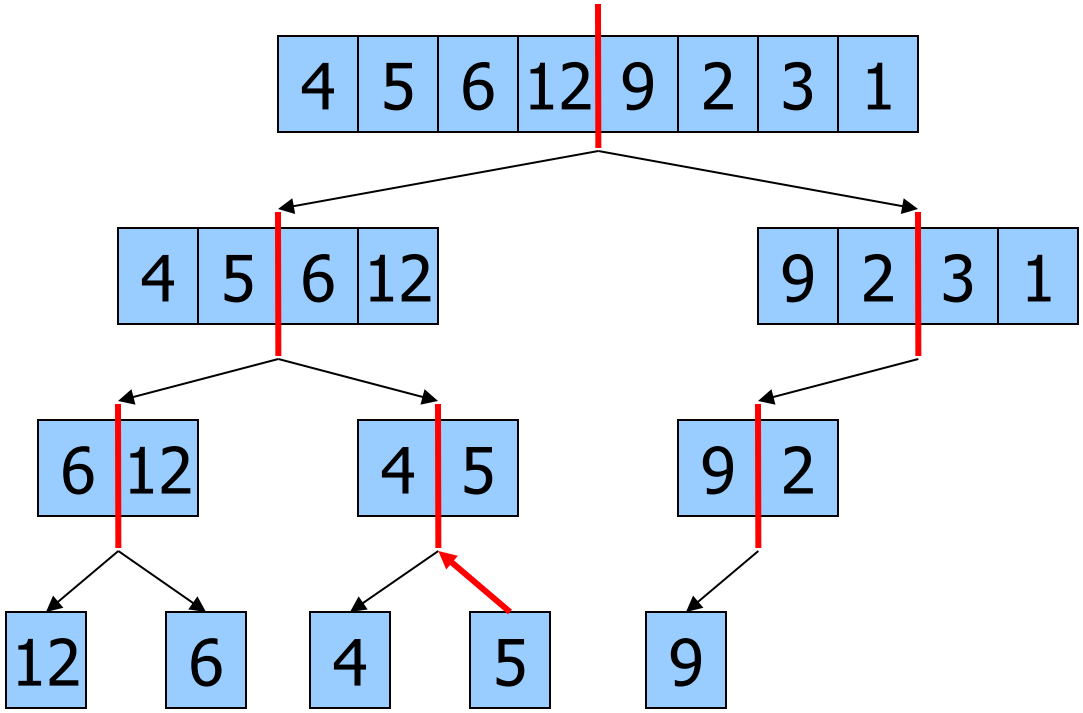
Discesa ricorsiva:



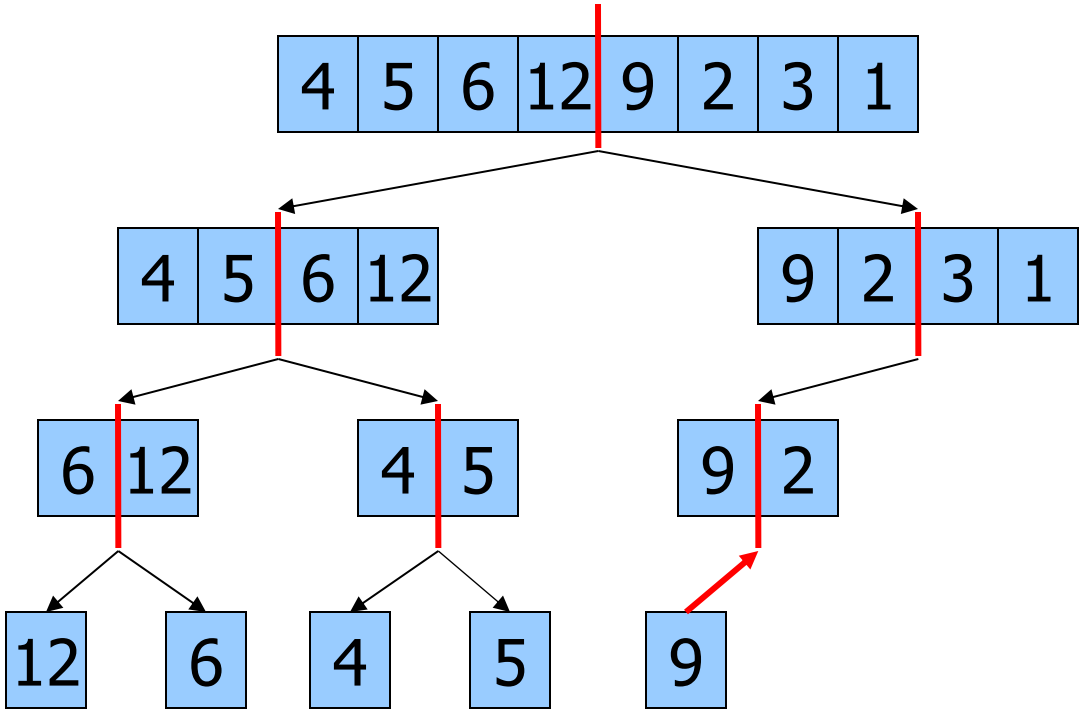
Discesa ricorsiva:



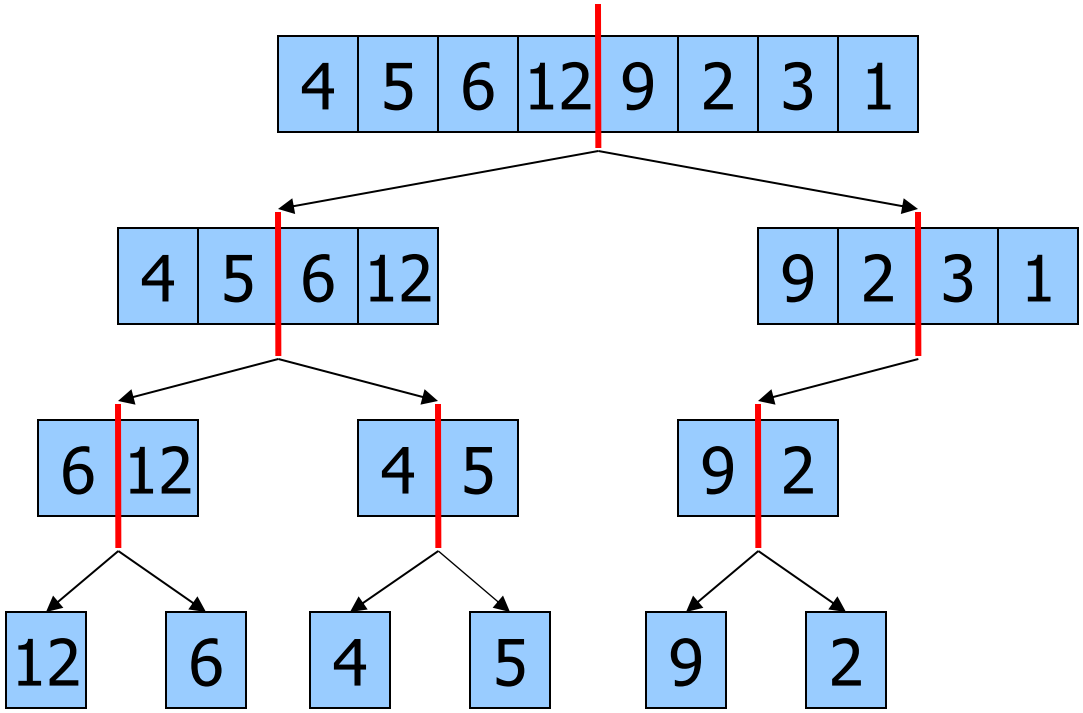
Discesa ricorsiva:



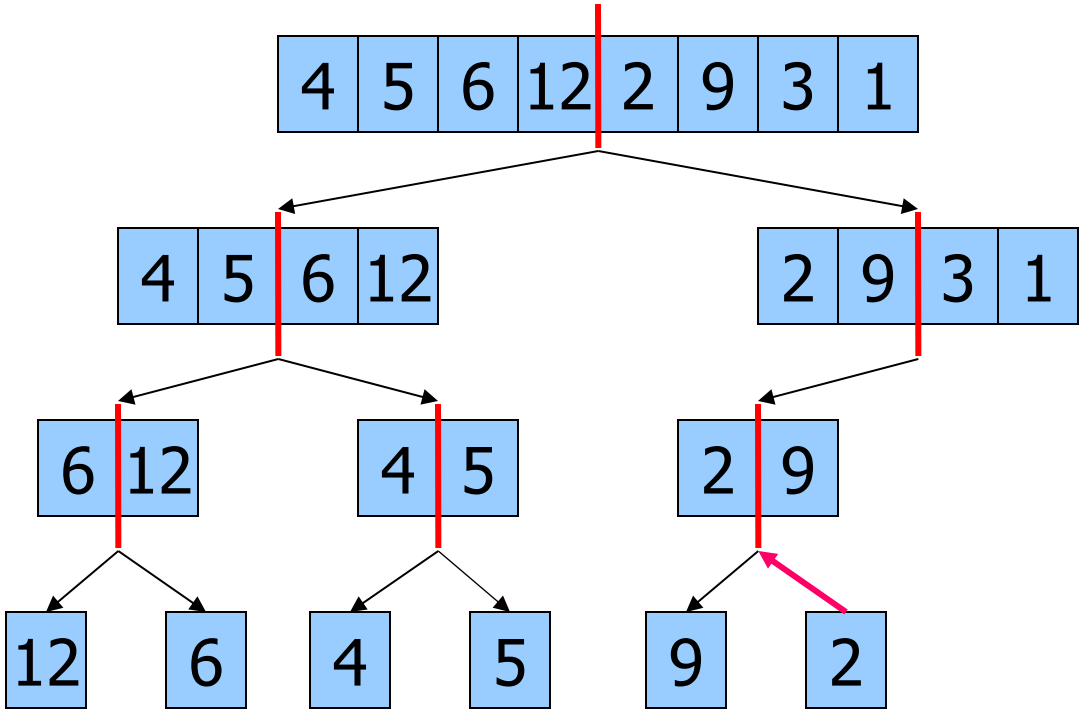
Ricombinazione:



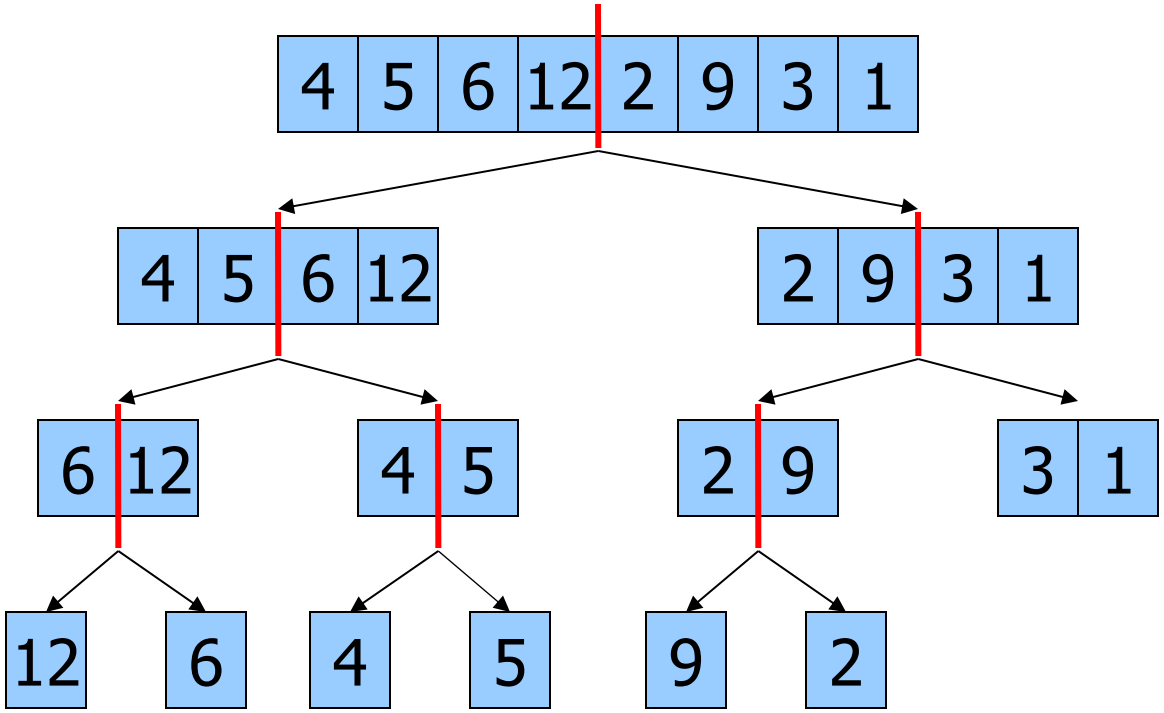
Discesa ricorsiva:



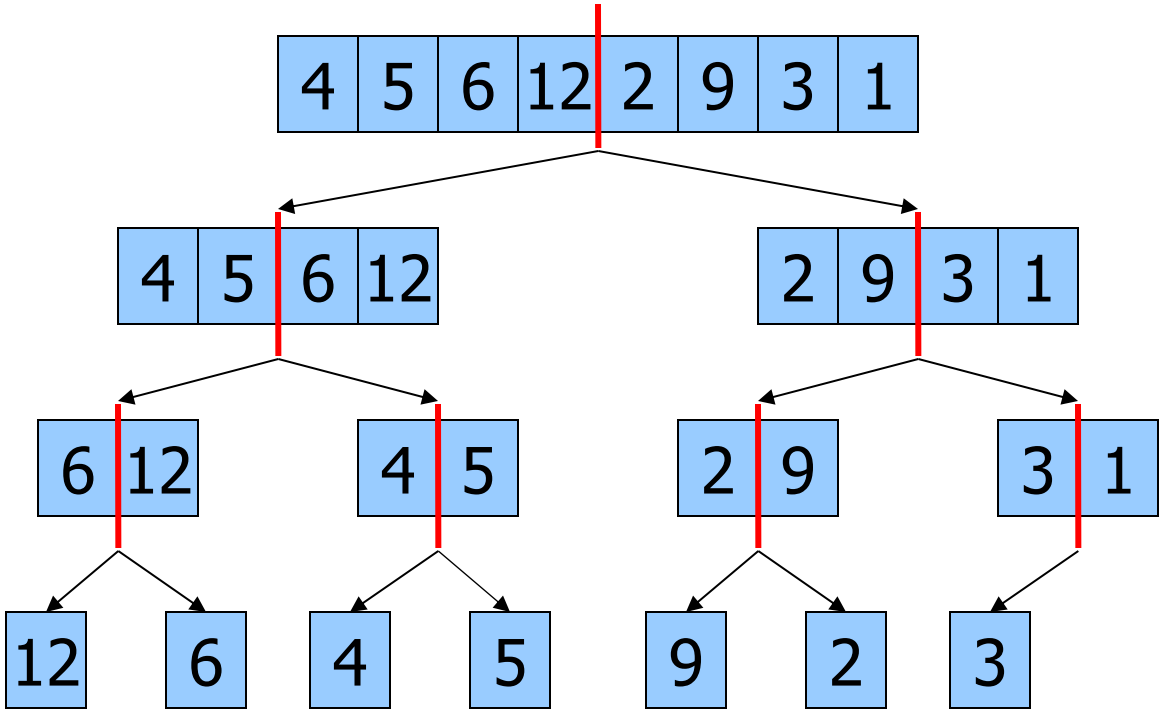
Ricombinazione:



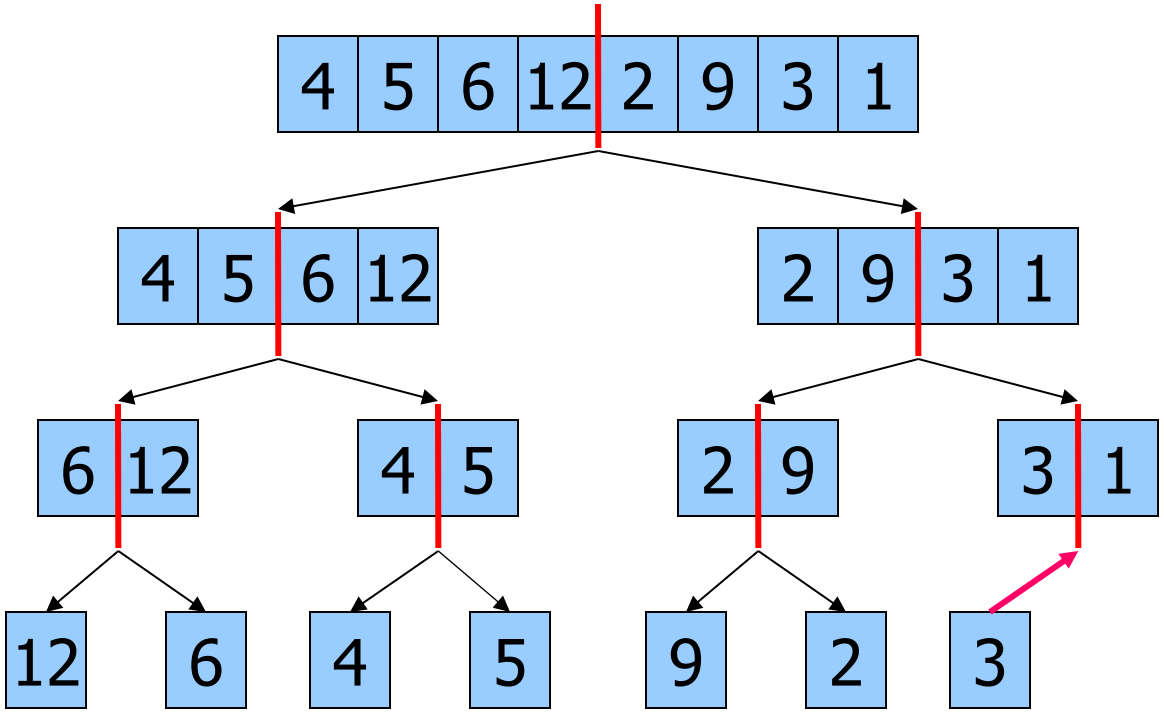
Discesa ricorsiva:



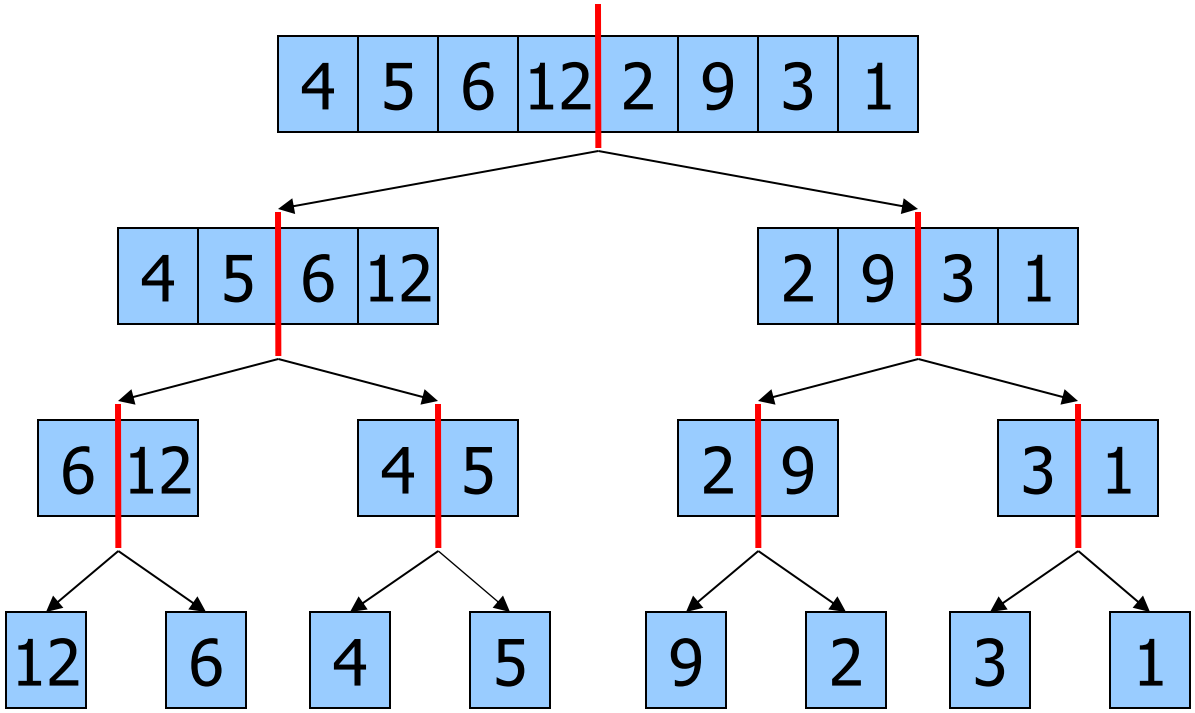
Discesa ricorsiva:



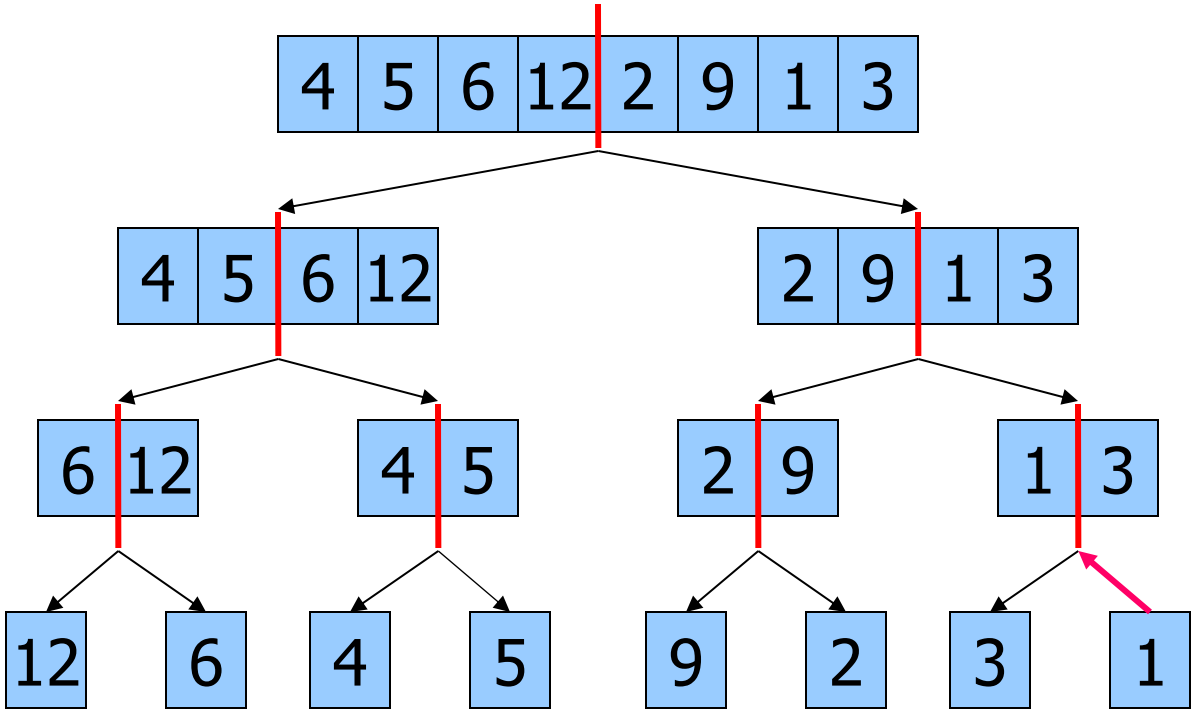
Ricombinazione:



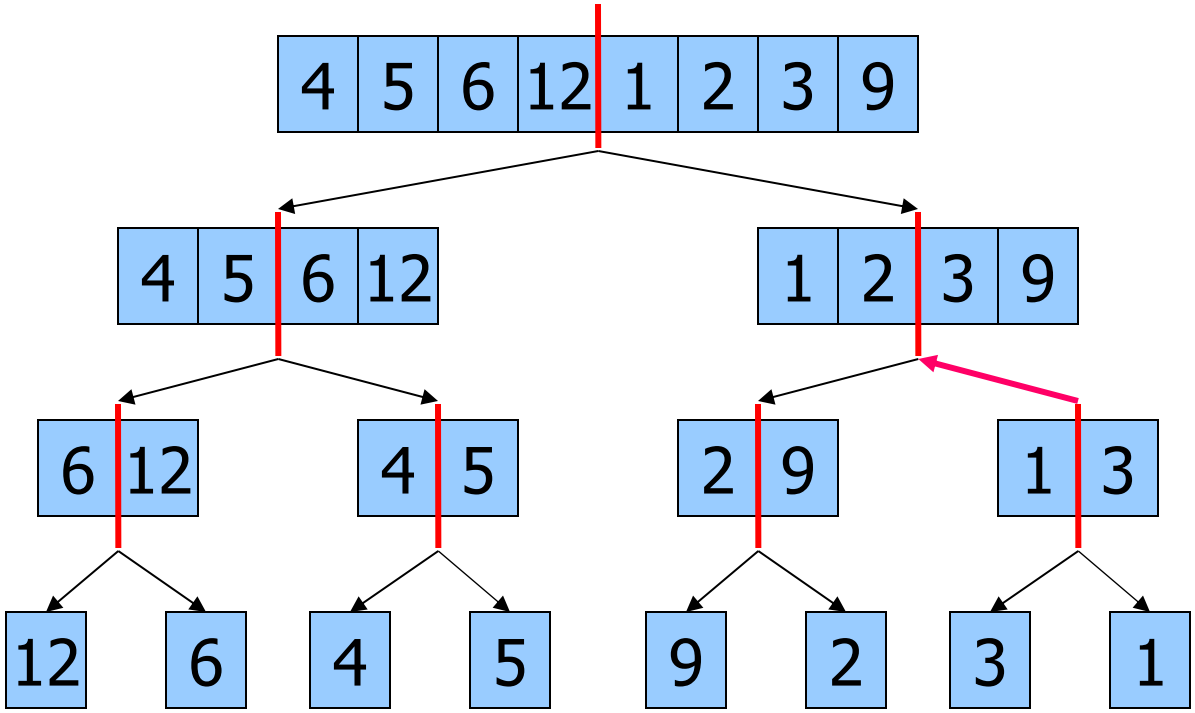
Discesa ricorsiva:



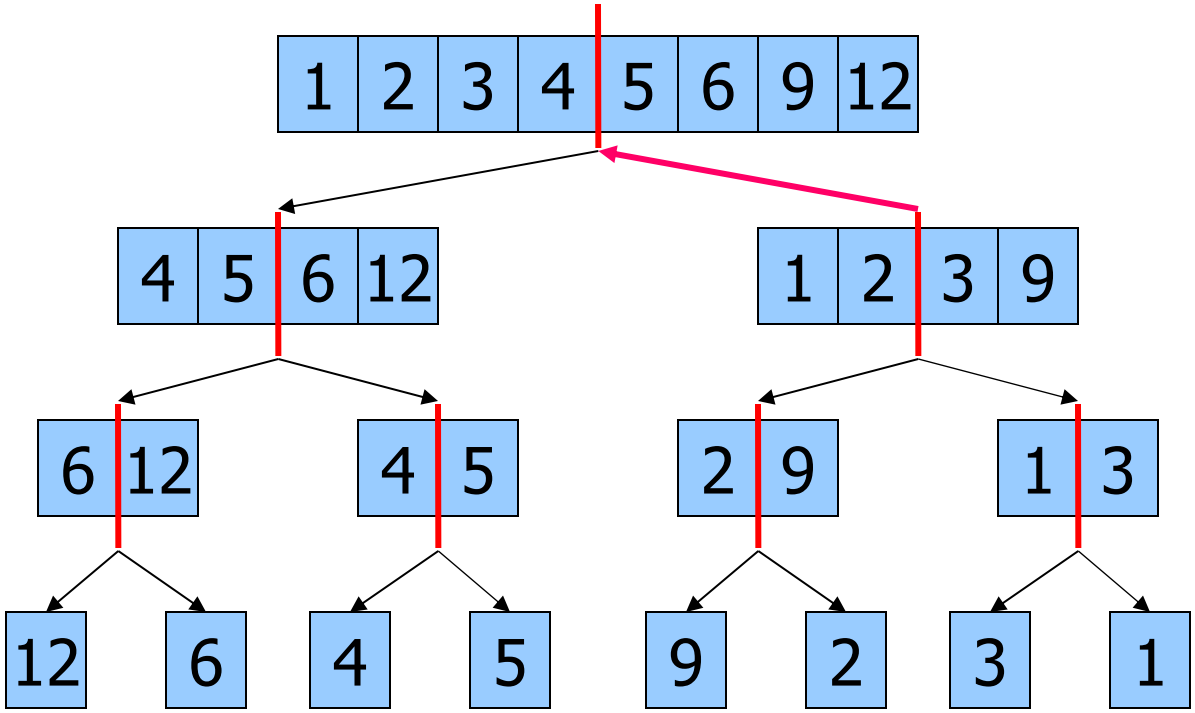
Ricombinazione:



Ricombinazione:



Ricombinazione:



RecursiveSort.c

wrapper

estremi

vettore ausiliario

```
void MergeSort(Item *A, int N)
  int l=0, r=N-1;
  Item *B = (Item *)malloc(N*sizeof(Item));
  MergeSortR(A, B, l, r);
}
```

chiamata alla funzione ricorisva

```
void MergeSortR(Item *A, Item *B, int l
  int q = (l + r)/2;
  if (r <= l)
    return;
  MergeSortR(A, B, l, q);
  MergeSortR(A, B, q+1, r);
  Merge(A, B, l, q, r);
}
```

divisione

terminazione

chiamata ricorsiva

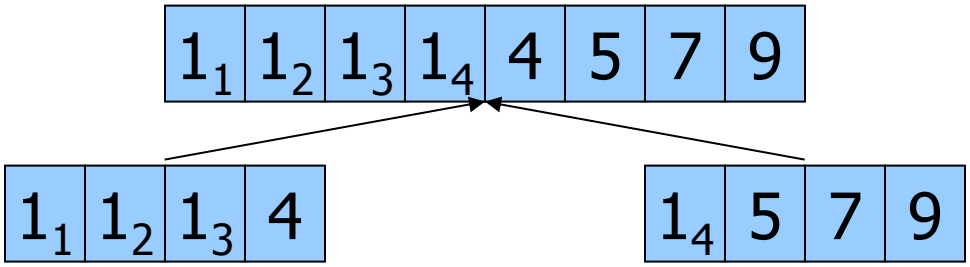
chiamata ricorsiva

ricombinazione


```
void Merge(Item *A, Item *B, int l, int q, int r) {
  int i, j, k;
  i = l;
  j = q+1;
  for (k = l; k <= r; k++)
    if (i > q)
      B[k] = A[j++];
    else if (j > r)
      B[k] = A[i++];
    else if ( less(A[i], A[j]) || eq(A[i], A[j])  )
      B[k] = A[i++];
    else
      B[k] = A[j++];
  for ( k = l; k <= r; k++ )
    A[k] = B[k];
  return;
}
```

Caratteristiche

- Non in loco (usa un vettore ausiliario)
- Stabile: in quanto la funzione merge prende dal sottovettore SX in caso di chiavi uguali:

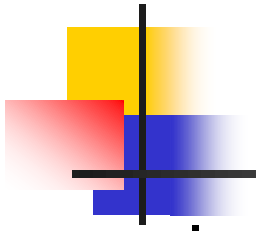


Analisi asintotica di caso peggiore

Ipotesi: $n = 2^k$ solo ai fini dell'analisi.

- Dividi: calcola la metà di un vettore $D(n)=\Theta(1)$
- Risolvi: risolve 2 sottoproblemi di dimensione n/2 ciascuno $2T(n/2)$
- Terminazione: semplice test $\Theta(1)$
- Combina: basata su Merge $C(n) = \Theta(n)$
- $C(n) + D(n) = \Theta(n)$
- Equazione alle ricorrenze:

$$T(n) = 2T(n/2) + n \qquad n>1$$
$$T(1) = 1 \qquad n=1$$



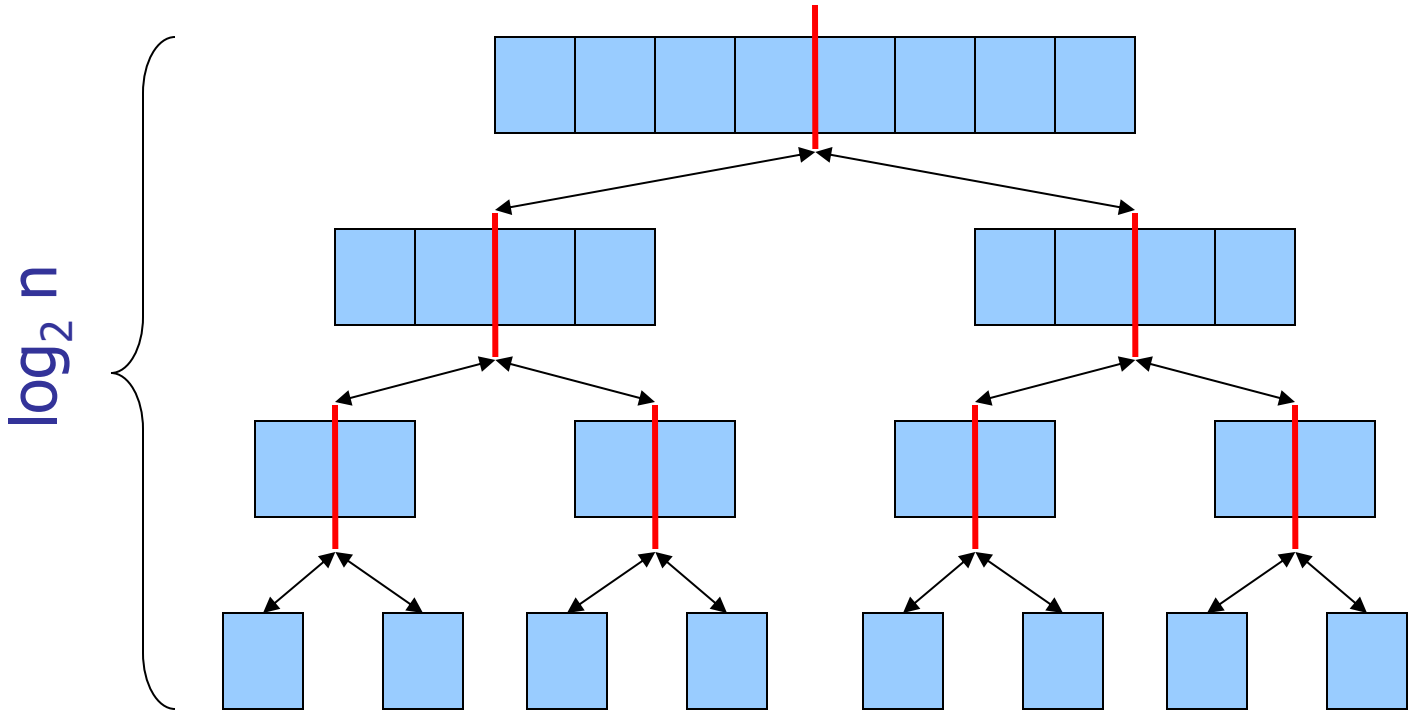
- soluzione per sviluppo (unfolding)

$T(n/2) = 2T(n/4) + n/2 \quad T(n/4) = 2T(n/8) + n/4$ etc.

Terminazione: a ogni passo i dati si dimezzano, dopo i passi sono $n/2^i$. Si termina per $n/2^i = 1$, $i = \log_2 n$

$$T(n) = n + 2*(n/2) + 2^2 *(n/4) + 2^3 *T(n/8)$$
$$= \Sigma_{0 \le i \le \log_2 n} 2^i / 2^i * n = n * \Sigma_{0 \le i \le \log_2 n} 1$$
$$= n*(1 + \log_2 n) \quad n \log_2 n + n$$
$$T(n) = O(n \log n)$$

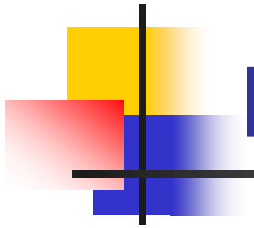
Intuitivamente:



Livelli di ricorsione: $\log_2 n$

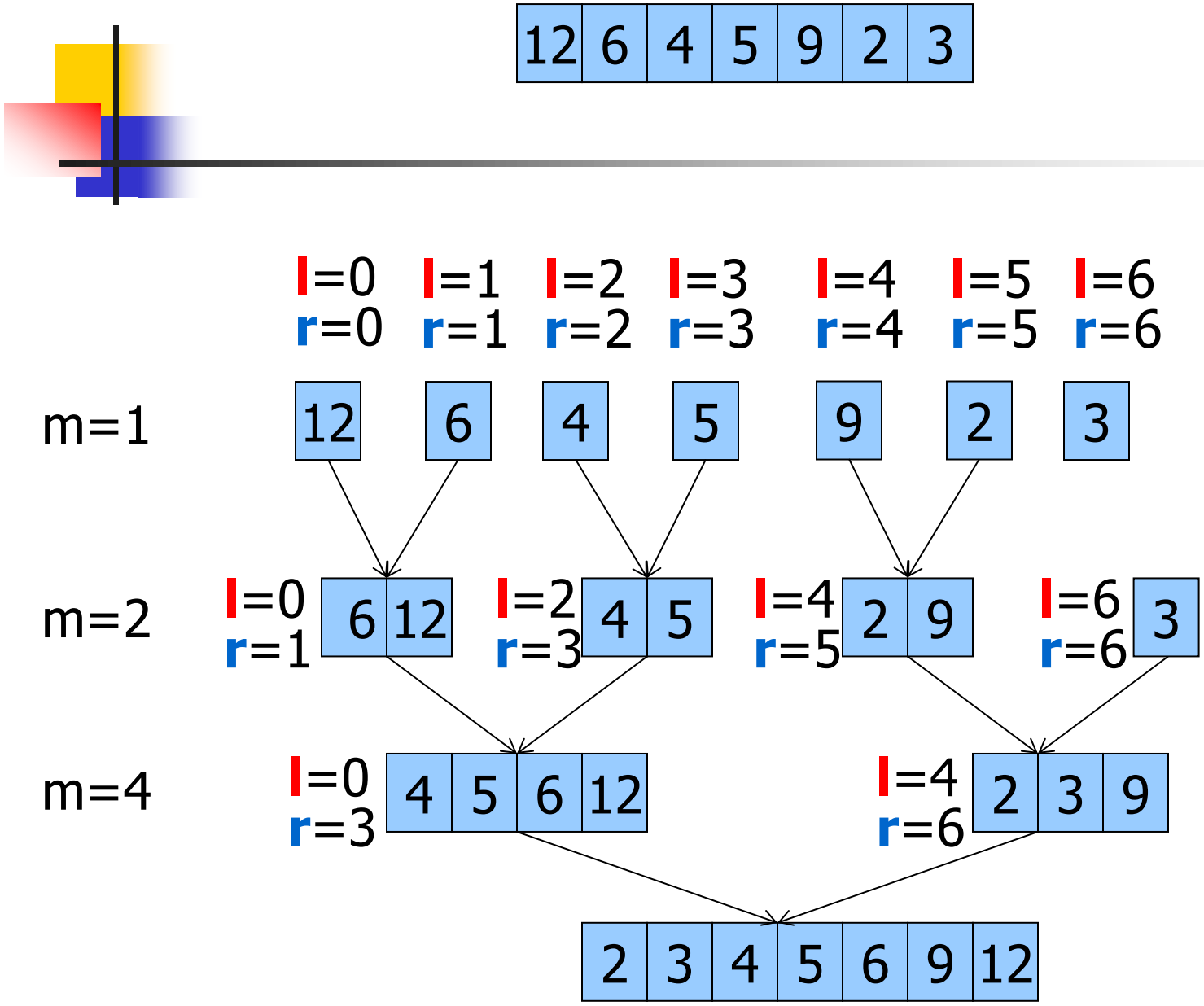
Operazioni per livello: n

Operazioni totali: $n \log_2 n$



Bottom-up Merge Sort

- Versione non ricorsiva
- Partendo da sottovettori di lunghezza 1 (quindi ordinati), si applica `Merge` per ottenere a ogni passo vettori ordinati di lunghezza doppia
- Terminazione: il vettore ha dimensione pari a quello di partenza.


12 6 4 5 9 2 3
l=0 r=0
l=1 r=1
l=2 r=2
l=3 r=3
l=4 r=4
l=5 r=5
l=6 r=6
m=1
12 6 4 5 9 2 3
m=2
l=0 r=1
6 12
l=2 r=3
4 5
l=4 r=5
2 9
l=6 r=6
3
m=4
l=0 r=3
4 5 6 12
l=4 r=6
2 3 9
2 3 4 5 6 9 12

RecursiveSort.c

```
int min(int A, int B) {
   if (A < B)
      return A;
   else
      return B;
}
void BottomUpMerge(Item A[], int N) {
  int i, m, l=0, r=N-1;
  Item *B = (Item *)malloc(N*sizeof(Item));
  for (m = 1; m <= r - l; m = m + m)
    for (i = l; i <= r - m; i += m + m)
      Merge(A, B, i, i+m-1, min(i+m+m-1,r));
}
```

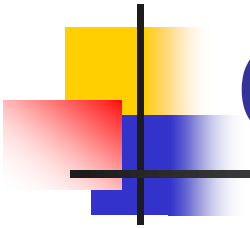
estremi

vettore ausiliario

raddoppia la dimensione del vettore ordinato

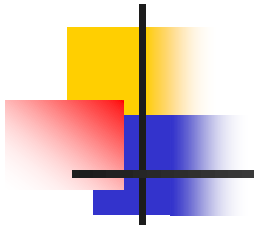
considera le coppie di sottovettori ordinati di dimensione m

fondile



Quicksort (Hoare, 1961)

- Divisione:
  - partiziona il vettore A[l..r] in due sottovettori SX e DX:
    - dato un elemento pivot x = A[q]
    - SX A[l..q-1] contiene tutti elementi < x
    - DX A[q+1..r] contiene tutti elementi > x
    - A[q] si trova al posto giusto
    - la divisione non è necessariamente a metà, a differenza del mergesort.



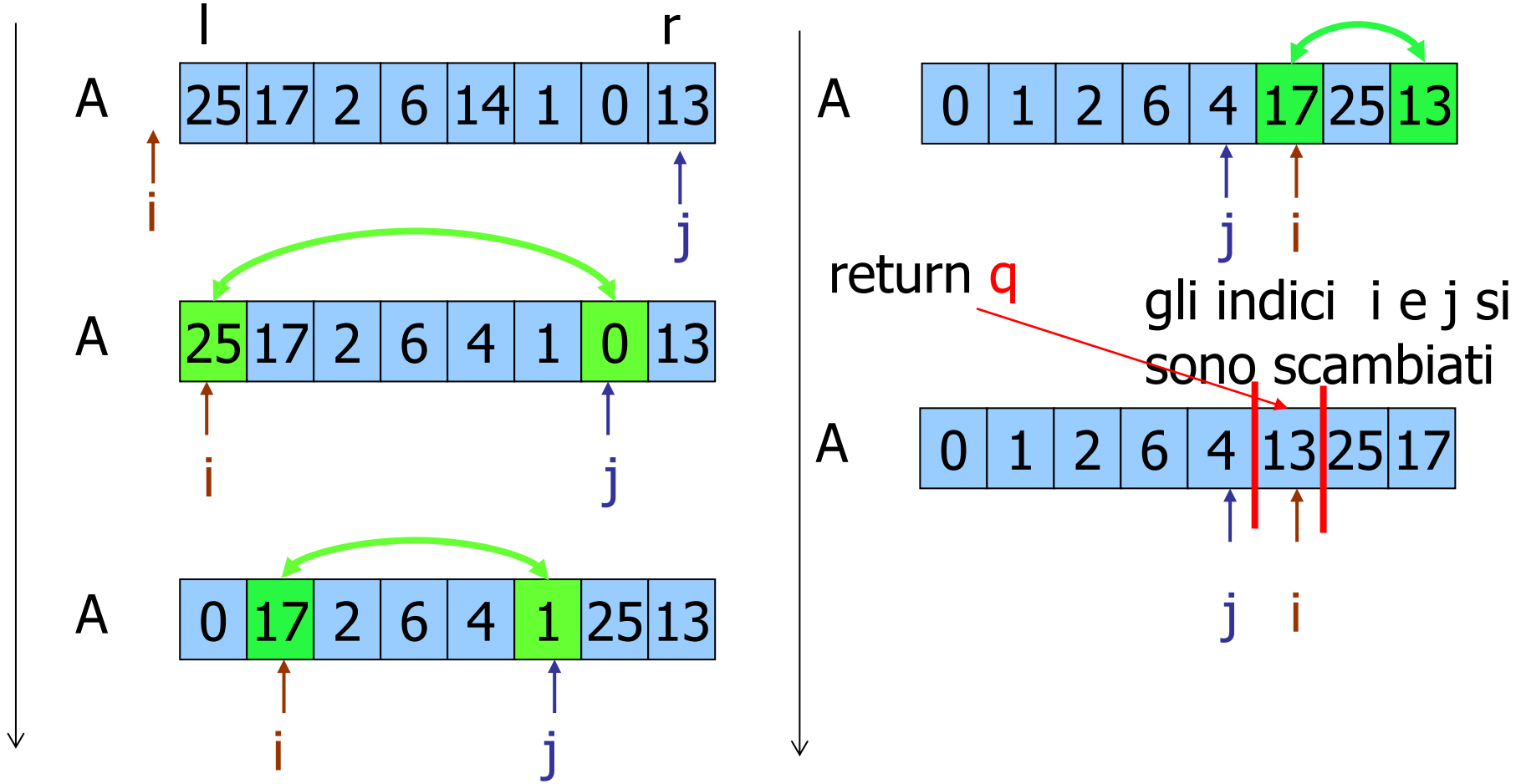
- Ricorsione
  - quicksort su sottovettore SX A[l..q-1]
  - quicksort su sottovettore DX A[q+1..r]
  - condizione di terminazione: se il vettore ha 1 elemento è ordinato
- Ricombinazione:
  - nulla.

Partition (à la Hoare)

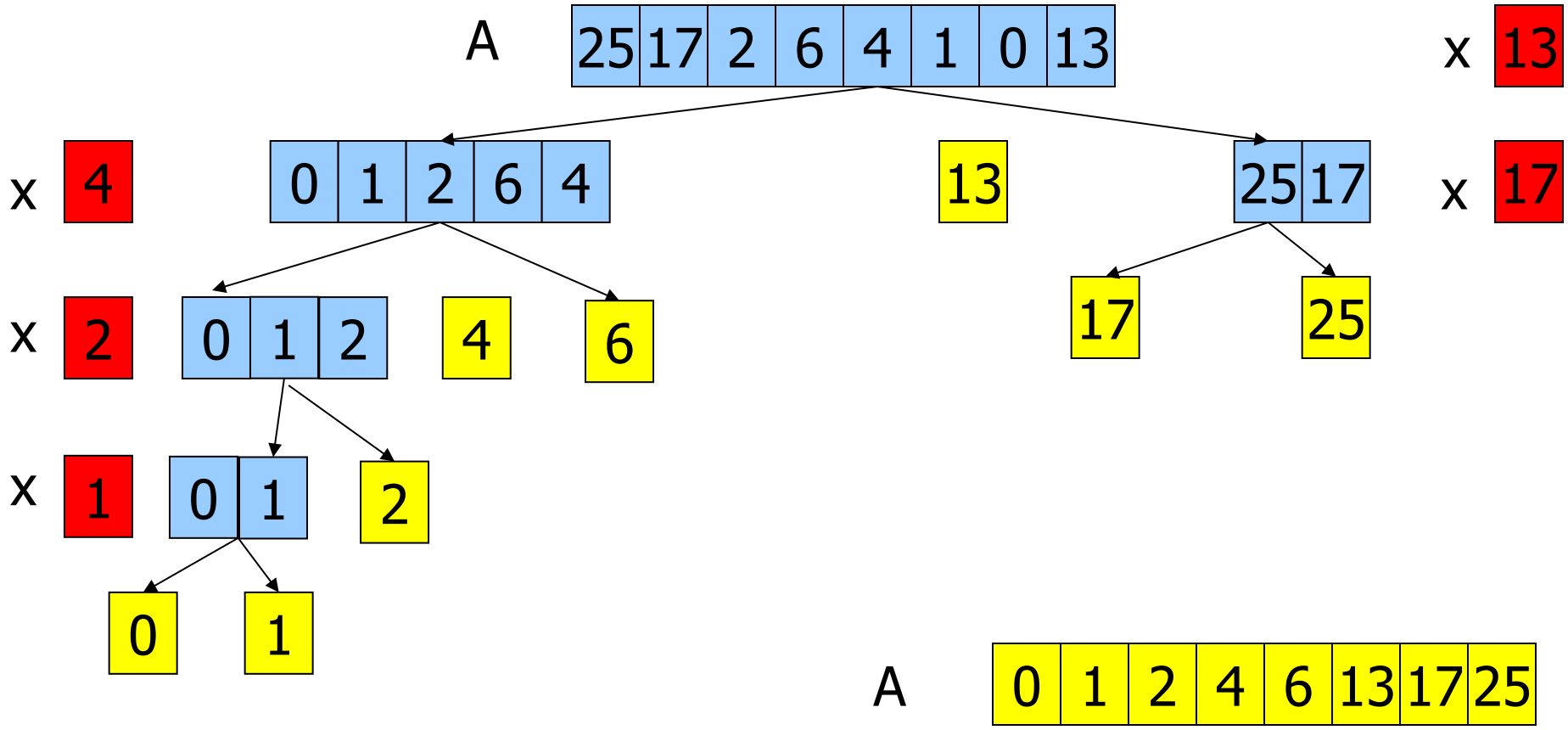
- Pivot x = A[r]
- individua A[i] e A[j] elementi "fuori posto"
  - ciclo discendente j fino a trovare un elemento minore del pivot x
  - ciclo ascendente su i fino a trovare un elemento maggiore del pivot x
- scambia A[i] e A[j]
- ripeti fintanto che i < j
- alla fine scambia A[i] e il pivot x
- ritorna q = i
- $T(n) = \Theta(n)$.

PS: esiste anche la partition à la Lomuto.

Esempio di partition



Esempio di quicksort



wrapper

estremi

chiamata alla funzione ricorsiva

RecursiveSort.c

```
void QuickSort(Item *A, int N) {
  int l=0, r=N-1;
  QuickSortR(A, l, r);
}
void quicksortR(Item *A, int l, int r ){
  int q;
  if (r <= l)
    return;
  q = partition(A, l, r);
  quicksortR(A, l, q-1);
  quicksortR(A, q+1, r);
  return;
}
void Swap(Item *v, int n1, int n2) {
   Item temp;
   temp  = v[n1];
   v[n1] = v[n2];
   v[n2] = temp;
   return;
}
```

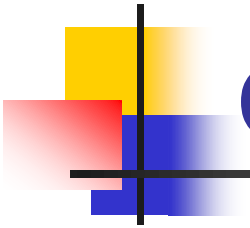
terminazione

divisione

chiamata ricorsiva

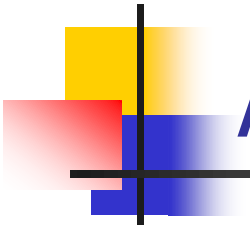
chiamata ricorsiva


```
int partition (Item *A, int l, int r ){
  int i = l-1, j = r;
  Item x = A[r];
  for ( ; ; ) {
    while(less(A[++i], x));
    while(greater(A[--j], x));
      if (j == l)
        break;
    if (i >= j)
      break;
    Swap(A, i, j);
  }
  Swap(A, i, r);
  return i;
}
```



Caratteristiche

- In loco
- Non stabile: la funzione partition può provocare uno scambio tra elementi «lontani», facendo sì che un'occorrenza di una chiave duplicata si sposti a SX di un'occorrenza precedente della stessa chiave «scavalcandola».



Analisi asintotica di caso peggiore

Efficienza legata al bilanciamento delle partizioni

A ogni passo partition ritorna:

- caso peggiore: un vettore da n-1 elementi e l'altro da 1
- caso migliore: due vettori da n/2 elementi
- caso medio: due vettori di dimensioni diverse.

Bilanciamento legato alla scelta del pivot.

Caso peggiore

- Caso peggiore: pivot = minimo o massimo (vettore già ordinato)
- Equazione alle ricorrenze:

$$T(n) = T(n-1) + n \qquad n>=2$$

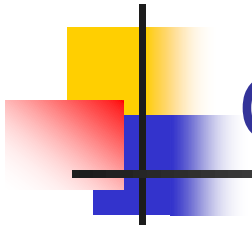
$$T(1) = 1 \qquad n = 1$$

- Risoluzione per sviluppo:

$$T(n) = n + (n-1) + (n-2) + \ldots. \; 3 + 2 + 1$$

$$= n/2 * (n+1)$$

$$T(n) = O(n^2)$$



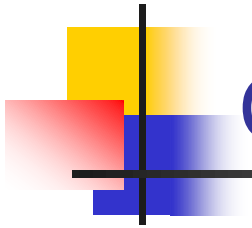
Caso migliore

- Equazione alle ricorrenze:

$$T(n) = 2T(n/2) + n \qquad n>=2$$

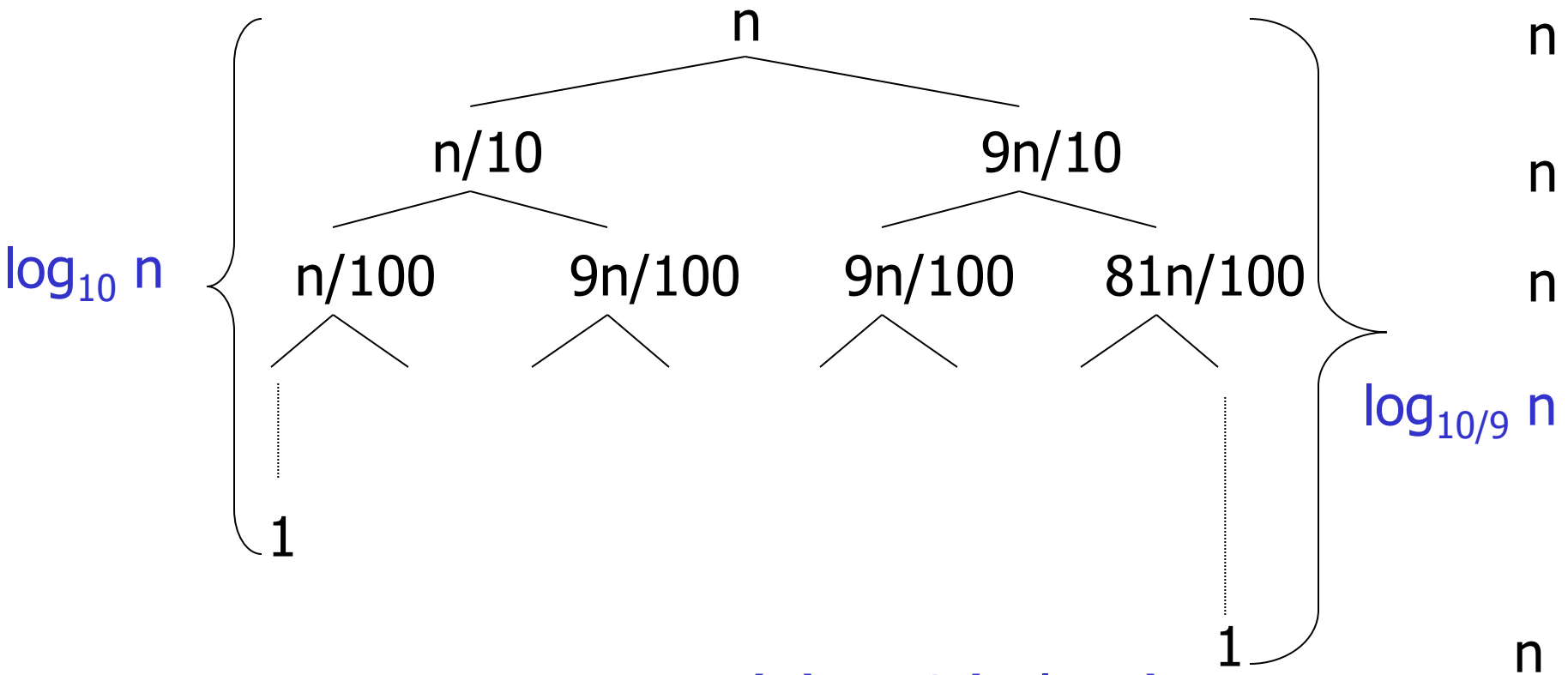
$$T(1) = 1 \qquad n = 1$$

$$T(n) = O(n \lg n)$$

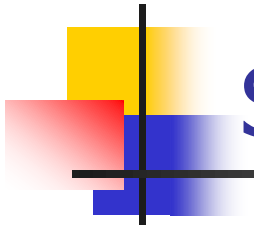


Caso medio

- Purché non si ricada nel caso peggiore, anche se il partizionamento è molto sbilanciato, caso medio = caso migliore
- Esempio: ad ogni passo si generano 2 partizioni, la prima con 9/10 n e la seconda con n/10 elementi.



$T(n) = O(n \lg n)$

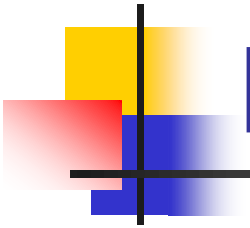


Scelta del pivot

- Elemento a caso: genera un numero casuale i con $l \leq i \leq r$, poi scambia A[r] e A[i], usando come pivot A[r]
- Elemento di mezzo: $x \leftarrow A[(l+r)/2]$
- Scegliere il valore medio tra min e max
- Scegliere la mediana tra 3 elementi presi a caso nel vettore
- …

Quadro riassuntivo

Algoritmo	In loco	Stabile	Caso peggiore
Bubble sort	sì	sì	$O(n^2)$
Selection sort	sì	sì	$O(n^2)$
Insertion sort	sì	sì	$O(n^2)$
Shellsort	sì	no	dipende
Mergesort	no	sì	O(nlog n)
Quicksort	sì	no	$O(n^2)$
Counting sort	no	sì	O(n)



Riferimenti

- Mergesort e Bottom-up mergesort
  - Sedgewick 8.3 e 8.5
  - Cormen 1.3
- Quicksort
  - Sedgewick 7.1 e 7.2
  - Cormen 8.1, 8.2