POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

Gli Ordinamenti Ricorsivi

Paolo Camurati

I dati da ordinare

- I dati da ordinare non sono necessariamente sempre e solo interi.
- Generalizzazione: i dati da ordinare appartengono ad un tipo Item definito come struct
  - uno dei campi = *chiave* di ordinamento
  - restanti campi = dati aggiuntivi
- funzioni di interfaccia a oggetti di tipo Item
  - lettura/scrittura
  - generazione di valori casuali
  - accesso alla chiave
- operatori relazionali su oggetti di tipo Item

`Item` definito come:

- ADT di I classe
- Quasi ADT

trattati più avanti

Tipologie:

1. semplice scalare e chiave coincidente
2. vettore dinamico di caratteri e chiave coincidente
3. scalare e vettore di caratteri sovradimensionato staticamente in una `struct`
4. scalare e vettore di caratteri allocato dinamicamente in una `struct`.

Cfr. *Puntatori e strutture dati dinamiche* cap. 6.2

Esempio: Quasi ADT – tipologia 1

```
...
#define maxKey 100

typedef int Item;


Item ITEMscan();
int  ITEMeq(Item A, Item B);
int  ITEMneq(Item A, Item B);
int  ITEMlt(Item A, Item B);
int  ITEMgt(Item A, Item B);
void ITEMshow(Item A);
Item ITEMrand();
```

definizione di un nuovo tipo `Item`

prototipi di funzioni su dati di tipo `Item`

```
...
int ITEMeq(Item A, Item B) {
  return (A == B);
}

int ITEMneq(Item A, Item B) {
  return (A != B);
}

int ITEMlt(Item A, Item B) {
  return (A < B);
}

int ITEMgt(Item A, Item B) {
  return (A > B);
}
...
```

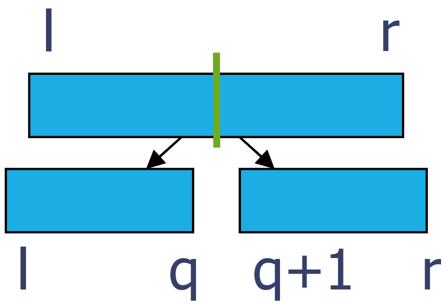
implementazione di funzioni su dati di tipo `Item`

```
Item ITEMscan(){
  Item A;
  printf("item = "); scanf("%d", &A);
  return A;
}

void ITEMshow(Item A) {
  printf("%6d \n", A);
}

Item ITEMrand() {
  Item A= maxKey*((float)rand()/RAND_MAX);
  return A;
}
```

Merge Sort (von Neumann, 1945)

Divisione:

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



Ricorsione

- condizione di terminazione: con 1 (l=r) o 0 (l>r) elementi il vettore è ordinato
- merge sort su sottovettore SX
- merge sort su sottovettore DX

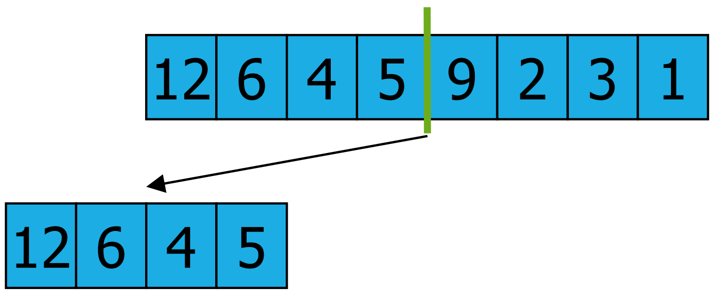
Ricombinazione:

- fondi i due sottovettori ordinati in un vettore ordinato.

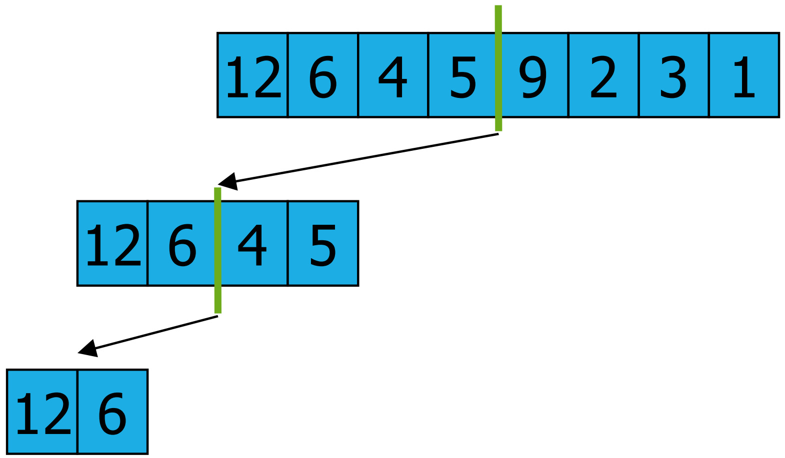
Esempio

12	6	4	5	9	2	3	1

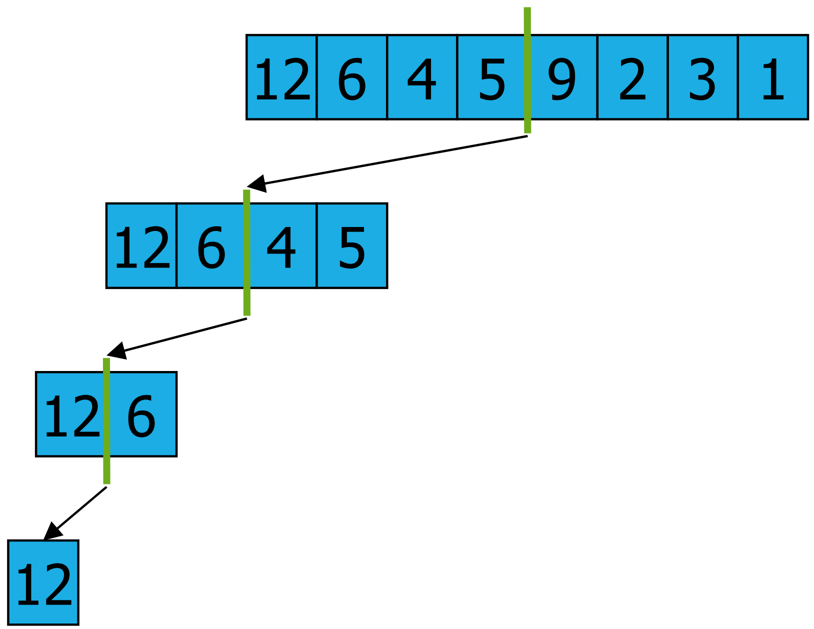
Divisione ricorsiva:



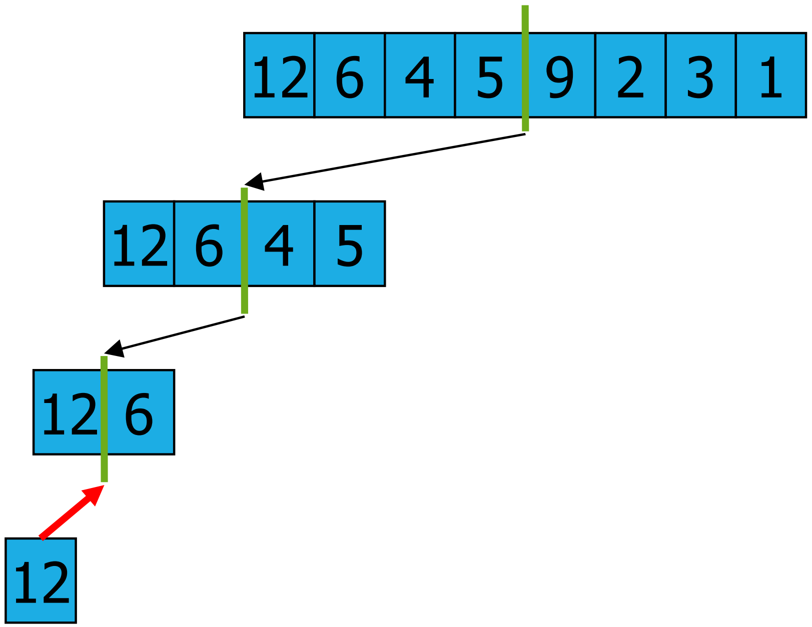
Divisione ricorsiva:



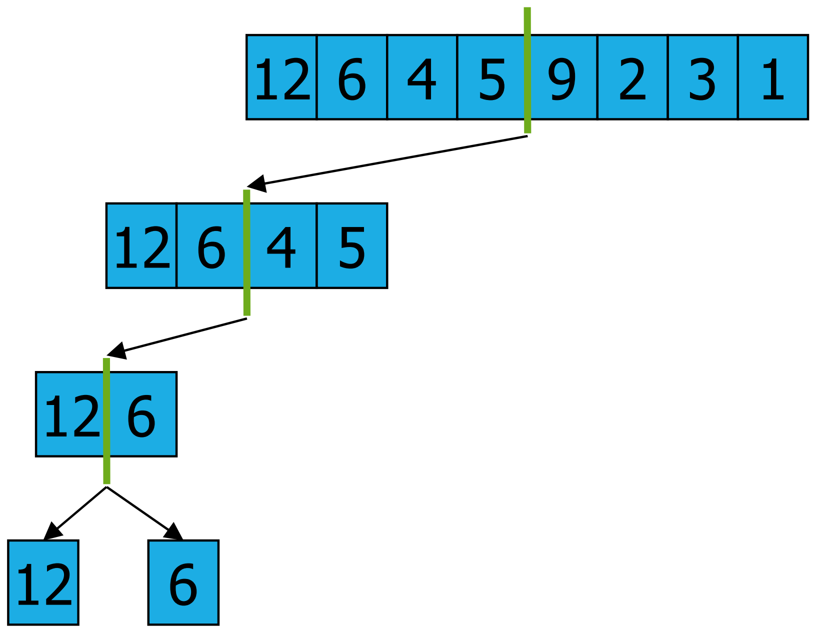
Divisione ricorsiva:



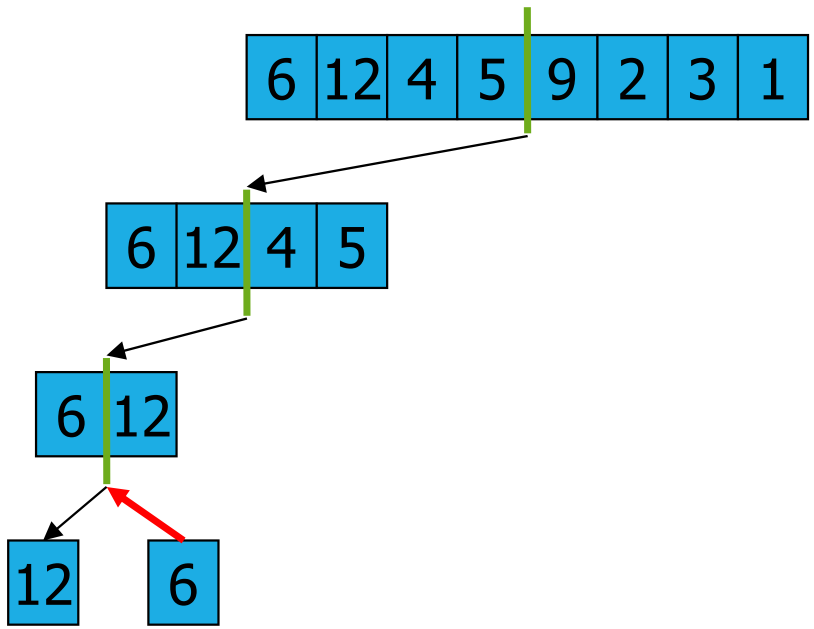
Ricombinazione:



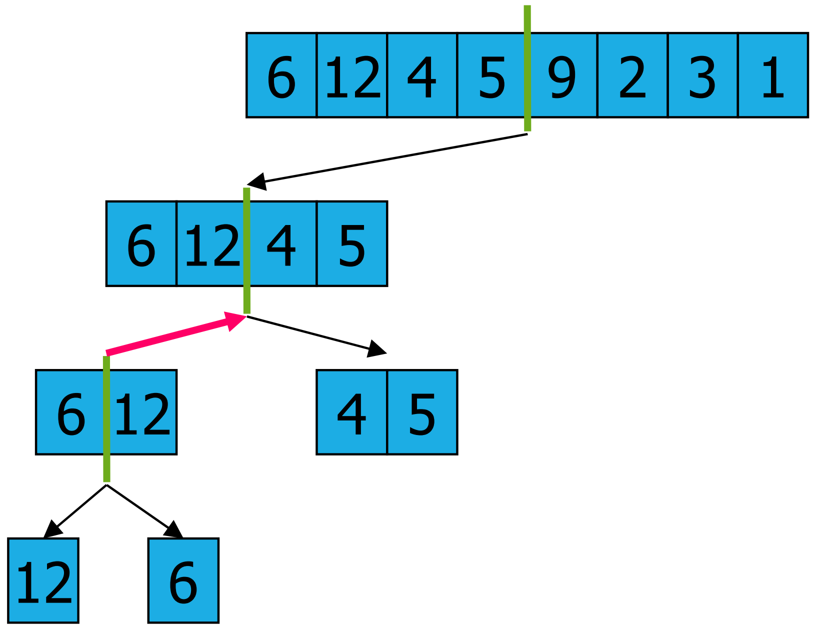
Divisione ricorsiva:



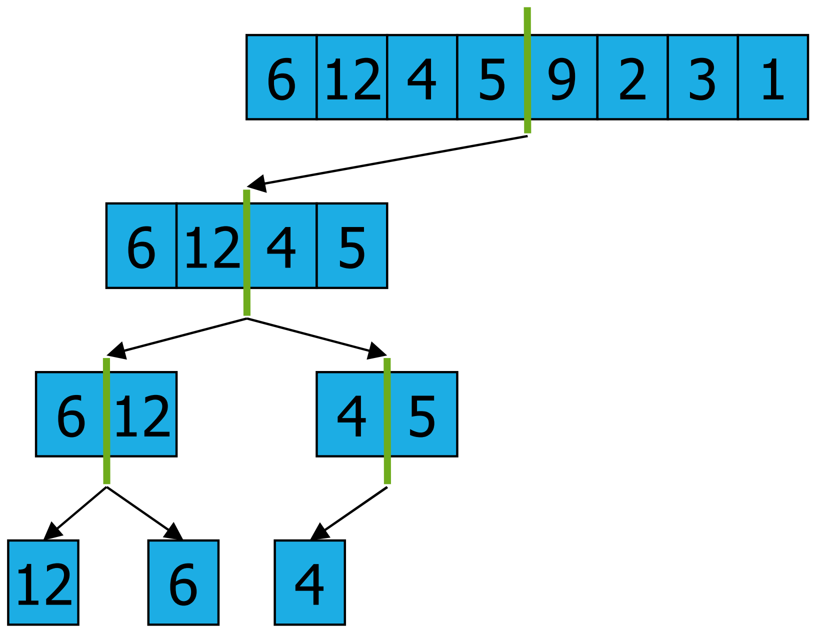
Ricombinazione:



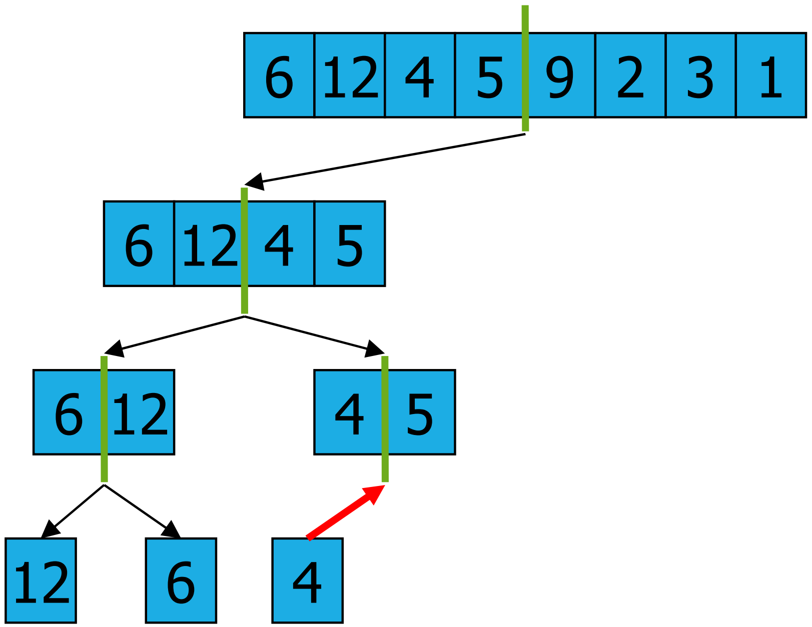
Divisione ricorsiva:



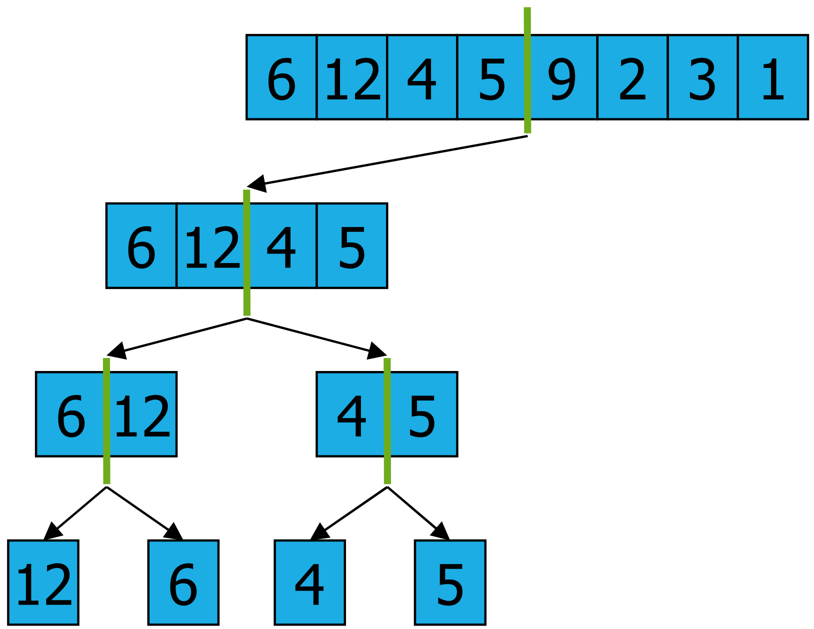
Divisione ricorsiva:



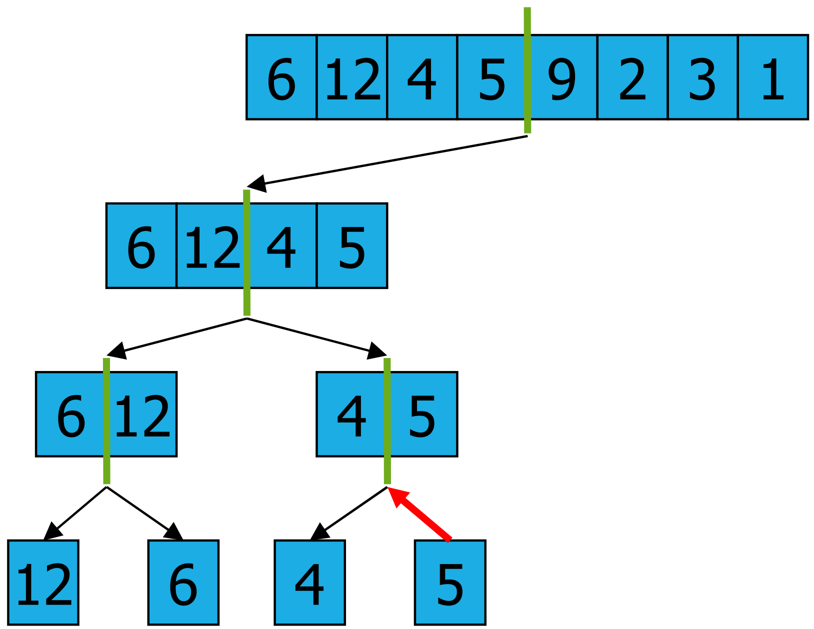
Ricombinazione:



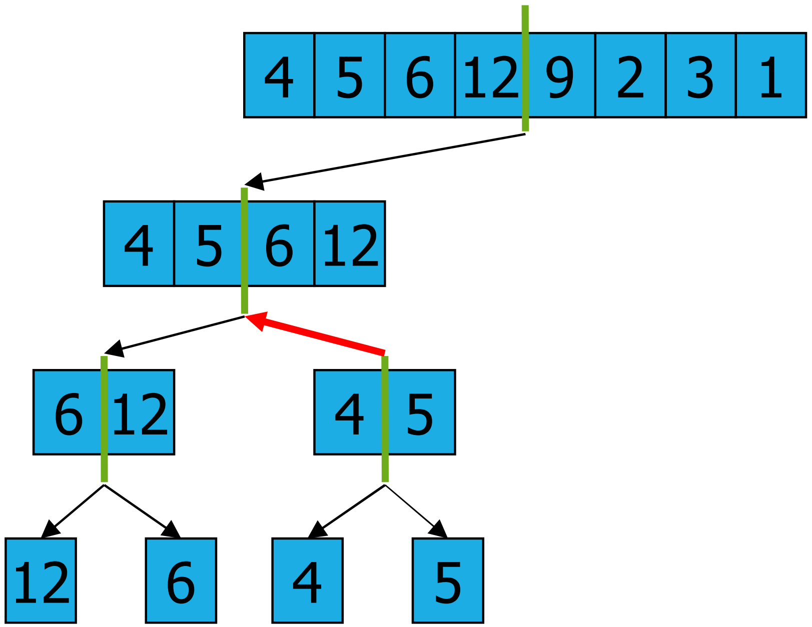
Divisione ricorsiva:



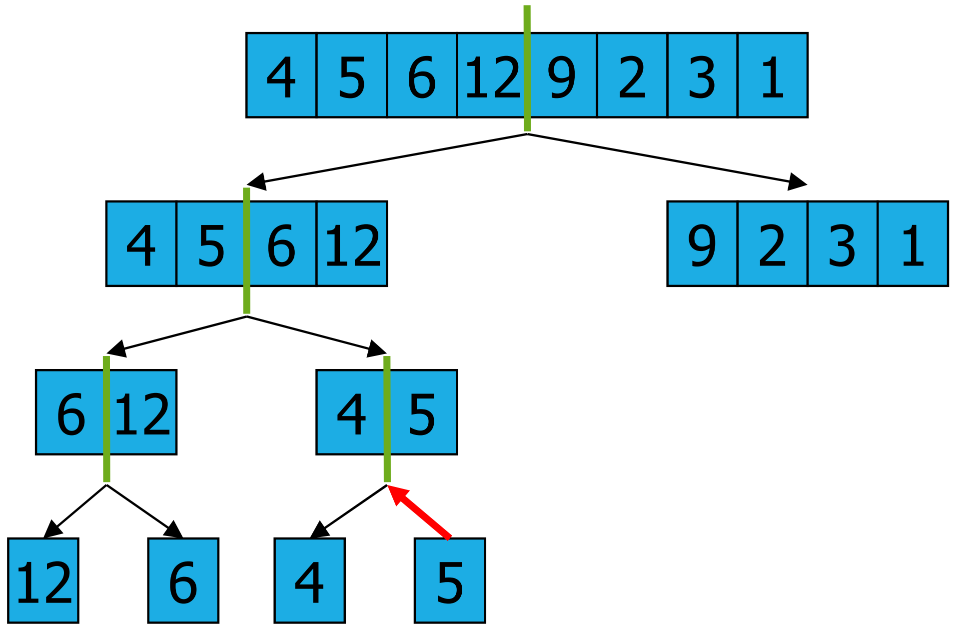
Ricombinazione:



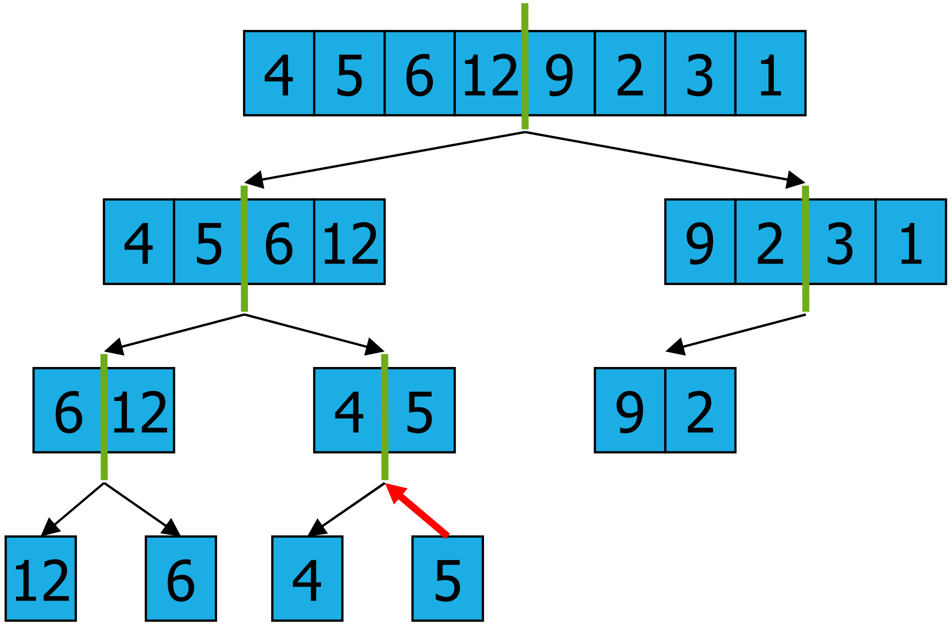
Ricombinazione:



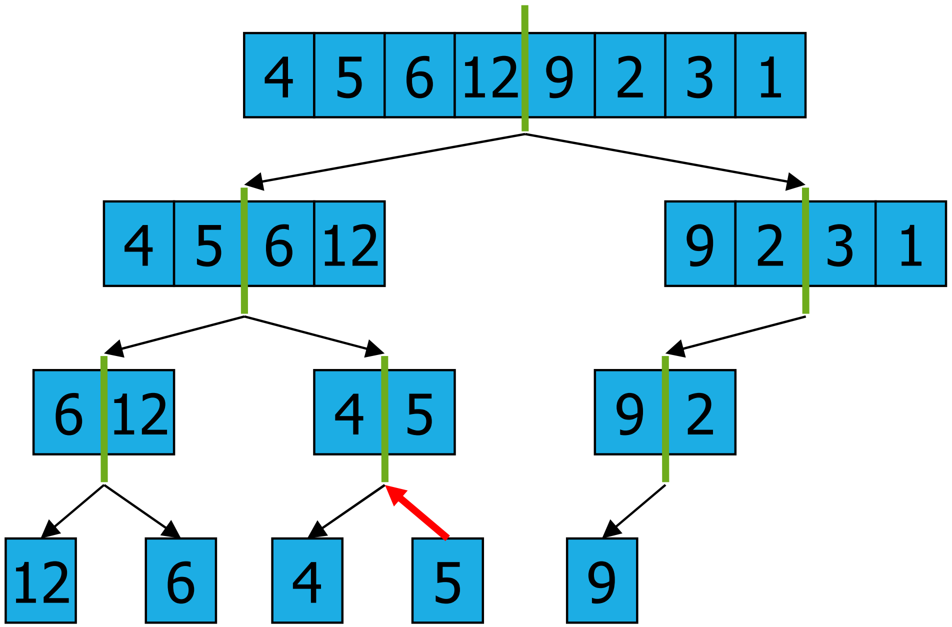
Discesa ricorsiva:



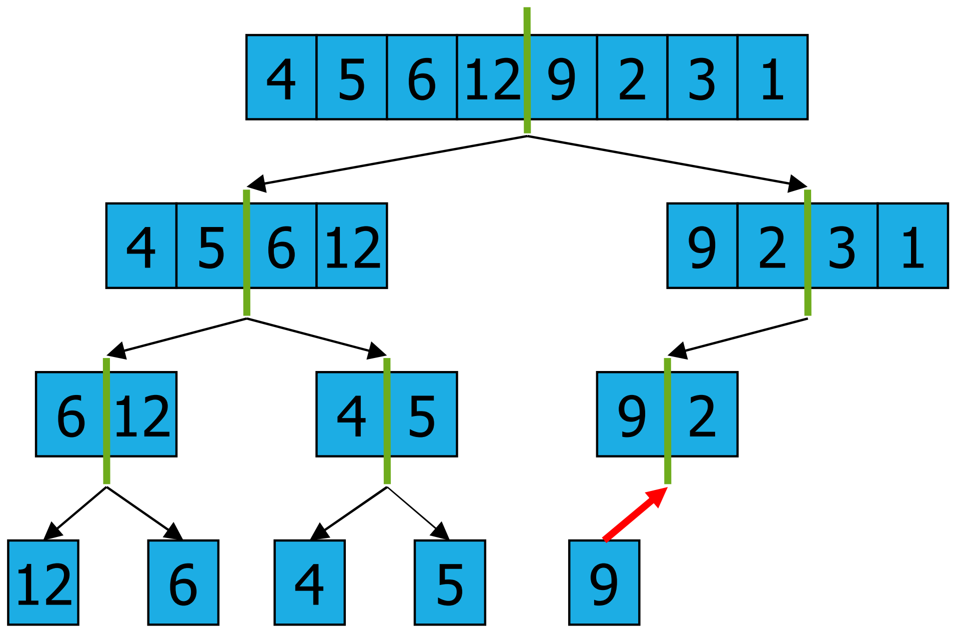
Discesa ricorsiva:



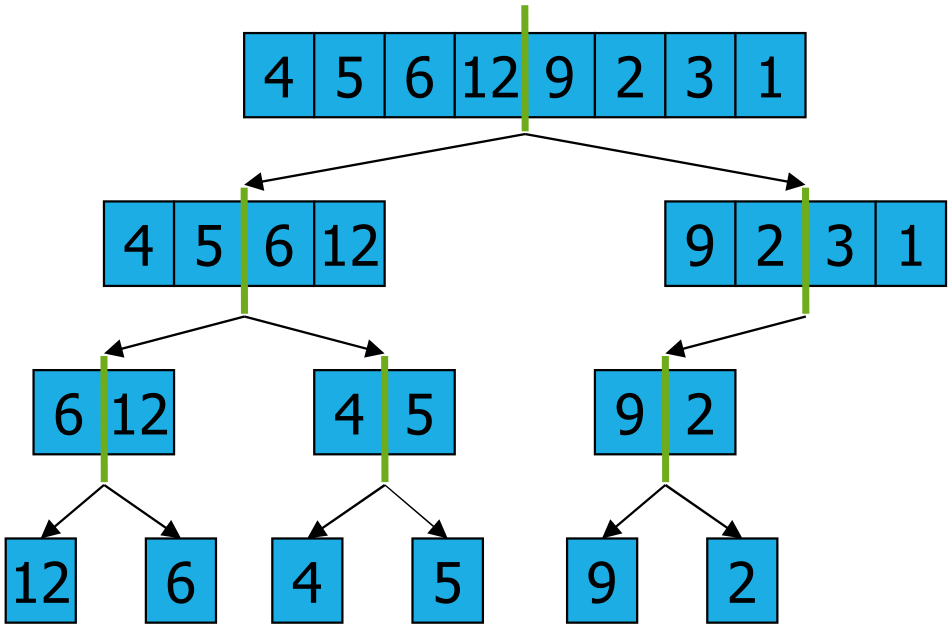
Discesa ricorsiva:



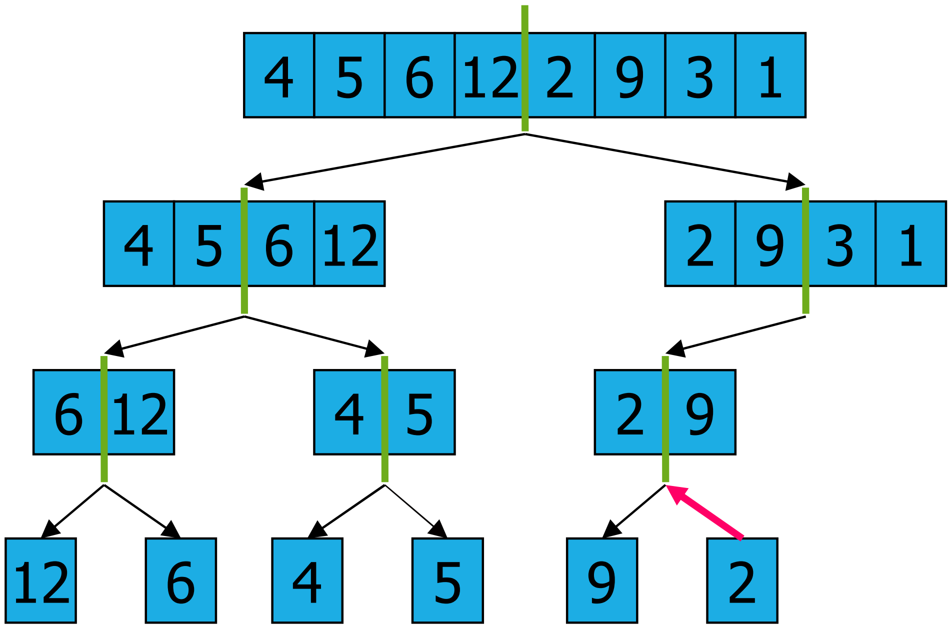
Ricombinazione:



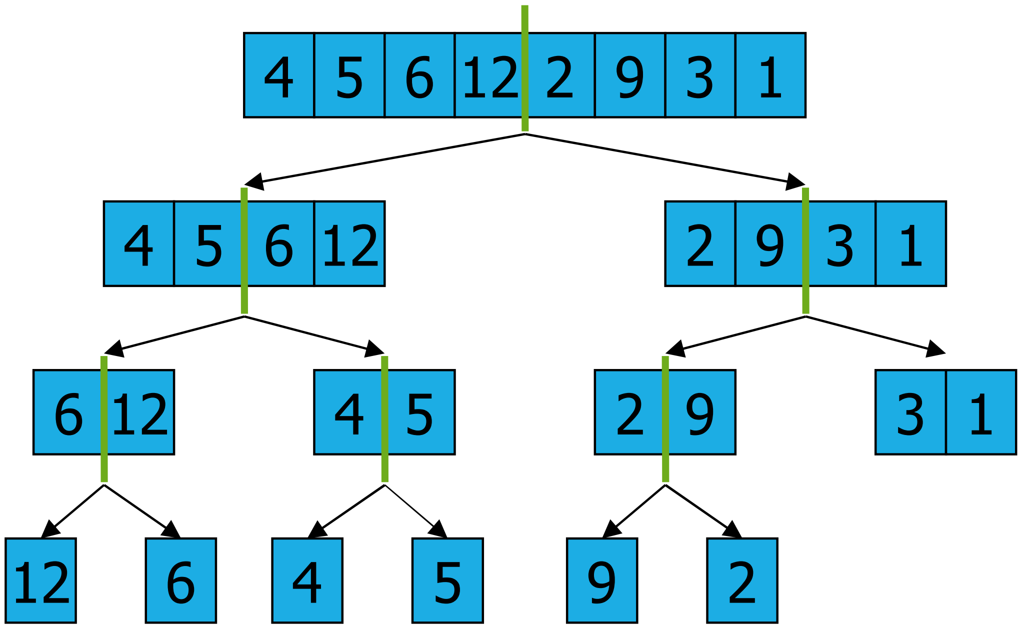
Discesa ricorsiva:



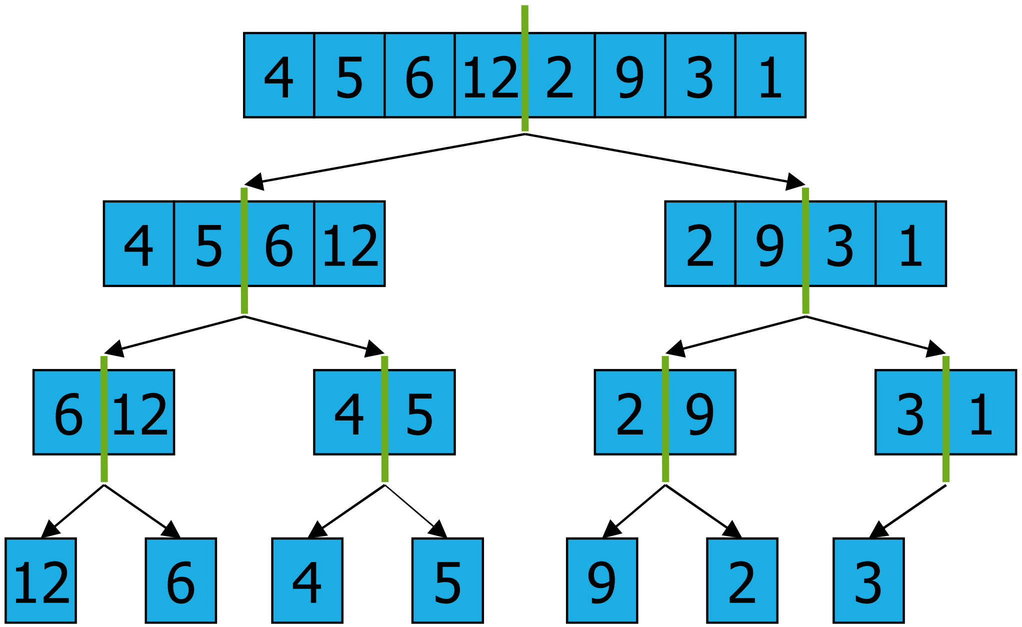
Ricombinazione:



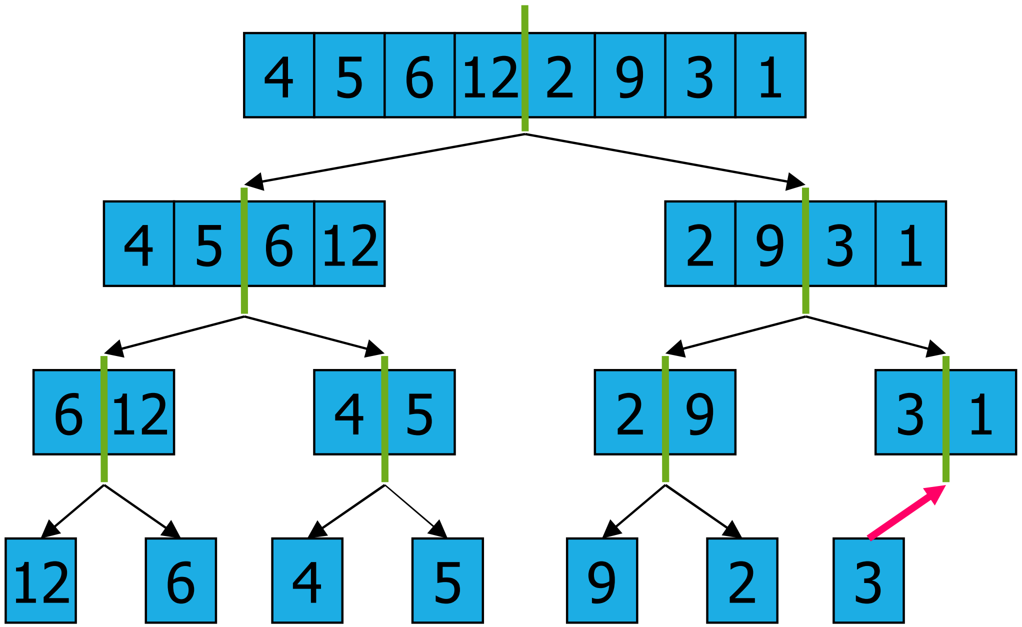
Discesa ricorsiva:



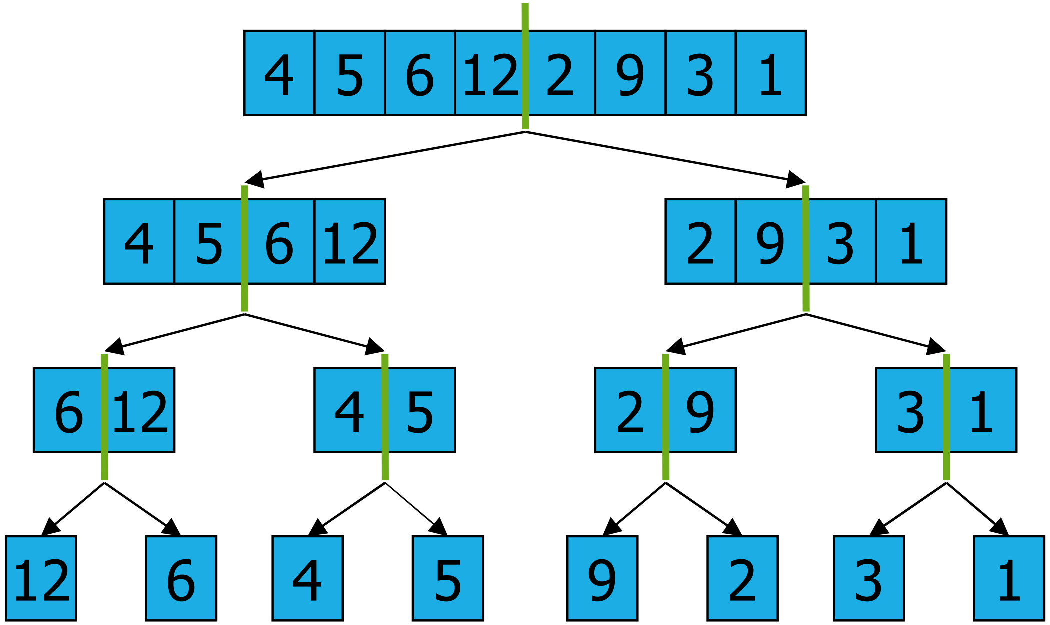
Discesa ricorsiva:



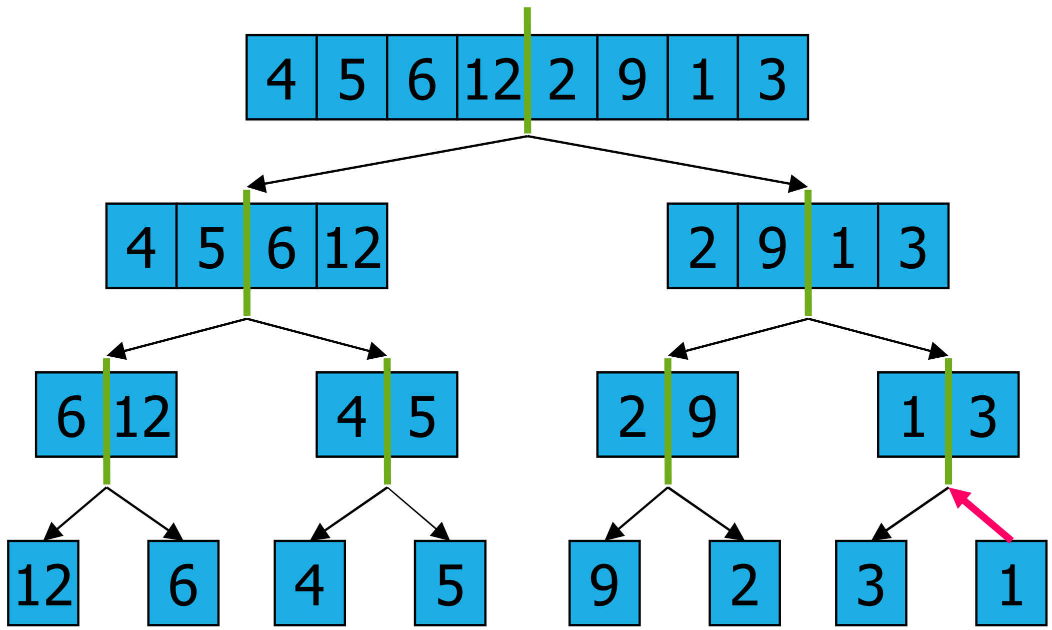
Ricombinazione:



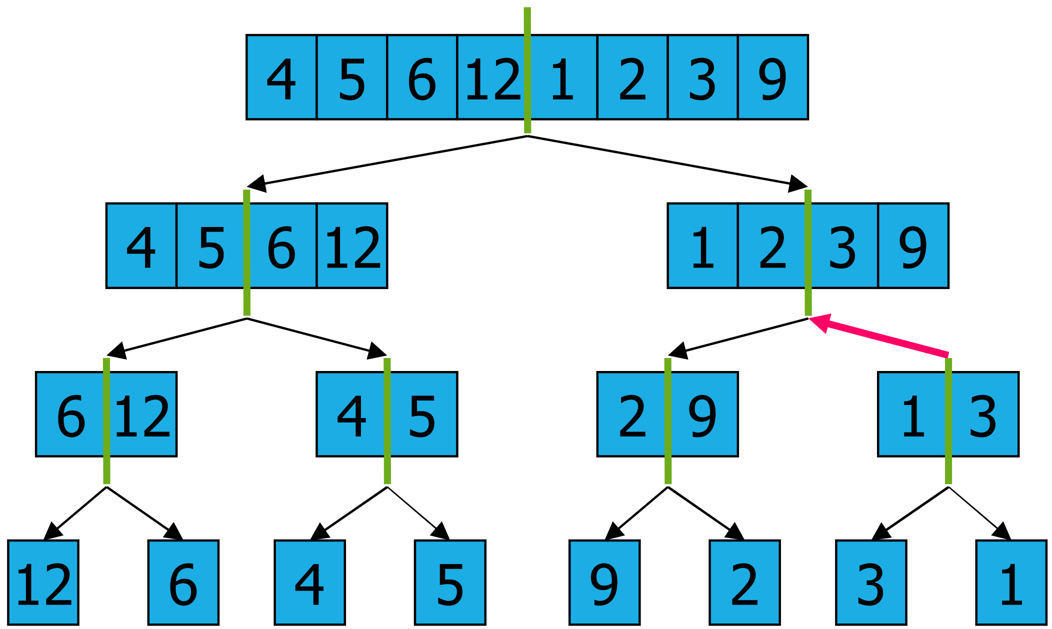
Discesa ricorsiva:



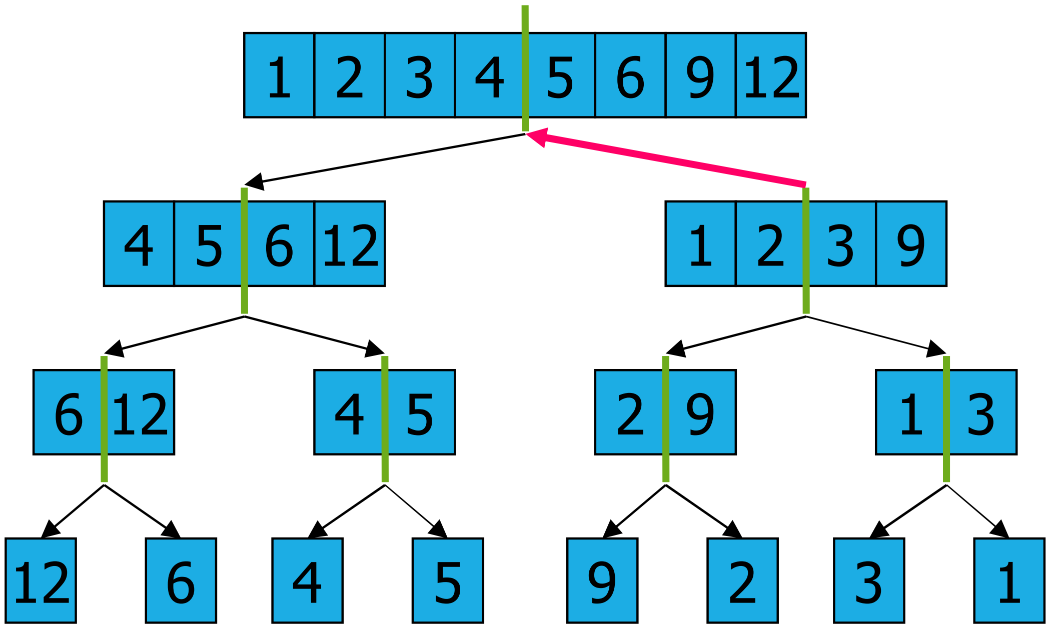
Ricombinazione:



Ricombinazione:



Ricombinazione:



vettore ausiliario

wrapper

```c
void MergeSort(Item A[], Item B[], int N) {
  int l=0, r=N-1;
  MergeSortR(A, B, l, r);
}
```

estremi

chiamata alla funzione ricorsiva

RecursiveSort.c

```
void MergeSortR(Item A[], Item B[], int l, int r){
  int q;
  if (l >= r)
    return;
  q = (l + r)/2;
  MergeSortR(A, B, l, q);
  MergeSortR(A, B, q+1, r);
  Merge(A, B, l, q, r);
}
```

terminazione

divisione

chiamata ricorsiva

chiamata ricorsiva

ricombinazione

2-way Merge

- ipotesi: la dimensione del vettore A è una potenza di 2 $N = 2^p$
- fusione di 2 (2-way) sottovettori di A ordinati di ugual dimensione per ottenere un vettore ordinato di dimensione doppia
- generalizzabile a k vettori (k-way Merge)
- indice q per dividere sottovettori di A a metà in 2 sottovettori sinistro e destro
- sottovettore sinistro con indice $l \leq i \leq q$
- sottovettore destro con indice $q+1 \leq j \leq r$
- vettore ausiliario B di dimensione N con indice $l \leq k \leq r$ per memorizzare i risultati delle fusioni passato come parametro

Approccio:

- scorrere i sottovettori sinistro e destro mediante gli indici i e j e il vettore B mediante l'indice k
- se è esaurito il sottovettore sinistro (i>q), ricopiare gli elementi rimanenti del sottovettore destro in B
- altrimenti se è finito il sottovettore destro (j>r), ricopiare gli elementi rimanenti del sottovettore sinistro in B
- altrimenti confrontare l'elemento corrente A[i] del sottovettore sinistro con quello del sottovettore destro A[j]
  - se A[i] ≤ A[j], ricopiare A[i] in B e avanzare i, j resta invariato
  - altrimenti ricopiare A[j] in B e avanzare j, i resta invariato

I confronti avvengono mediante operatori relazionali su Item.

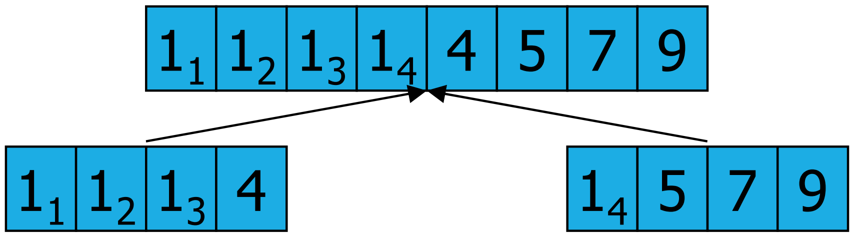
```
void Merge(Item A[], Item B[], int l, int q, int r) {
  int i, j, k;
  i = l;
  j = q+1;
  for (k = l; k <= r; k++)
    if (i > q)
      B[k] = A[j++];
    else if (j > r)
      B[k] = A[i++];
    else if (ITEMlt(A[i], A[j]) || ITEMeq(A[i], A[j]))
      B[k] = A[i++];
    else
      B[k] = A[j++];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

esaurito sottovettore SX

esaurito sottovettore DX

Caratteristiche

- Non in loco (usa il vettore ausiliario B la cui dimensione è funzione di N)
- Stabile: in quanto la funzione merge prende dal sottovettore SX in caso di chiavi uguali:



Analisi di complessità

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:

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \qquad & n>1 \\
T(1) &= 1 \qquad & n=1
\end{aligned}
$$

- soluzione per sviluppo (unfolding)

$T(n/2) = 2T(n/4) + n/2$

$T(n/4) = 2T(n/8) + n/4$

etc.

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

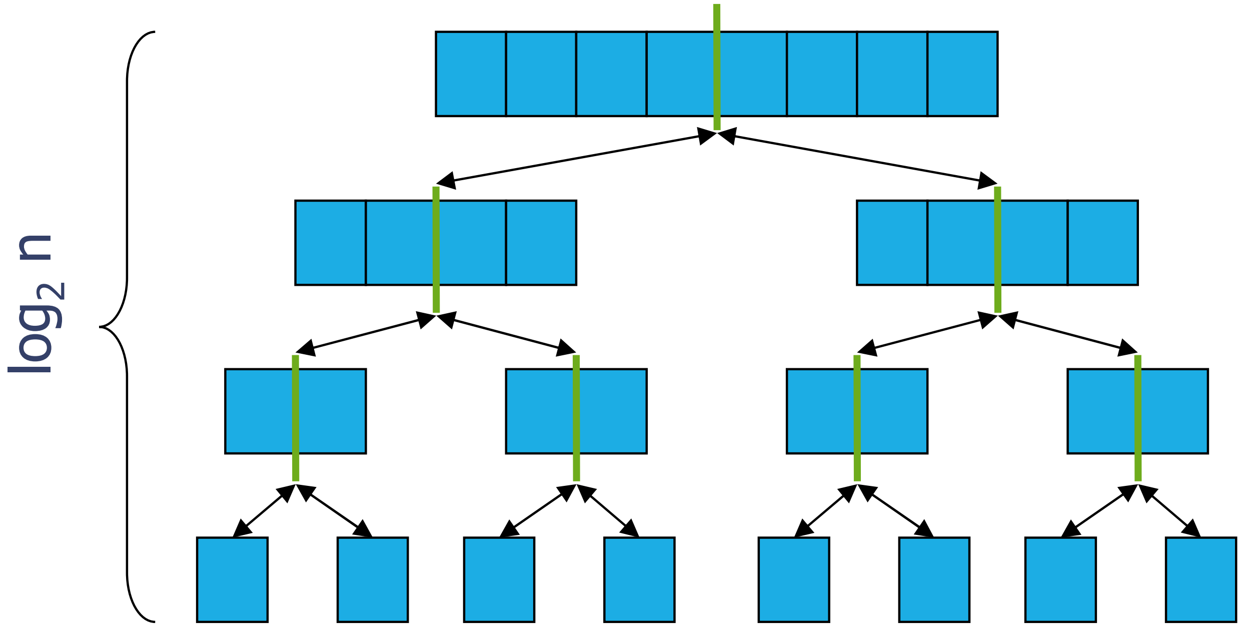
$T(n) = n + 2*(n/2) + 2^2 *(n/4) + 2^3 *T(n/8)$

$= \Sigma_{0 \leq i \leq \log 2n} 2^i / 2^i * n = n * \Sigma_{0 \leq i \leq \log 2n} 1$

$= n*(1 + \log_2 n) \;\; n \log_2 n + n$

$T(n) = O(n \log n)$. Altri metodi di risoluzione dell'equazione alle ricorrenze portano a $T(n) = \Theta(n \log n)$.

Intuitivamente:



Livelli di ricorsione: $\log_2 n$

Operazioni per livello: n

Operazioni totali: $n \log_2 n$

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:

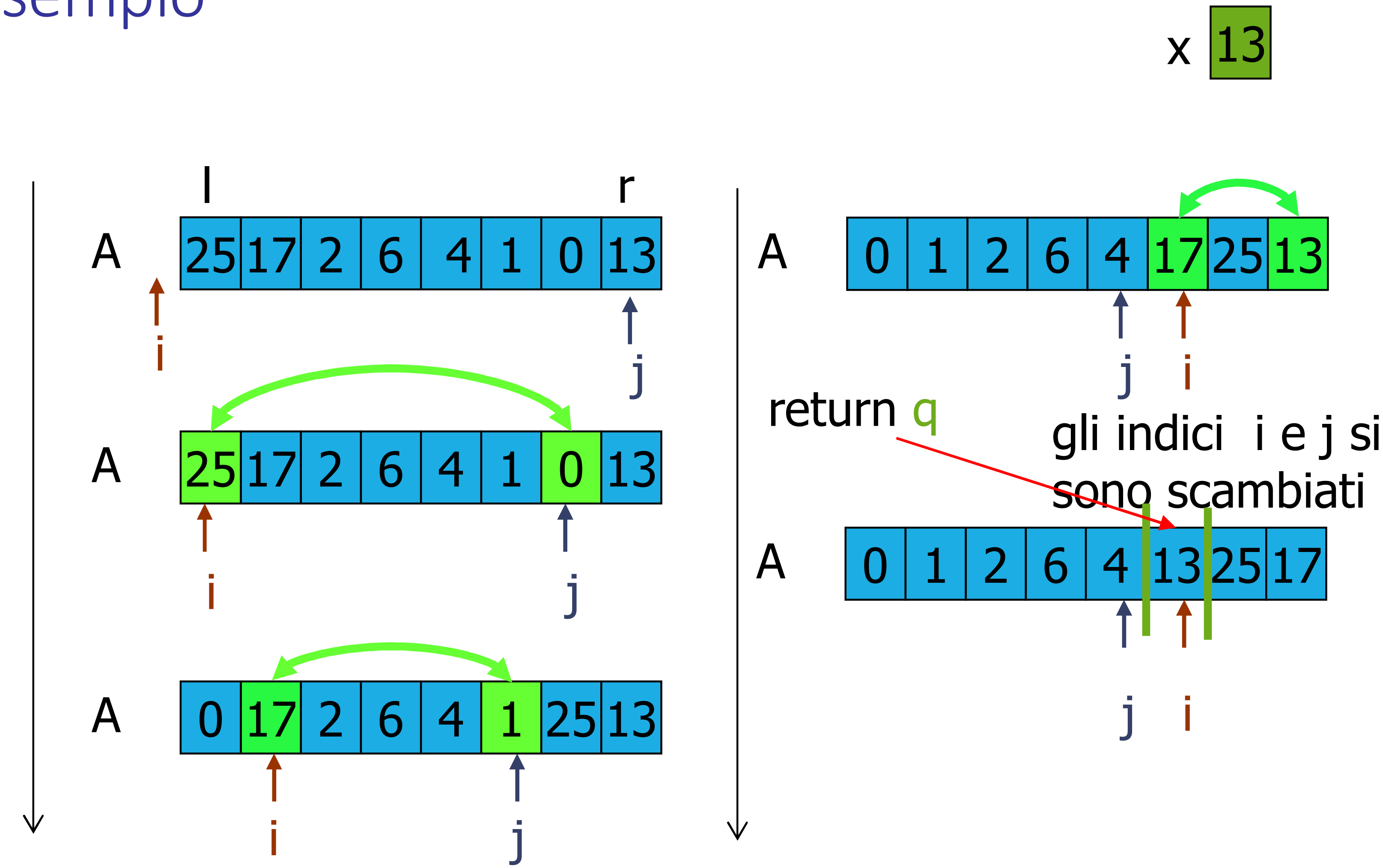
- nulla.

Partition (à la Hoare)

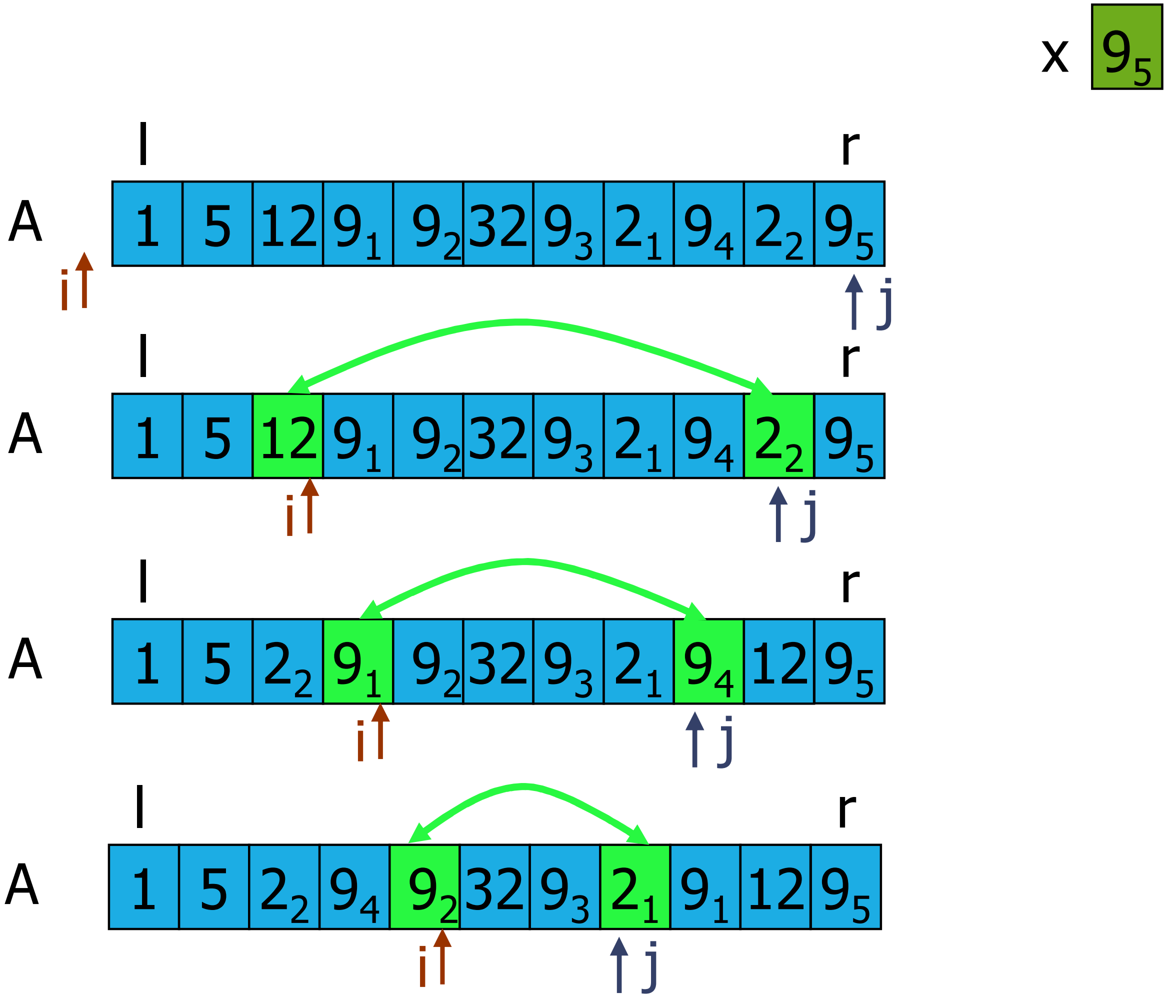
- Pivot $x = A[r]$
- i per scandire da SX a DX, j da DX a SX
- ripeti fintanto che $i < j$
  - individua $A[i]$ e $A[j]$ elementi "fuori posto"
    - ciclo ascendente su i fino a trovare un elemento maggiore del pivot x
    - ciclo discendente j fino a trovare un elemento minore del pivot x
  - scambia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

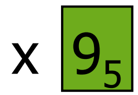
PS: esiste anche la partition à la Lomuto.

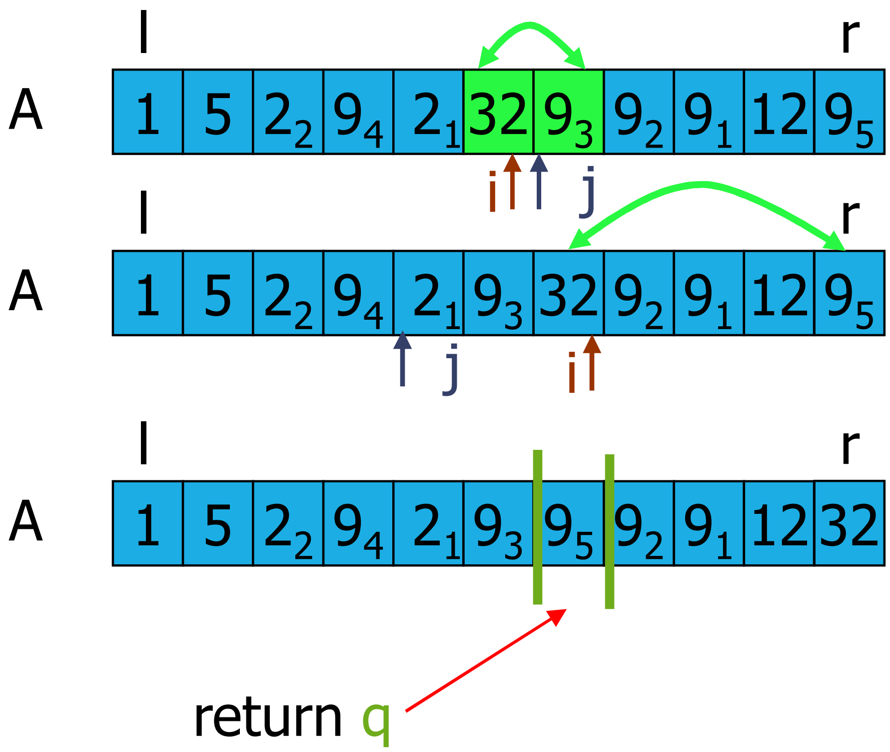
Esempio


x
13
l
r
A
25 17 2 6 4 1 0 13
i
j
A
25 17 2 6 4 1 0 13
i
j
A
0 17 2 6 4 1 25 13
i
j
A
0 1 2 6 4 17 25 13
j
i
return q
gli indici i e j si
sono scambiati
A
0 1 2 6 4 13 25 17
j
i

Si osservi che, se il pivot compare in più istanze, è irrilevante che esse si trovino alla fine della partition nel sottovettore sinistro o in quello destro, l’importante è che l’istanza scelta come pivot finisca nella posizione finale.

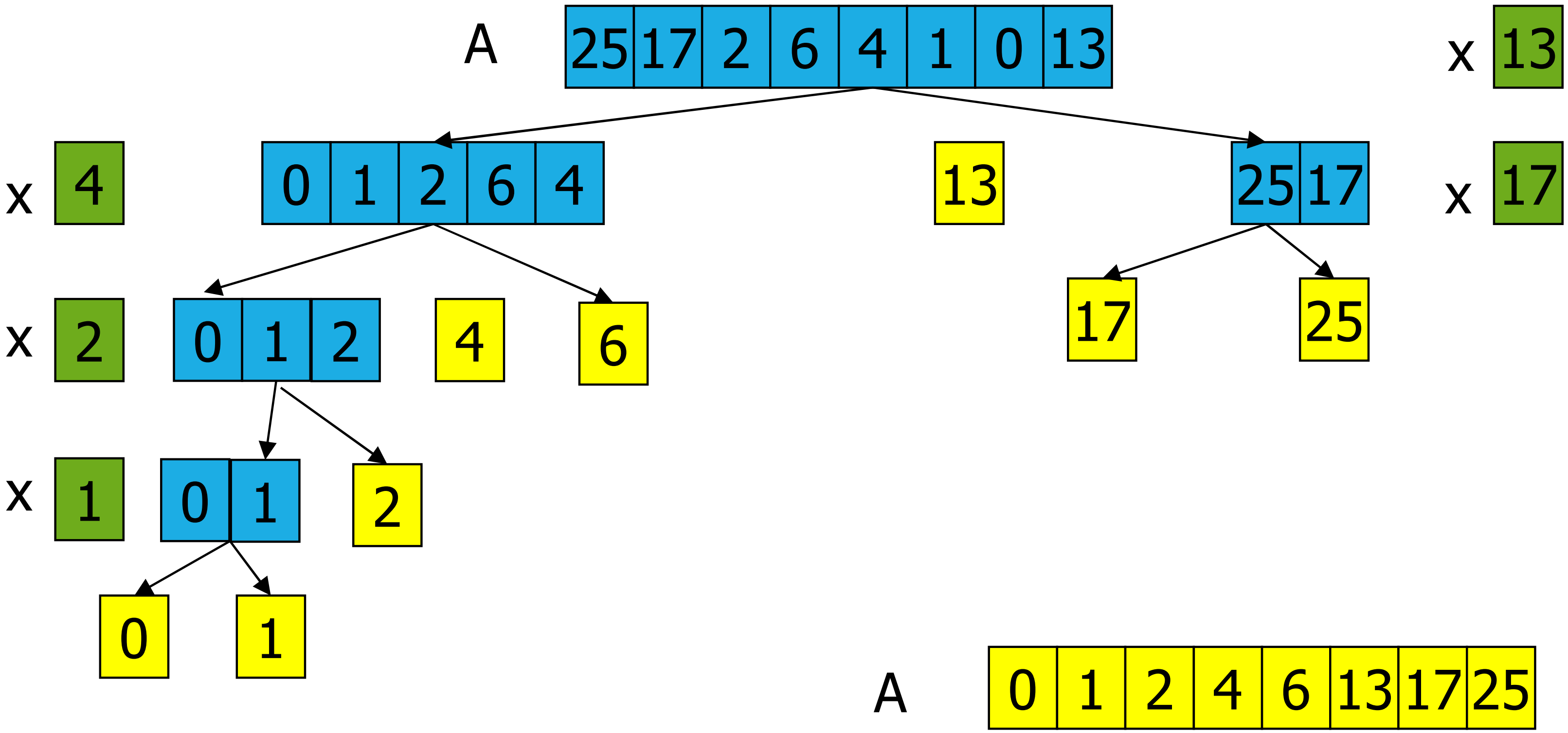

x
9_5
l
r
A
1 5 12 9_1 9_2 32 9_3 2_1 9_4 2_2 9_5
i
j
A
1 5 12 9_1 9_2 32 9_3 2_1 9_4 2_2 9_5
A
1 5 2_2 9_1 9_2 32 9_3 2_1 9_4 12 9_5
A
1 5 2_2 9_4 9_2 32 9_3 2_1 9_1 12 9_5





gli indici i e j si
sono scambiati

Esempio


A
25 17 2 6 4 1 0 13
x 13
x 4
0 1 2 6 4
13
25 17
x 17
x 2
0 1 2
4
6
17
25
x 1
0 1
2
0
1
A
0 1 2 4 6 13 17 25

```
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 (l >= r)
    return;
  q = partition(A, l, r);
  quicksortR(A, l, q-1);
  quicksortR(A, q+1, r);
  return;
}
```

wrapper

estremi

chiamata alla funzione ricorsiva

terminazione

divisione

chiamata ricorsiva

chiamata ricorsiva

RecursiveSort.c

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

```c
void Swap(Item *v, int n1,
          int n2) {
  Item temp;
  temp  = v[n1];
  v[n1] = v[n2];
  v[n2] = temp;
  return;
}
```

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 di complessità

Efficienza legata al bilanciamento delle partizioni

A ogni passo partition ritorna:

- caso peggiore: un vettore da n-1 elementi e l'altro da 1 elemento
- 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 in vettore già ordinato in ordine opposto a quello desiderato

Equazione alle ricorrenze:

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

$$T(1) = 1$$

Risoluzione per sviluppo:

$$T(n) = n + (n-1) + (n-2) + \ldots.\ 3 + 2 + 1 = \Sigma_{1 \leq i \leq n}\, i = n * (n+1)/2$$

Quindi $T(n) = O(n^2)$. Altri metodi di risoluzione dell'equazione alle ricorrenze portano a $T(n) = \Theta(n^2)$.

Caso migliore

Equazione alle ricorrenze:

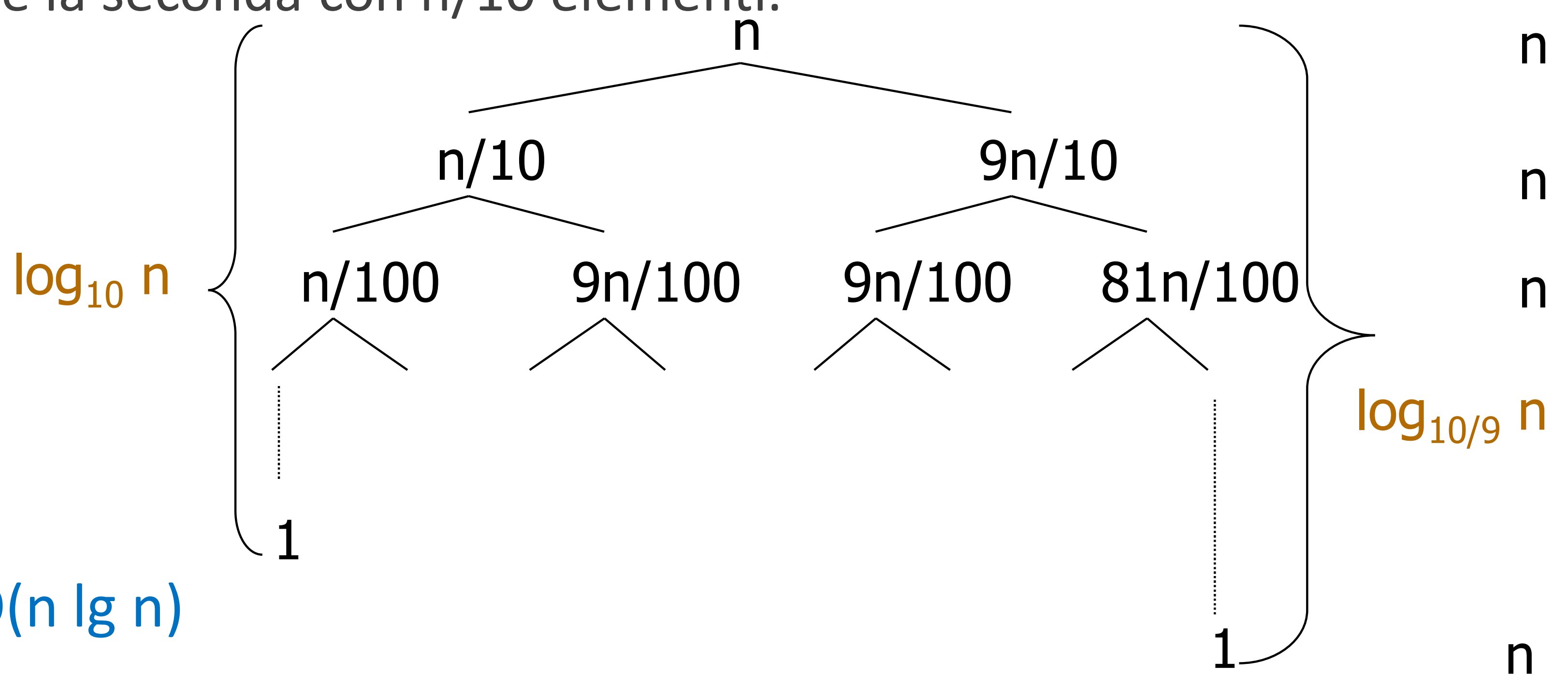
$T(n) = 2T(n/2) + n \qquad n \geq 2$

$T(1) = 1$

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

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 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
- …

Se lo scopo è semplicemente di rendere molto improbabile il caso peggiore, basta generare un numero casuale i con $l \leq i \leq r$, poi scambiare A[r] e A[i] e infine usare come pivot A[r].

Quadro riassuntivo

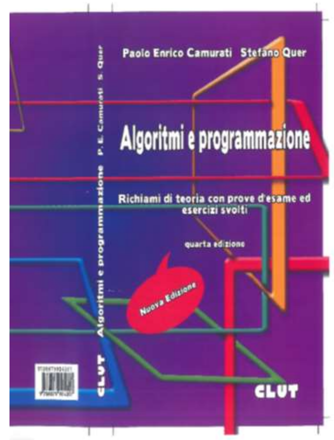
ALGORITMO	IN LOCO	STABILE	COMPLESSITÀ
Bubble Sort	Sì	Sì	$O(n^2)$
Selection Sort	Sì	No	$O(n^2)$
Insertion Sort	Sì	Sì	$O(n^2)$
Shell Sort	Sì	No	dipende
Merge Sort	No	Sì	$O(n \log n)$
Quick Sort	Sì	No	$O(n^2)$
Counting Sort	No	Sì	$O(n)$
Radix Sort	No	Sì	$O(n)$

Riferimenti

- Mergesort
  - Sedgewick 8.3 e 8.5
  - Cormen 2.3
- Quicksort
  - Sedgewick 7.1 e 7.2
  - Cormen 7.1, 7.2

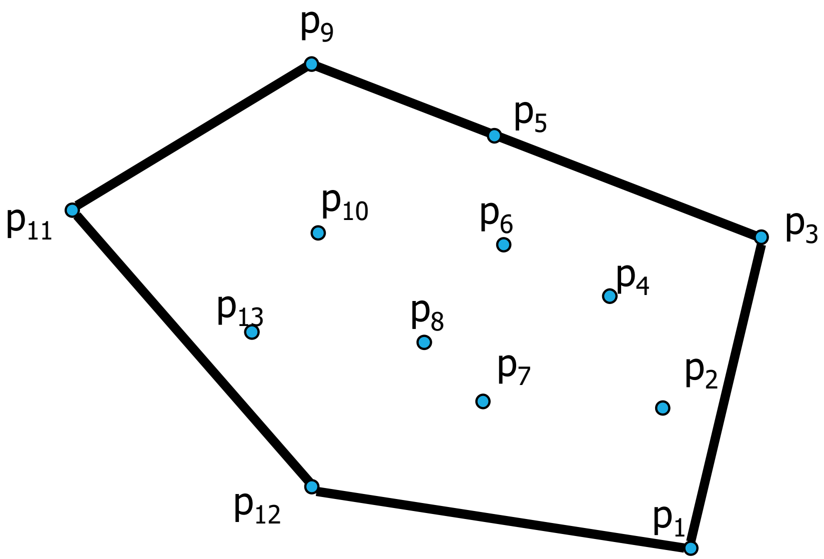
Esercizi di teoria

- 2. Ordinamenti ricorsivi
  - 2.1 Merge Sort
  - 2.2 Quick Sort



Un'applicazione: l'inviluppo convesso

Dato un insieme di punti Q, l'**inviluppo convesso** è il poligono convesso P di area minima per cui ogni punto di Q è interno a P o al più sul perimetro di P:



Soluzione brute-force:

- dato l'insieme dei punti, costruire l'insieme dei suoi sottoinsiemi (insieme delle parti)
- per ciascun sottoinsieme verificare che si tratti di un poligono convesso e se sì calcolarne l'area
- tenere traccia dell'area minima
- complessità esponenziale

Soluzione efficiente: Graham Scan (1972)

- ordinare i punti con Mergesort
- scansione lineare dei punti
- complessità T(N) = O(N lg N)

Il Graham Scan è trattato nei lucidi di approfondimento disponibili sul Portale della Didattica.