

## Complessita' temporale del Merge Sort

```
Merge-Sort(A, p, r)
If (p < r)
    then q =  $\lfloor (p+r)/2 \rfloor$ 
        Merge-Sort(A, p, q)
        Merge-Sort(A, q+1, r)
        Merge(A, p, q, r)
```

Per costruzione, ci aspettiamo che il comportamento asintotico del **merge sort** sia migliore del comportamento asintotico di **IS, SS** e **BS**. Abbiamo, infatti, mostrato che, per il problema dell'ordinamento:

- l'approccio **divide et impera** e' asintoticamente piu' vantaggioso dell' approccio incrementale;
- L'approccio divide-et-impera puo' essere applicato in maniera **ricorsiva**.

Vogliamo ora determinare in maniera quantitativa la complessita' asintotica del merge-sort.

## Complessita temporale del Merge Sort

```
Merge-Sort(A, p, r)
If (p < r)
    then q = ⌊(p+r)/2⌋
        Merge-Sort(A, p, q)
        Merge-Sort(A, q+1, r)
        Merge(A, p, q, r)
```

Il merge sort e' un **algoritmo ricorsivo** →

Il tempo di esecuzione del MS verifica un **equazione di ricorrenza**

$$T_{ms}(n) = d(n) + 2 * T_{ms}(n/2) + c(n)$$

$d(n) \rightarrow$  tempo necessario a dividere in 2 sequenze lunghe  $n/2$  →  $\Theta(1)$

$c(n) \rightarrow$  tempo necessario per combinare 2 sequenze ordinate di  $n/2$  elementi (merge) →  $\Theta(n)$

$$T_{ms}(n) = 2 * T_{ms}(n/2) + f(n)$$

$$f(n) = d(n) + c(n) = \Theta(n)$$

Questa equazione vale per tutti i valori di  $n$  eccetto che per  $n=1$ :

**Notare -**

Se conosco  $T_{ms}(n/2)$  ed  $f(n)$  posso determinare  $T_{ms}(n)$ .

## Complessita' temporale MS

Supponiamo di sapere:

$T_{ms}(n^*) = C \quad \rightarrow$  Per ordinare una sequenza lunga  $n^*$  impiego un tempo  $C$ .

Consideriamo un valore di  $n$  generico. Ipotizziamo che valga la relazione:

$$n = n^* 2^h \quad \rightarrow \quad h = \log_2(n/n^*)$$

Abbiamo che:

$$T_{ms}(n) = 2T_{ms}(n/2) + bn \quad \text{N.B. } f(n) = \Theta(n) \approx bn \text{ (asintot.)}$$

Consideriamo che  $T_{ms}(n/2) = 2 T_{ms}(n/4) + b n/2$ . Quindi:

$$T_{ms}(n) = 2 ( 2 T_{ms}(n/4) + b n/2 ) + b n$$

Di nuovo:  $T_{ms}(n/4) = 2 T_{ms}(n/8) + b n/4$ . Allora

$$\begin{aligned} T_{ms}(n) &= 2 [ 2 ( 2 T_{ms}(n/8) + b n/4 ) + bn/2 ] + bn = \\ &= 2^3 T_{ms}(n/2^3) + 2^2 bn/2^2 + 2 bn/2 + bn = \\ &= 2^3 T_{ms}(n/2^3) + 3 b n = \end{aligned}$$

Iterando:

$$T_{ms}(n) = 2^h T_{ms}(n/2^h) + h b n = (n/n^*) C + b \log_2(n/n^*) n = \Theta(n \log n)$$

Abbiamo ottenuto:

$$T_{ms}(n) = 2^h T_{ms}(n/2^h) + h n = (n/n^*) C + \log_2 (n/n^*) (bn) = \Theta(n \log n)$$

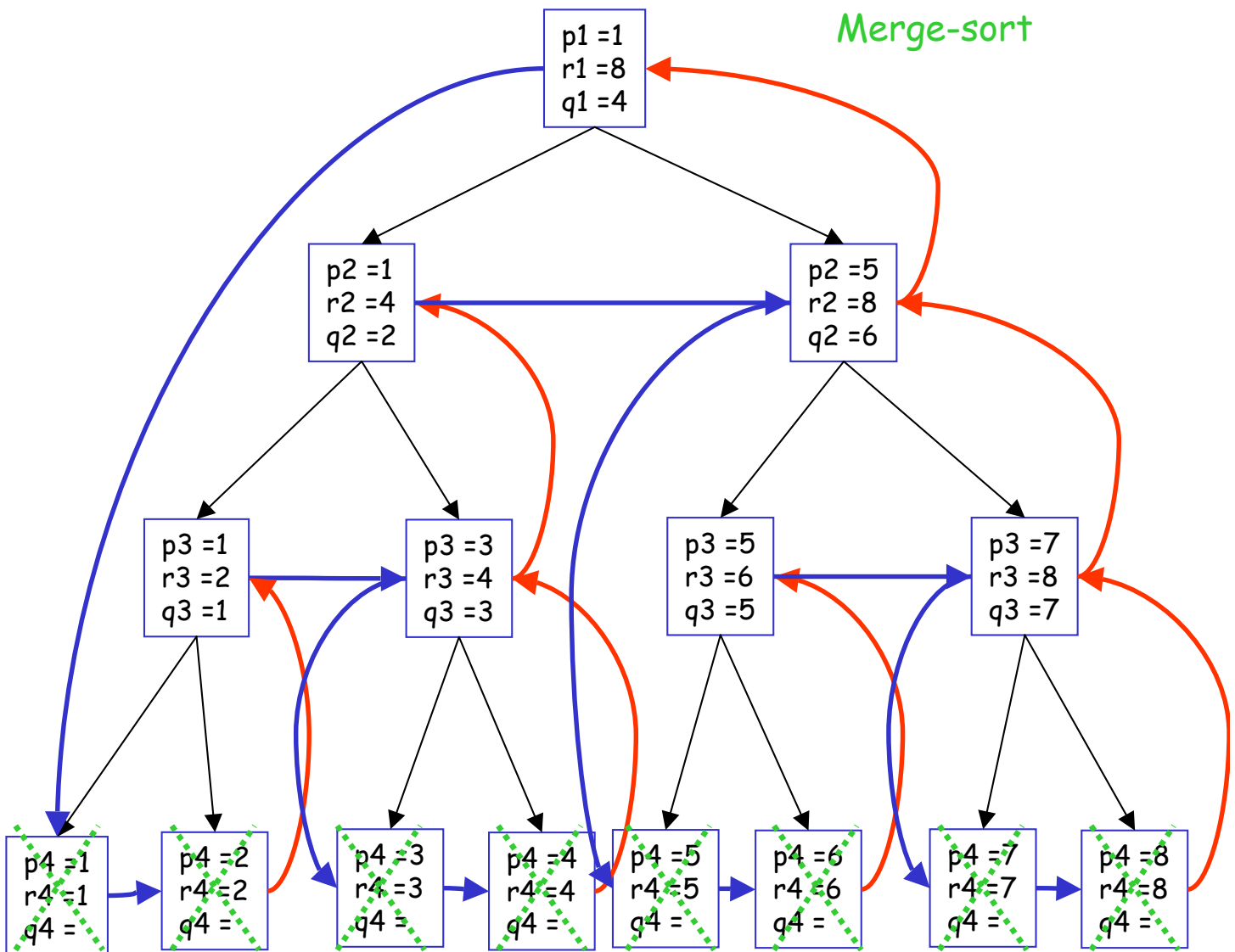
Perche'? Cosa vuol dire? Torniamo al nostro esempio .....

$$\begin{aligned} MS(8) &= 8 MS(1) + \text{tempo speso per le fusioni} = \\ &= 8MS(1) + 4M(1 \rightarrow 2) + 2M(2 \rightarrow 4) + 1M(4 \rightarrow 8) = \end{aligned}$$

Essendo  $\text{merge} = \Theta(n)$  abbiamo:

$2M(2 \rightarrow 4) = M(4 \rightarrow 8)$  e  $4M(1 \rightarrow 2) = 2M(2 \rightarrow 4)$ . Quindi:

$$MS(8) = 8MS(1) + 3M(4 \rightarrow 8) = 8MS(1) + \log_2(8) M(4 \rightarrow 8)$$



In conclusione

La **complessita' temporale** dell'algoritmo merge sort e':

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

Cio' implica che l'algoritmo merge sort e' un **algoritmo di ordinamento ottimale**.

### Esercizio -

Modificare il merge sort in modo tale che sottosequenze di dimensione  $k$  vengano ordinate utilizzando IS. Analizzare la complessita' temporale dell'algoritmo ottenuto.

Implementare l'algoritmo di merge sort in maniera non-ricorsiva

## Teorema principale

Siano  $a, b, c$  costanti non negative . La soluzione dell'equazione di ricorrenza:

$$\begin{aligned} T(n) &= b && \text{per } n = 1 \\ &= aT(n/c) + b n && \text{per } n > 1 \end{aligned}$$

e' :

$$\begin{aligned} T(n) &= O(n) && \text{se } a < c \\ &O(n \log n) && \text{se } a = c \\ &O(n^{\log_c(a)}) && \text{se } a > c \end{aligned}$$