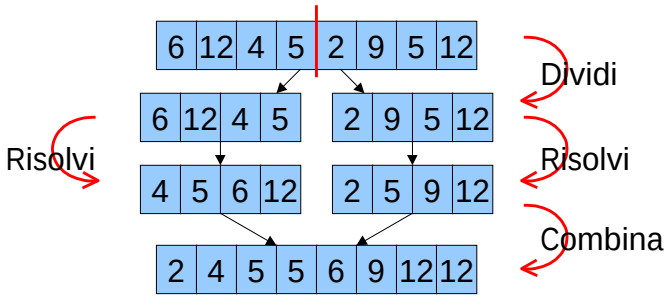
Merge Sort

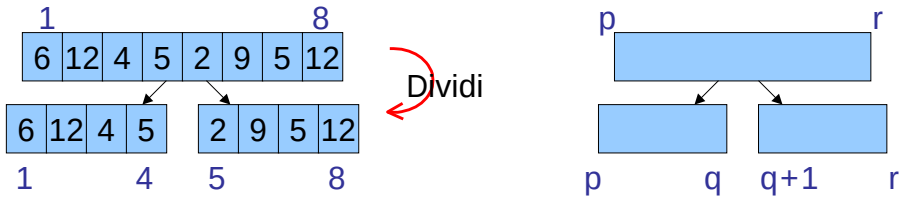
L'algoritmo noto come Merge Sort è l'applicazione diretta del paradigma Divide et Impera al problema dell'ordinamento.

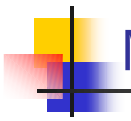


Merge Sort: Dividi

Il passo di divisione consiste semplicemente nel partizionare il vettore di partenza in due sotto-vettori, alla sinistra ed alla destra di un punto di divisione.

Solitamente il punto di divisione viene scelto al centro del sotto-vettore.





Merge Sort: terminazione

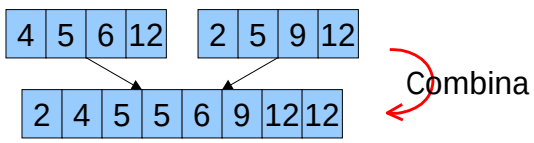
La condizione di terminazione si ha quando il sotto-vettore ha un solo elemento (p=r) oppure nessun elemento (p>r).

Merge Sort: Combina

Il passo di ricombinazione è basato sulla *fusione* di due vettori ordinati:

- Dati due vettori ordinati, costruire un terzo vettore contenente gli stessi elementi, complessivamente ordinati

Tale algoritmo è realizzabile in tempo $\Theta(n)$.



Pseudo-codice

```
MERGE-SORT(A, p, r)
1  if p < r                                  } Terminazione
2     then q ← ⌊(p + r)/2⌋                   } Dividi
3          MERGE-SORT(A, p, q)               } Risolvi
4          MERGE-SORT(A, q + 1, r)           }
5          MERGE(A, p, q, r)                 } Combina
```


Esercizio

Si scriva lo pseudo-codice della procedura Merge, garantendo che il suo tempo di esecuzione sia $\Theta(n)$.

Soluzione: procedura Merge

```
MERGE(A, p, q, r)
1  i ← p ; j ← q+1 ; k ← 1
2  while( i ≤ q and j ≤ r )
3       if( A[i] < A[j])   B[k] ← A[i] ; i ← i+1
4       else               B[k] ← A[j] ; j ← j+1
5       k ← k+1
6  while( i≤q )       B[k]←A[i] ; i←i+1; k←k+1
7  while( j≤r )       B[k]←A[j] ; j←j+1; k←k+1
8  A[p..q] ← B[1..k-1]
```


Soluzione: proce

Prende ogni volta il più piccolo tra i primi due elementi non ancora considerati

```
MERGE(A, p, q, r)
1  i ← p ; j ← q+1 ; k ← 1
2  while( i ≤ q and j ≤ r )
3       if( A[i] < A[j])   B[k] ← A[i] ; i ← i+1
4       else               B[k] ← A[j] ; j ← j+1
5       k ← k+1
6  while( i≤q )       B[k]←A[i] ; i←i+1; k←k+1
7  while( j≤r )       B[k]←A[j] ; j←j+1; k←k+1
8  A[p..q] ← B[1..k-1]
```

Esaurisce la “coda” del sottovettore restante

Esercizio proposto

Dimostrare che la procedura Merge così proposta ha una complessità $\Theta(n)$.

Complessità (I)

L'analisi della procedura Merge Sort porta alle seguenti formule:

- Terminazione: semplice test, $\Theta(1)$
- Dividi (2): calcola la metà di un array, $D(n)=\Theta(1)$
- Risolvi (3-4): risolve 2 sottoproblemi di dimensione n/2 ciascuno, $2T(n/2)$
- Combina (5): basata su Merge, $C(n) = \Theta(n)$.

Complessità (II)

T(n) =

- $\Theta(1)$ per n = 1
- $2T(n/2) + \Theta(n)$ per n > 1

Si può dimostrare che la soluzione è:

- $T(n) = \Theta(n \log n)$

Avvertenza

Non è detto che tutte le procedure ricorsive abbiamo tutte complessità $\Theta(n \log n)$.

Ad esempio, un merge sort con una partizione asimmetrica (q=p+1), degenera in un insertion sort, con n chiamate ricorsive in ciascuna delle quali viene aggiunto un elemento al set già ordinato, ottenendo $\Theta(n^2)$.

Esercizio proposto

Si implementi in linguaggio C l'algoritmo Merge Sort.

Si verifichi sperimentalmente il suo comportamento $\Theta(n \log n)$.