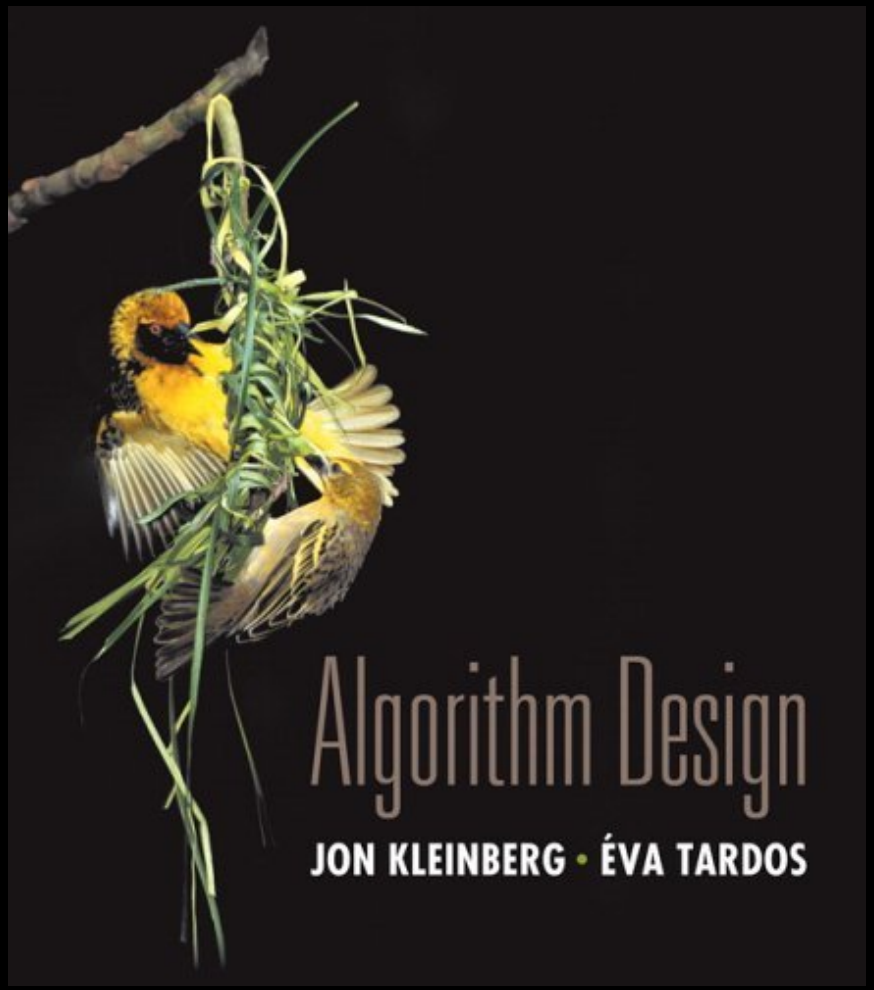


Chapter 5

Divide and Conquer

PEARSON
Addison Wesley

Divide-and-Conquer

Divide-and-conquer.

- Dividere il problema in diverse parti.
- Risolvere ogni parte ricorsivamente.
- Combinare le soluzioni dei sottoproblemi in soluzioni globali.

Uso più comune.

- Dividere il problema di tagli n in due parti uguali di taglia n/2.
- Risolvere ogni parte ricorsivamente.
- Combinare le soluzioni dei sottoproblemi in soluzioni globali in tempo lineare.

Efficienza.

- Brute force: $O(n^2)$.
- Divide-and-conquer: $O(n \log n)$.

Divide et impera.
Veni, vidi, vici.
- *Giulio Cesare*

Divide and Conquer

Algoritmi che vedremo:

- Mergesort
- Quicksort
- Counting Inversions
- Closest Pair of Points
- Integer Multiplication
- Matrix Multiplication

5.1 Mergesort

Ordinamento

Ordinamento. Dati n elementi, disporli in ordine crescente.

Applicazioni ovvie:
- List files in a directory.
- Organize an MP3 library.
- List names in a phone book.
- Display Google PageRank results.

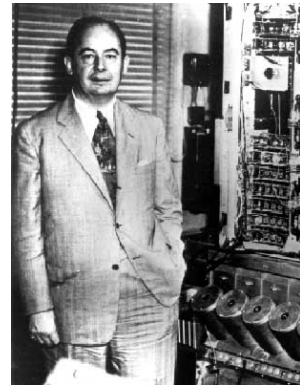
Problemi più semplici dopo ordinamento.
- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

Applicazioni non-ovvie:
- Data compression.
- Computer graphics.
- Interval scheduling.
- Computational biology.
- Minimum spanning tree.
- Supply chain management.
- Simulate a system of particles.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.
- . . .

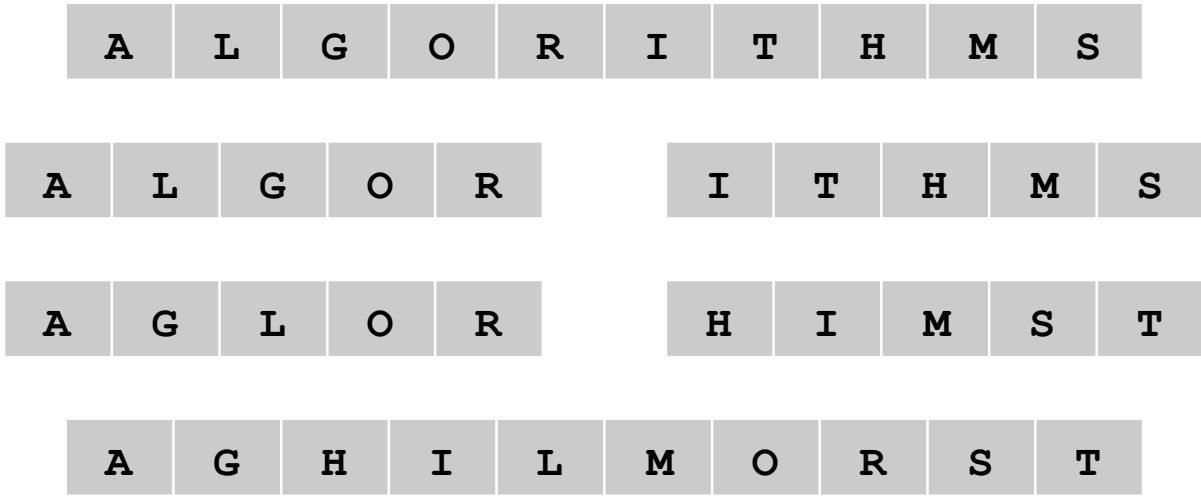
Mergesort

Mergesort.

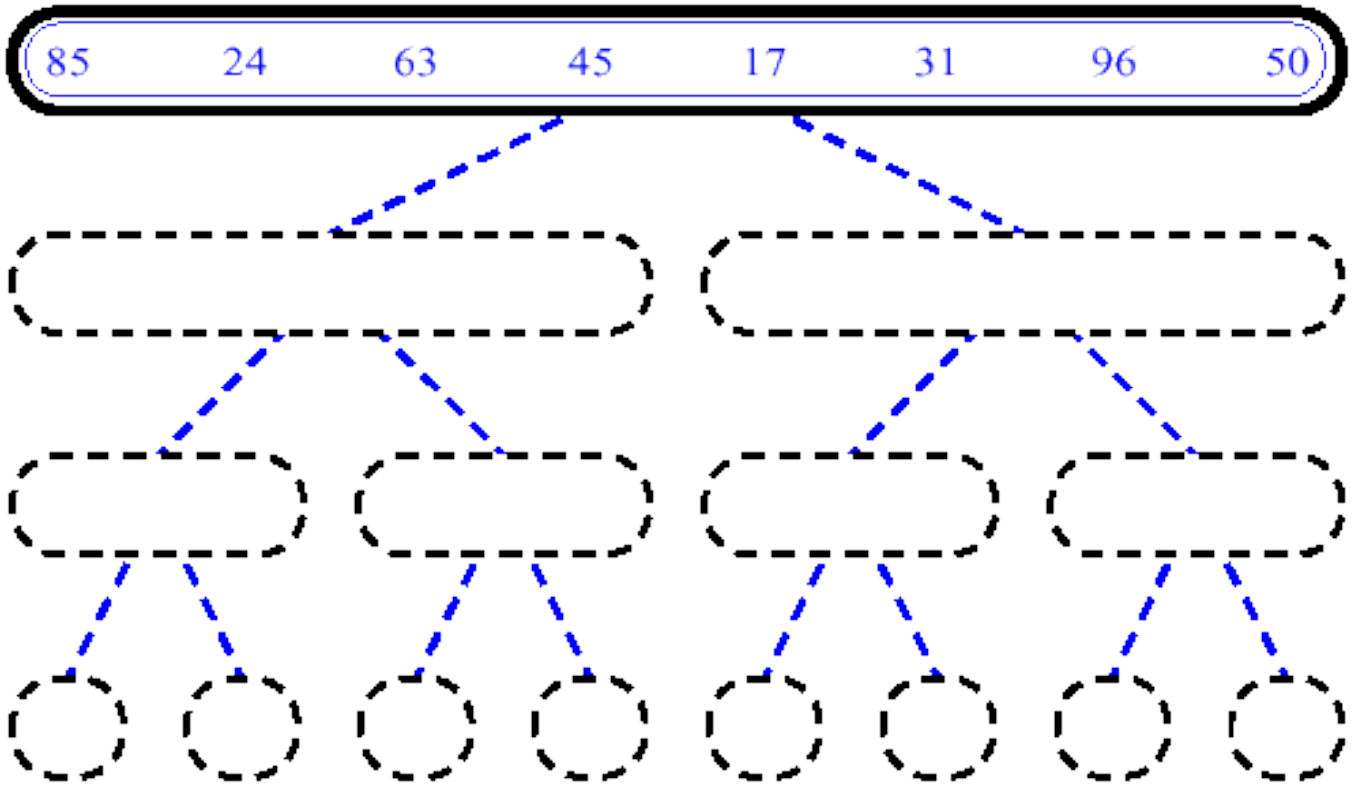
- Dividere array in due metà.
- Ricorsivamente ordina ogni metà.
- Merge due metà per ottenere tutta la sequenza ordinata.



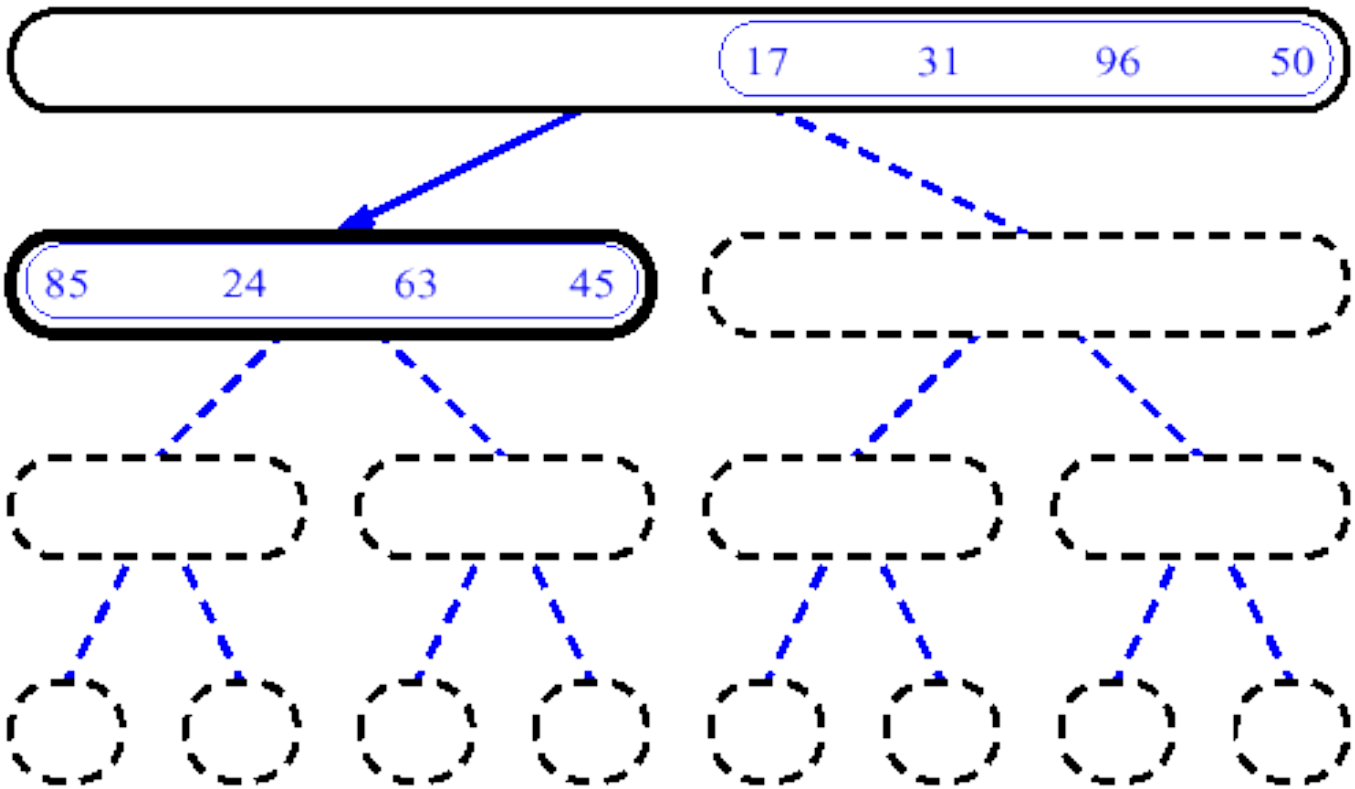
Jon von Neumann (1945)



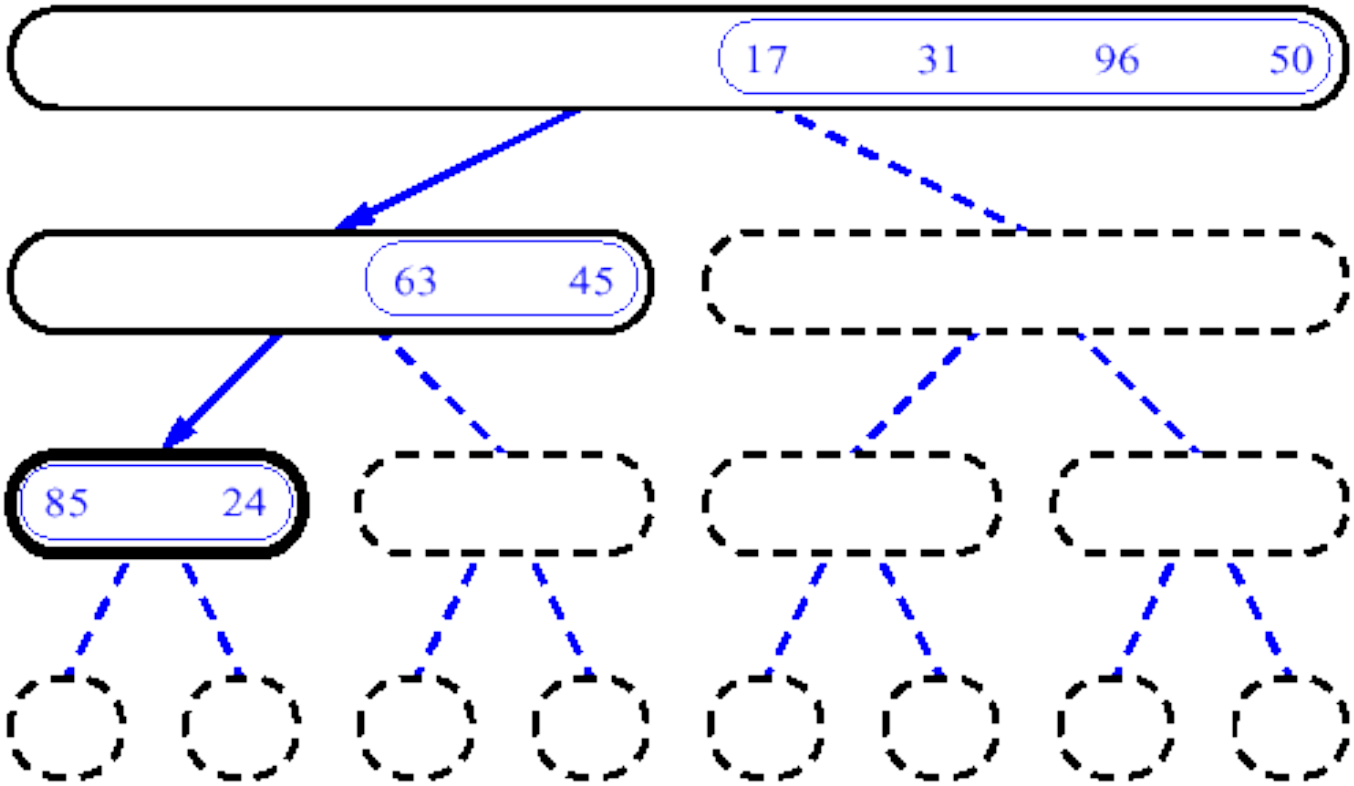
MergeSort (esempio) - 1



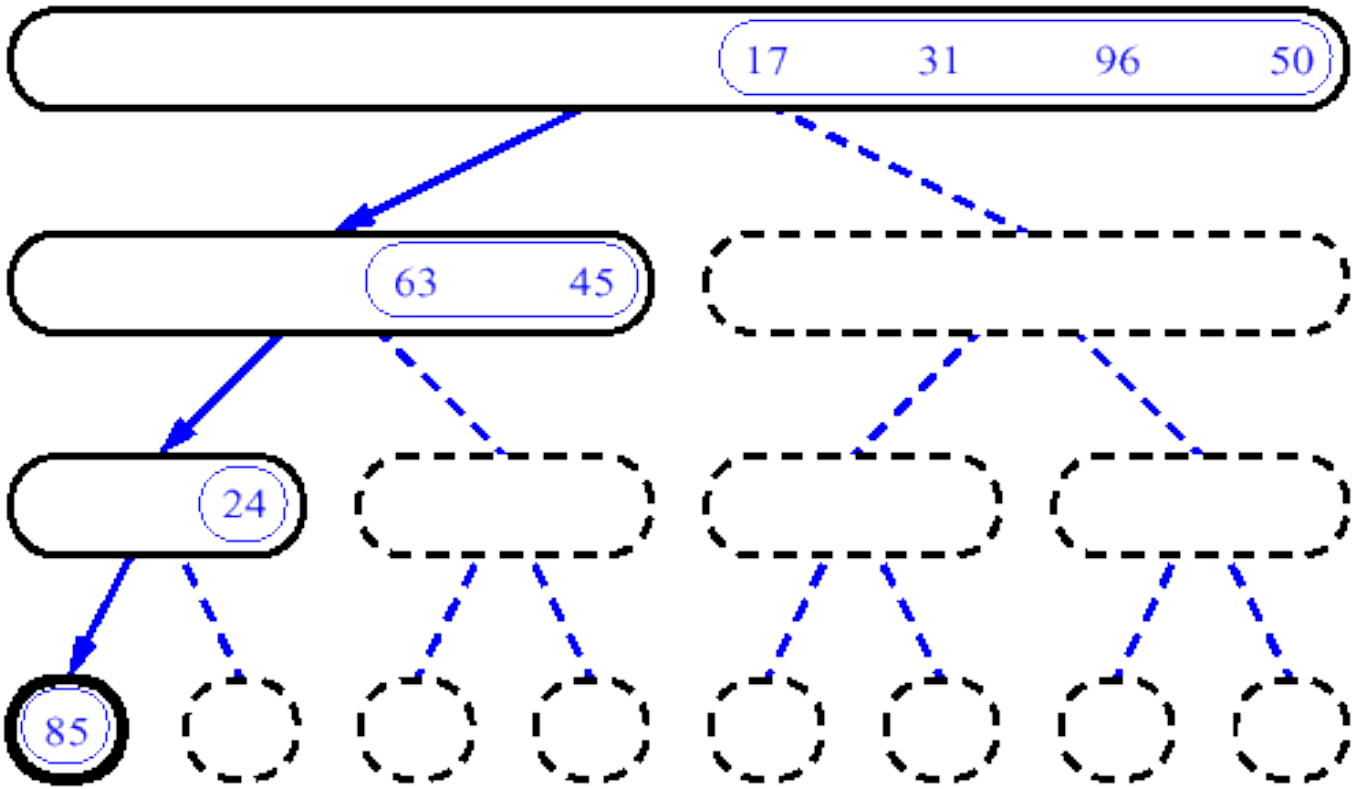
MergeSort (esempio) - 2



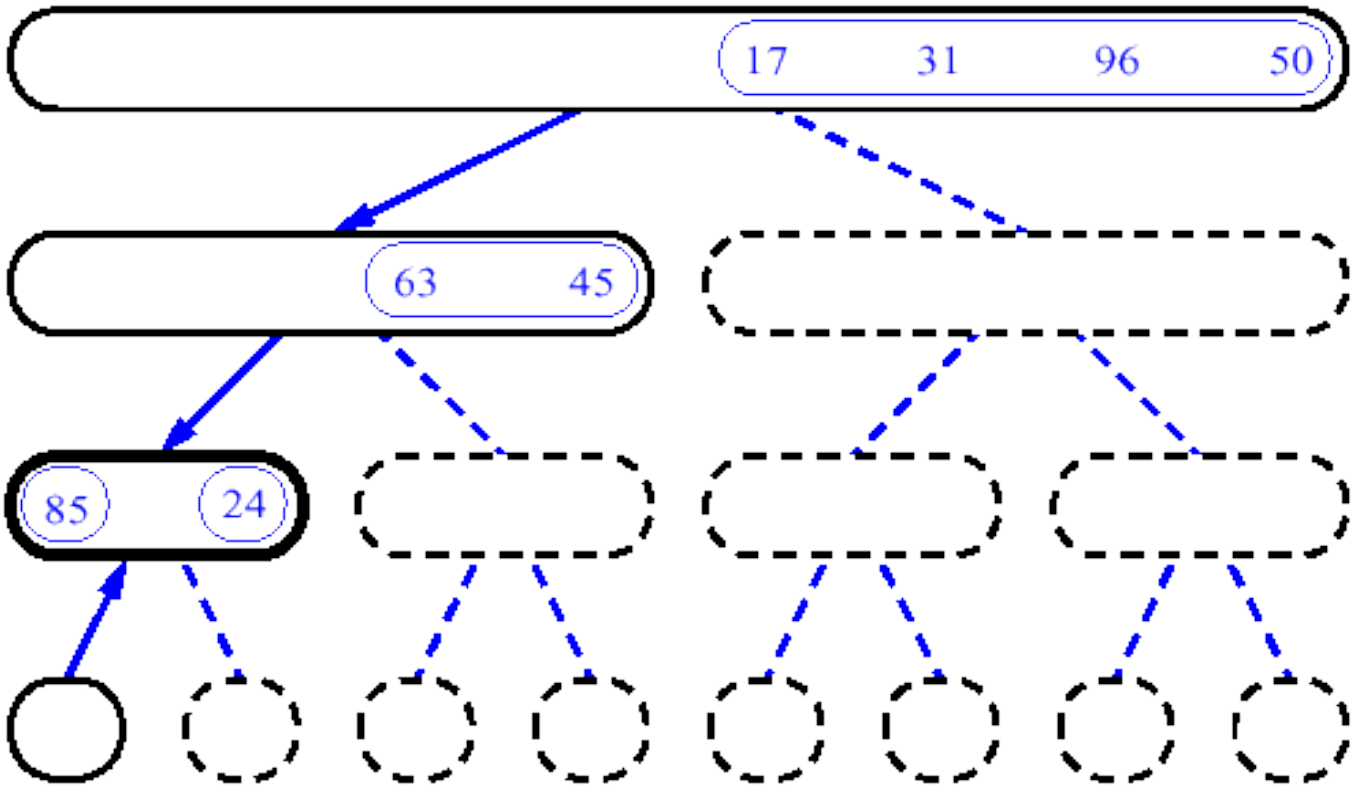
MergeSort (esempio) - 3



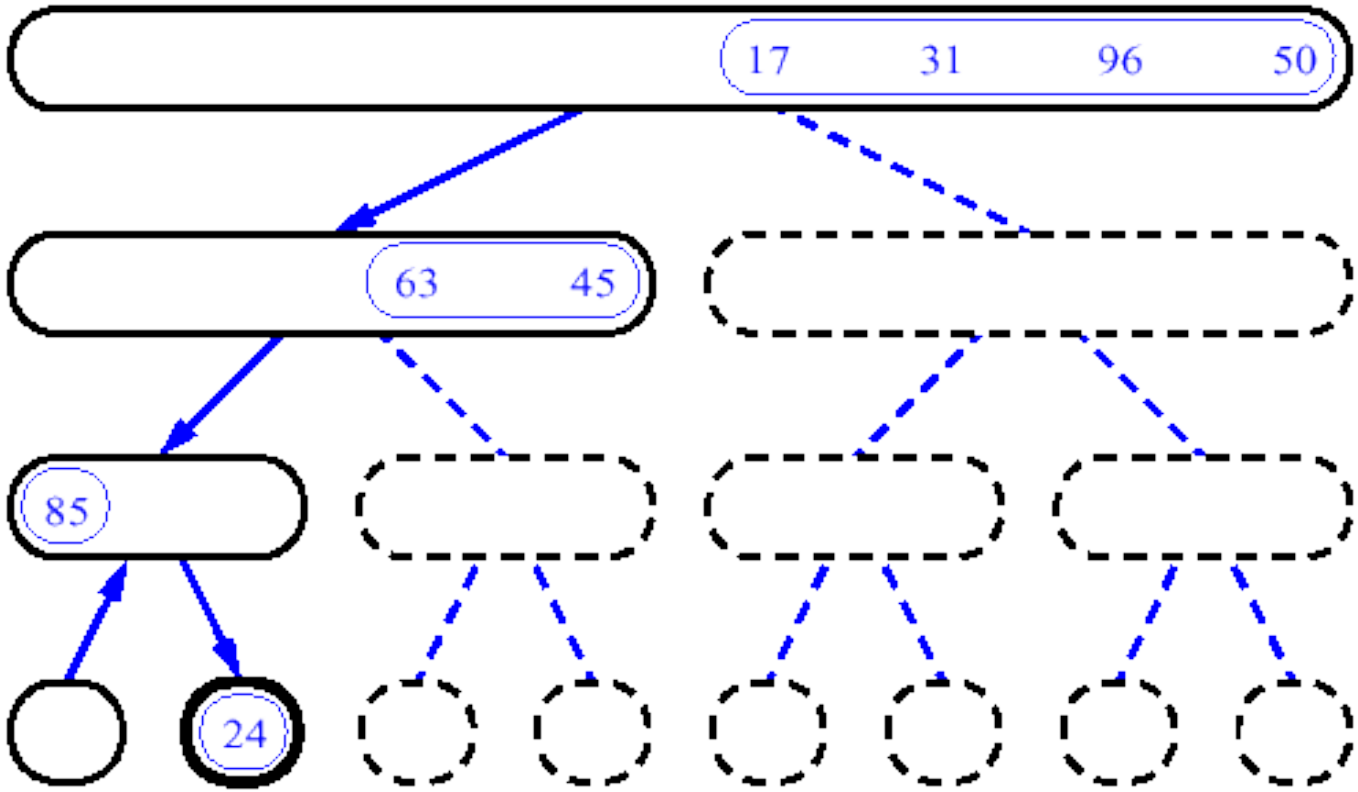
MergeSort (esempio) - 4



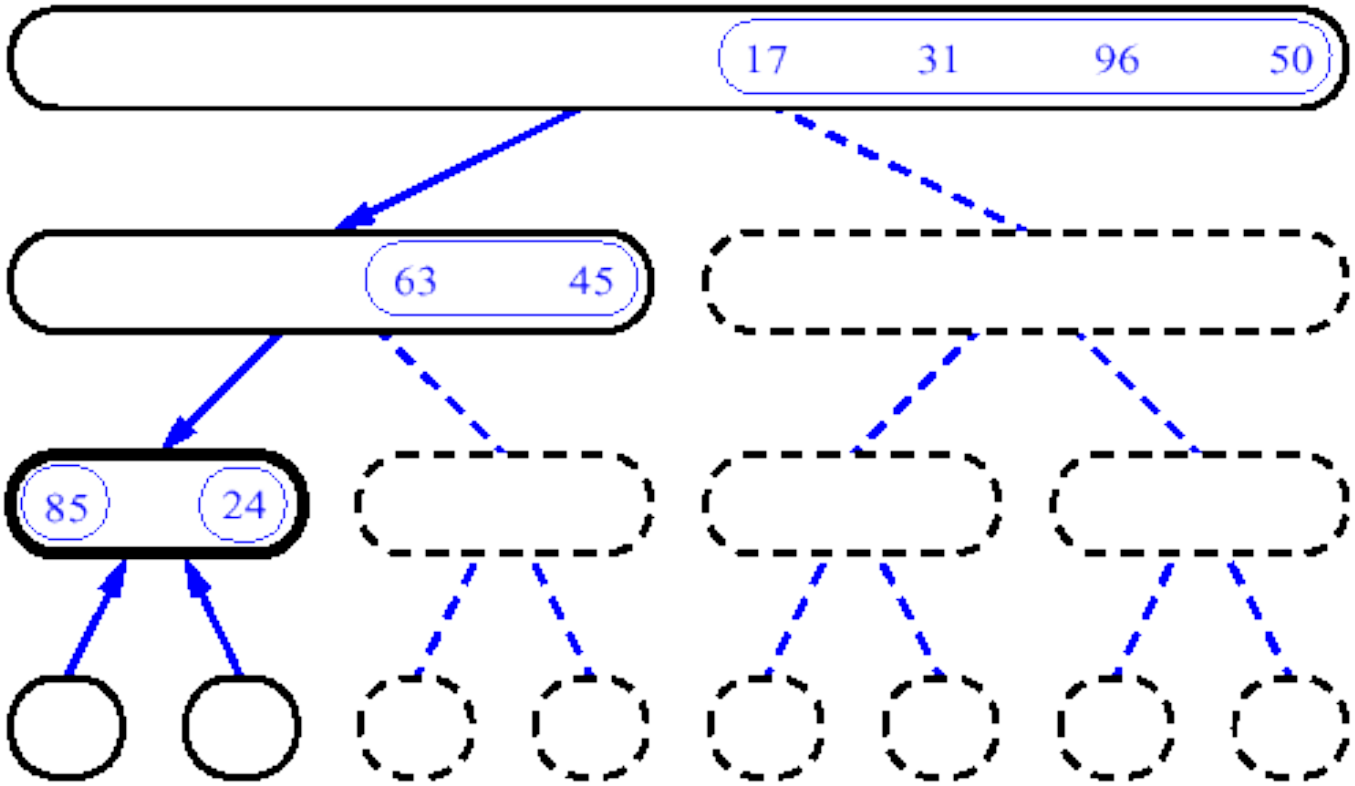
MergeSort (esempio) - 5



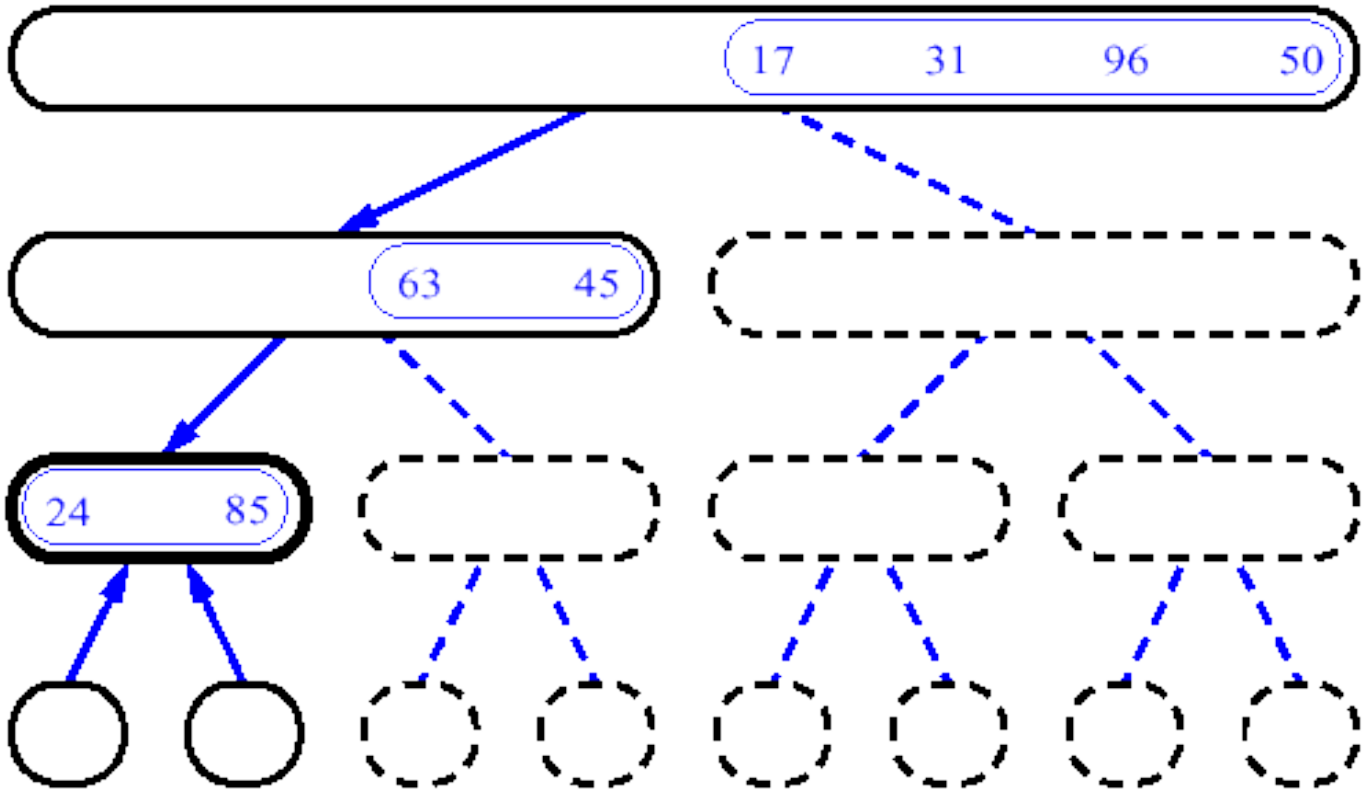
MergeSort (esempio) - 6



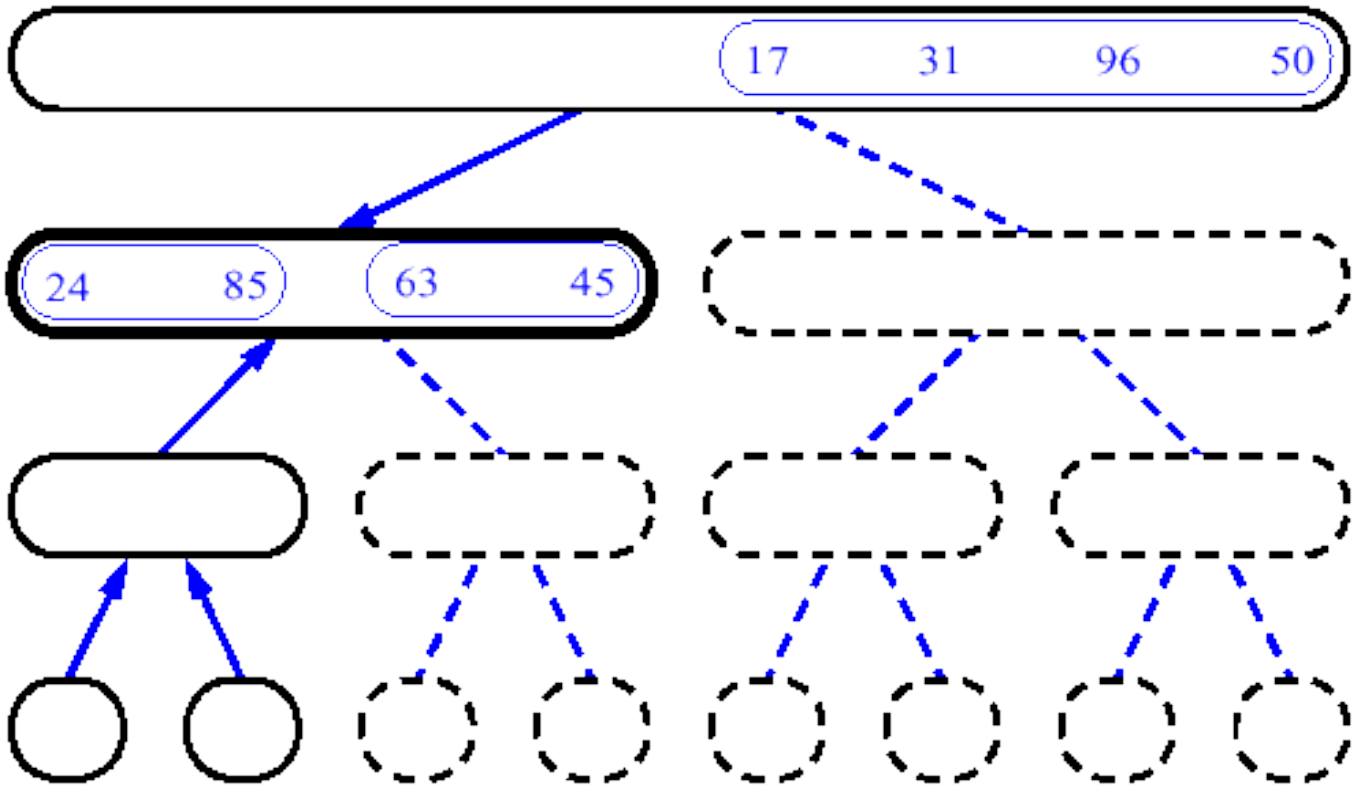
MergeSort (esempio) - 7



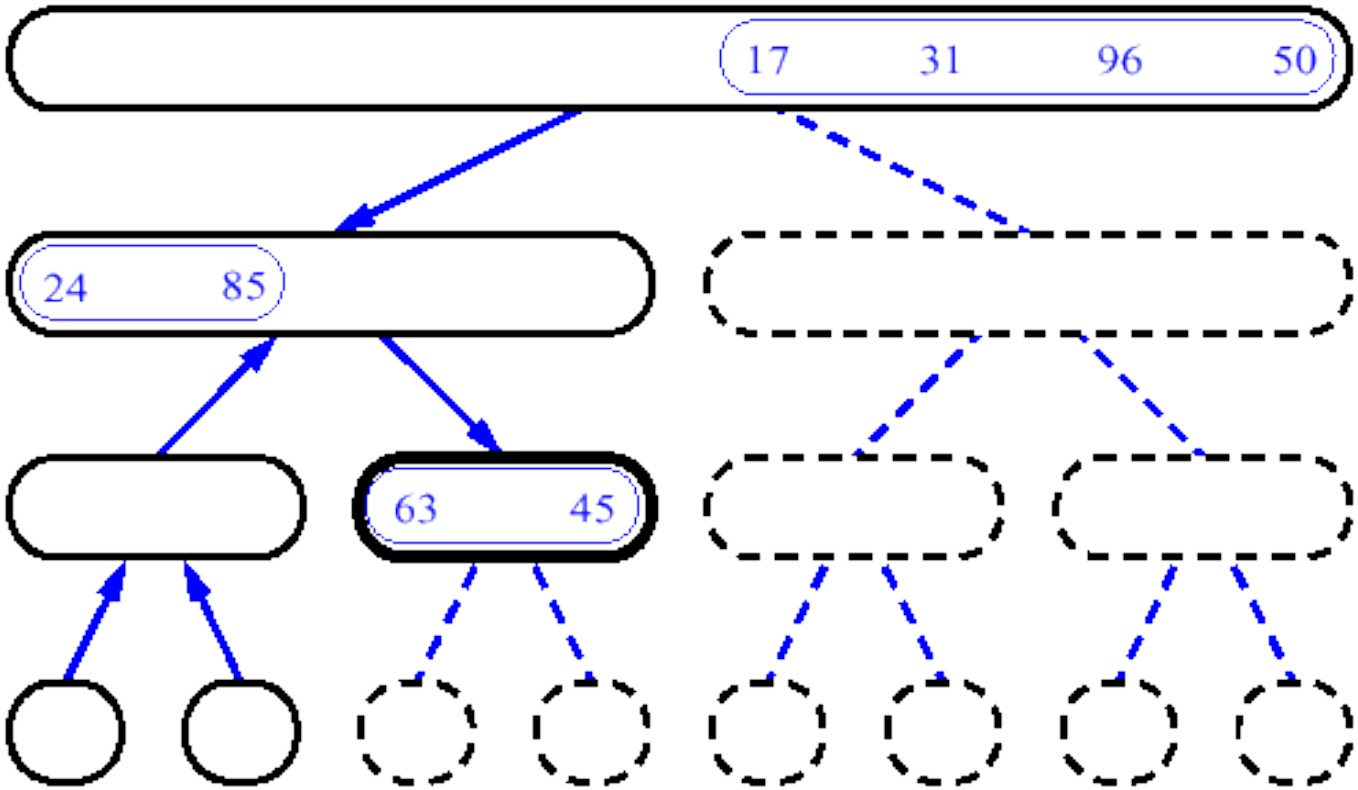
MergeSort (esempio) - 8



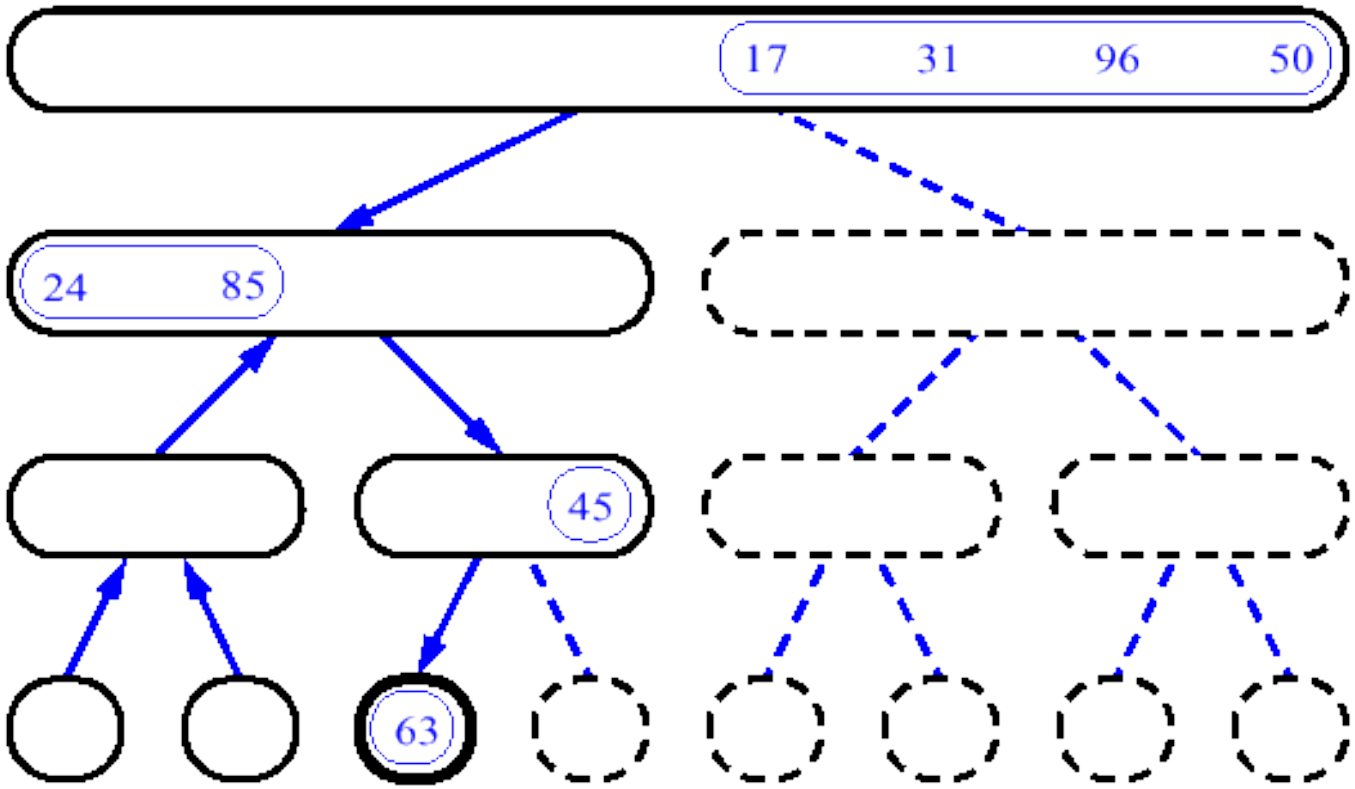
MergeSort (esempio) - 9



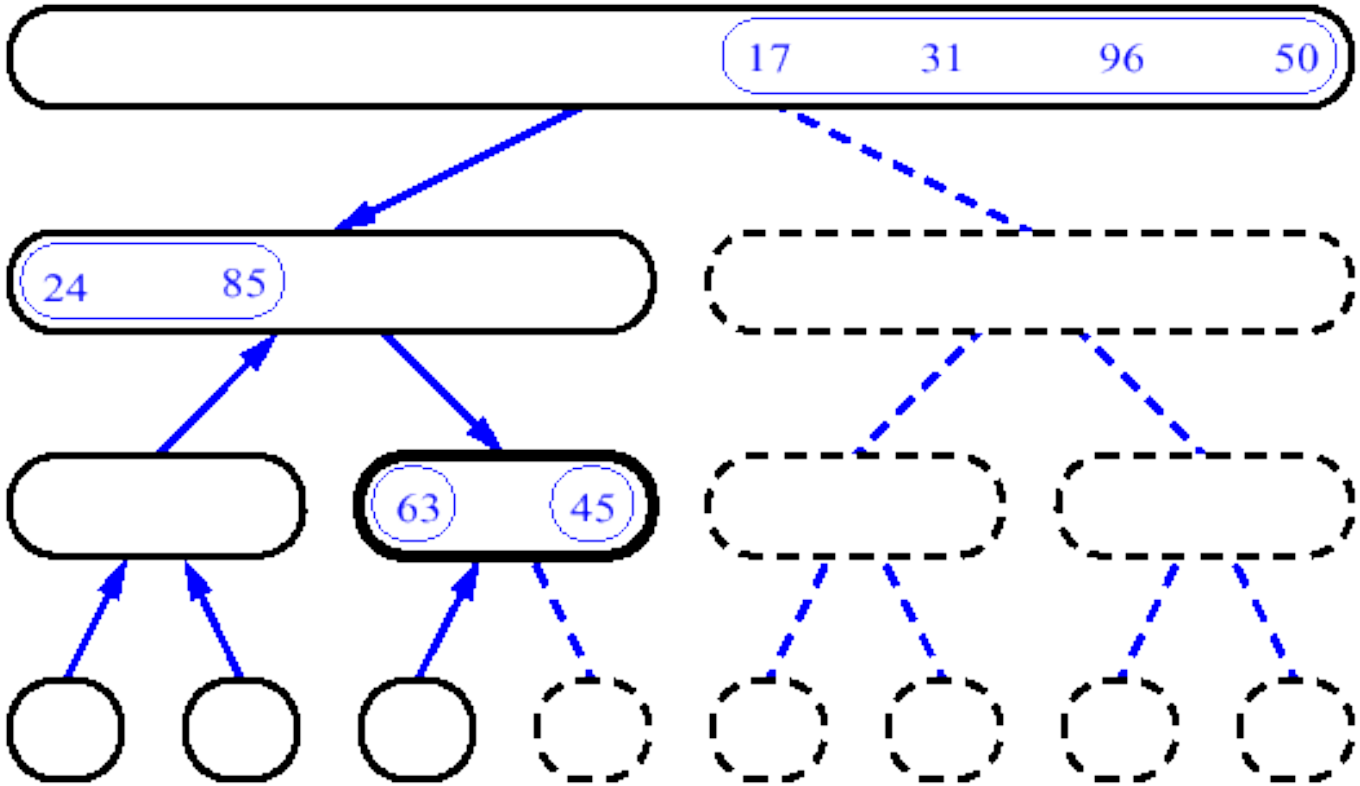
MergeSort (esempio) - 10



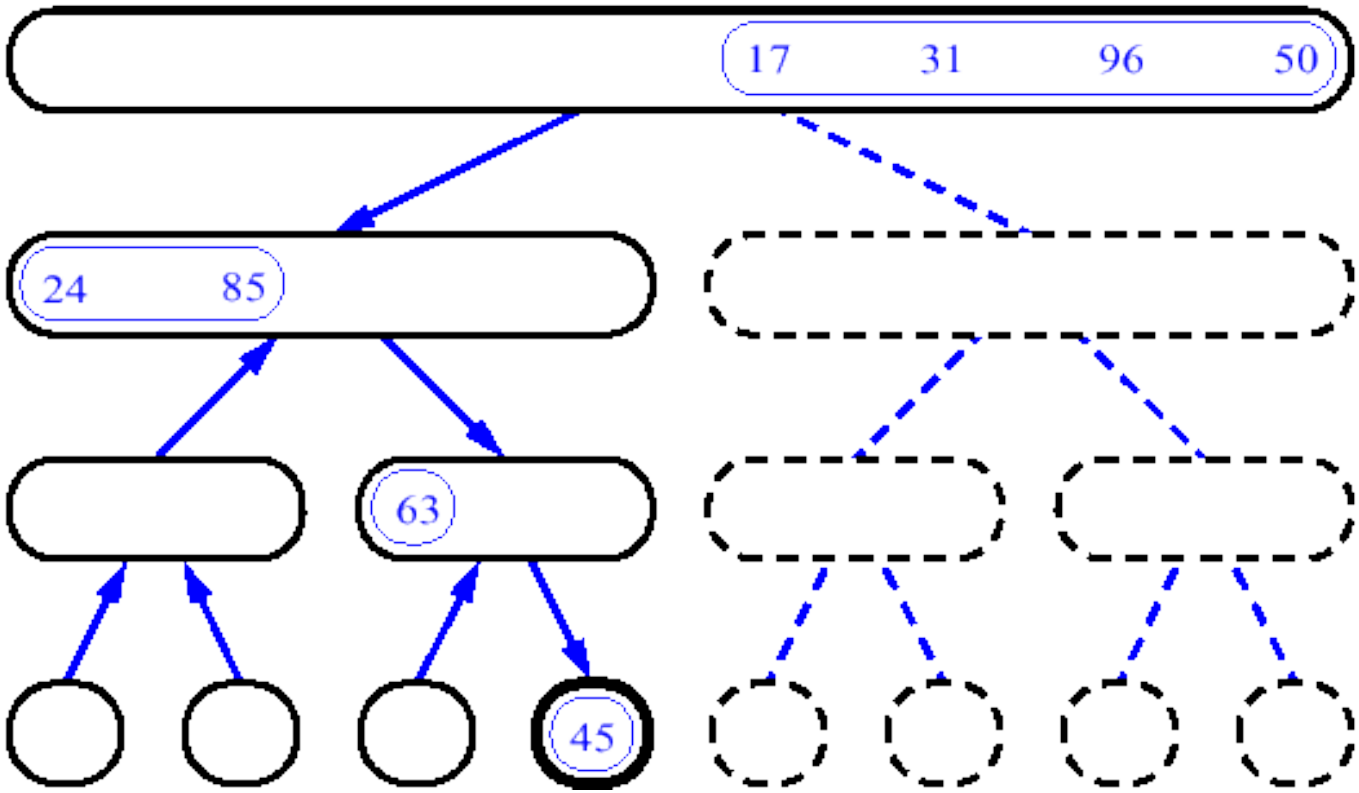
MergeSort (esempio) - 11



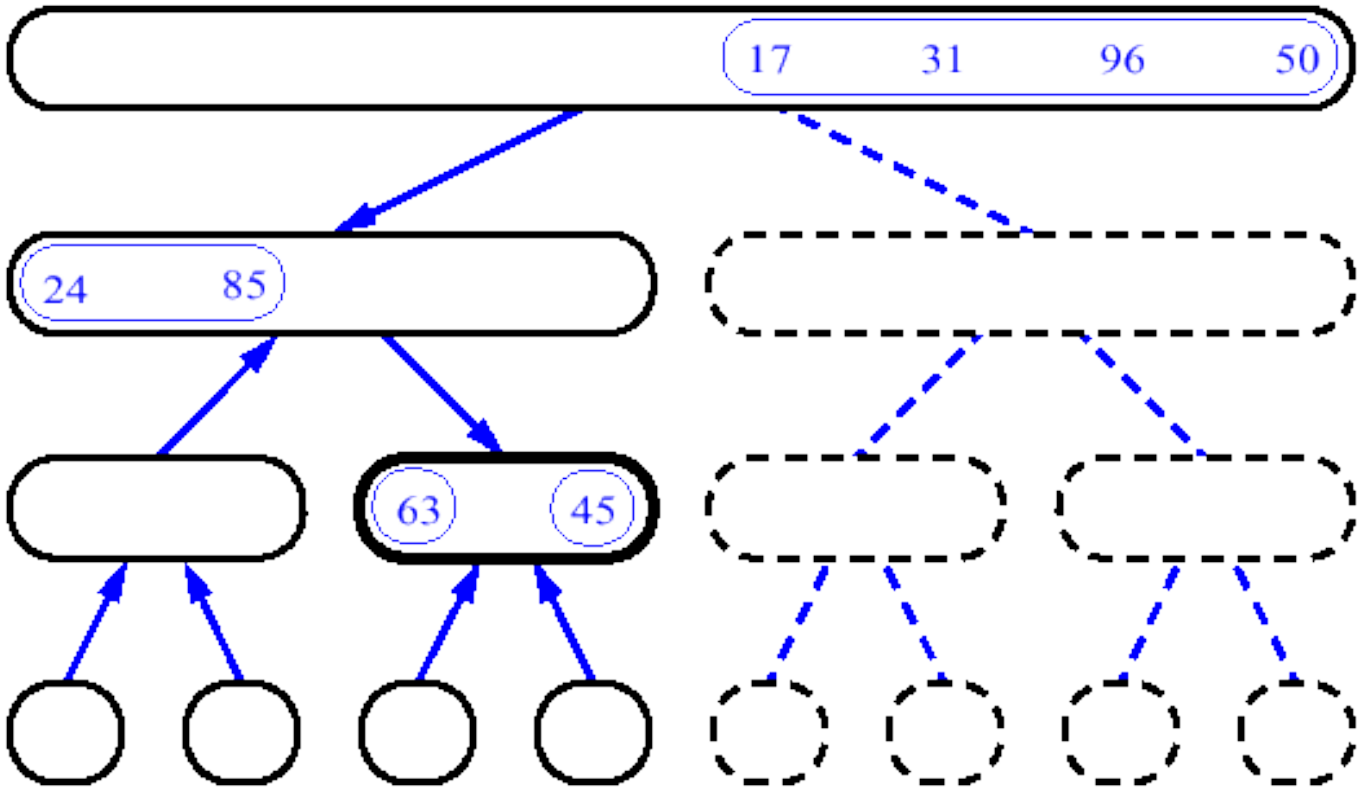
MergeSort (esempio) - 12



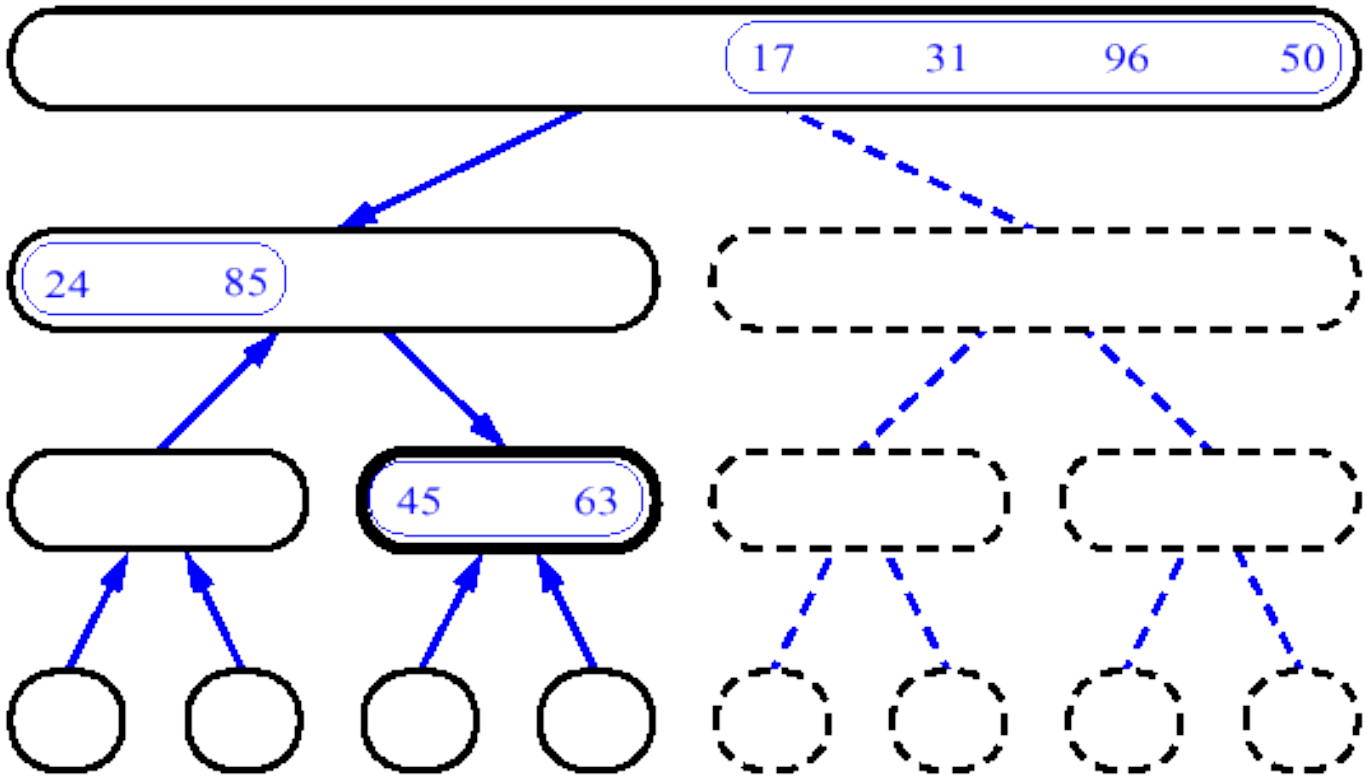
MergeSort (esempio) - 13



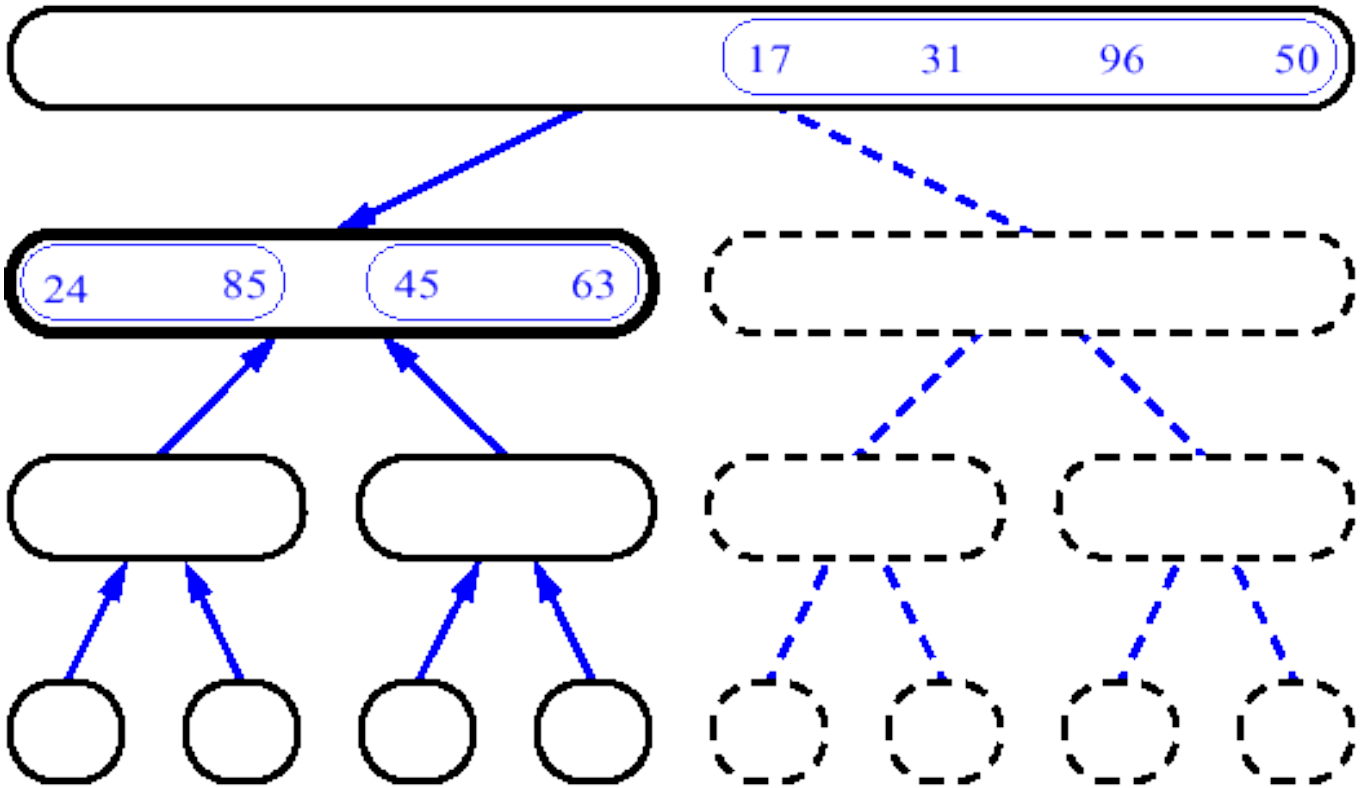
MergeSort (esempio) - 14



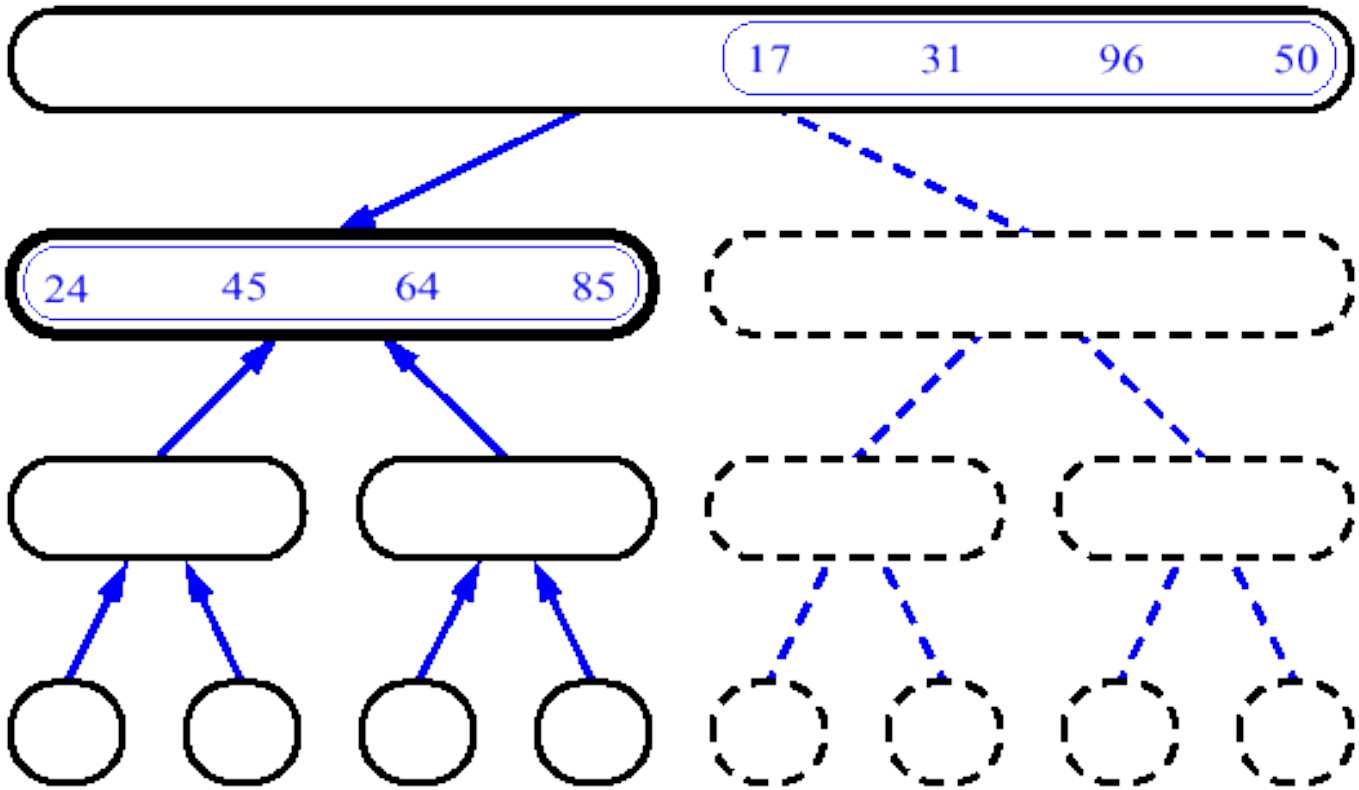
MergeSort (esempio) - 15



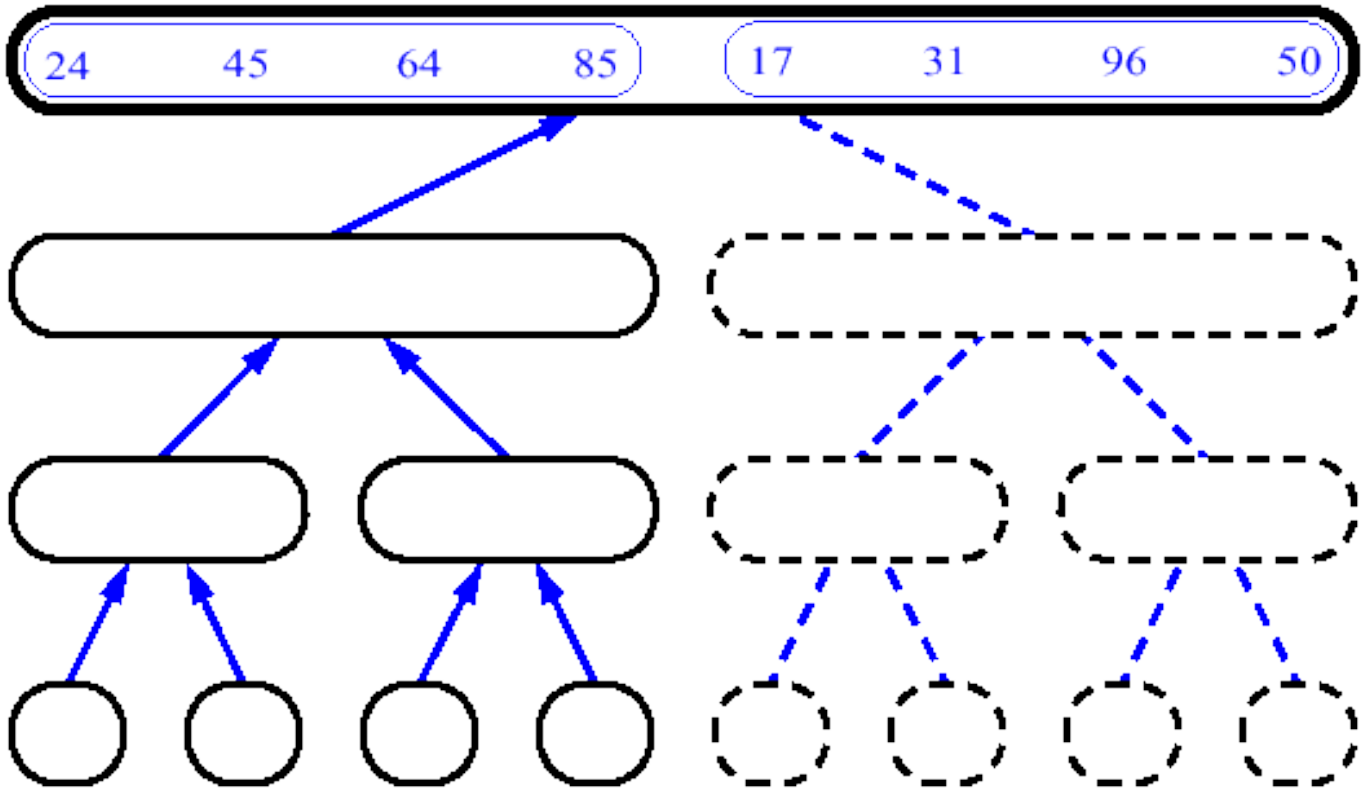
MergeSort (esempio) - 16



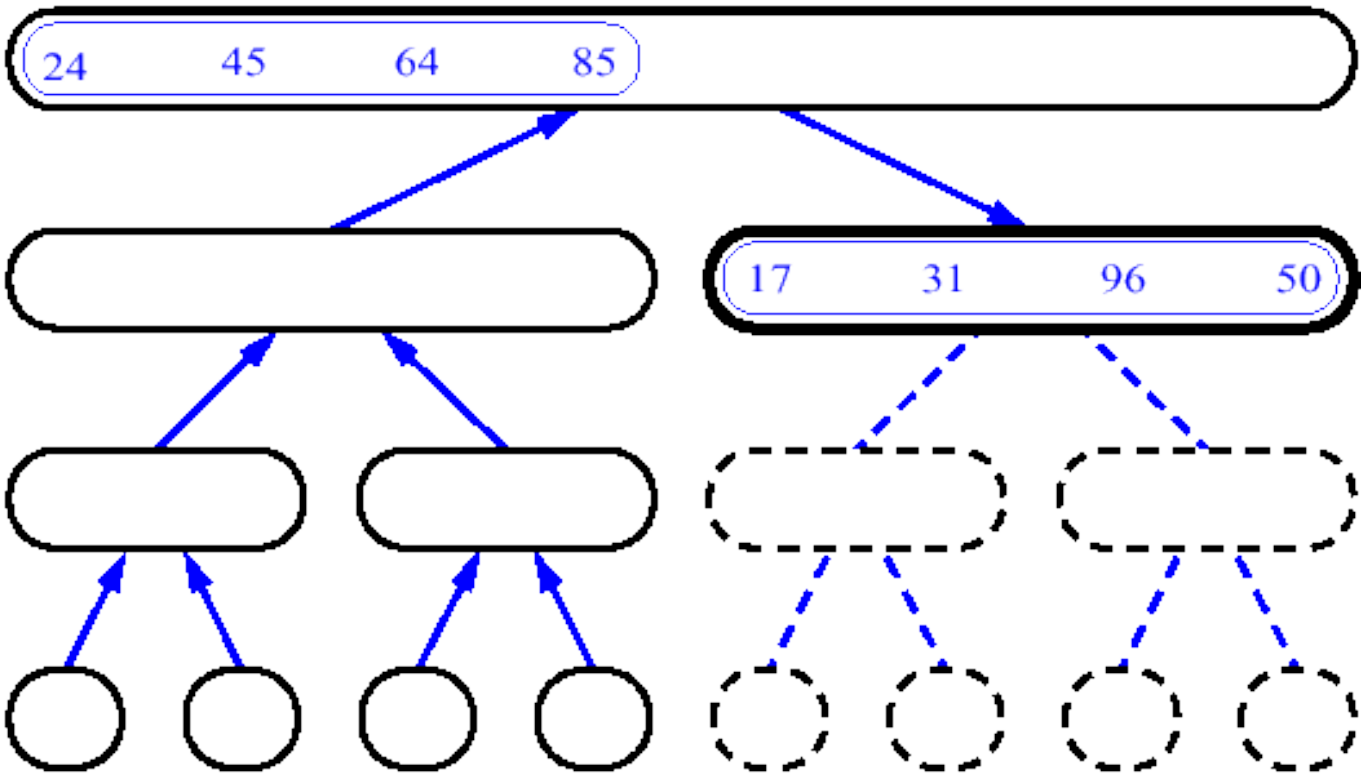
MergeSort (esempio) - 17



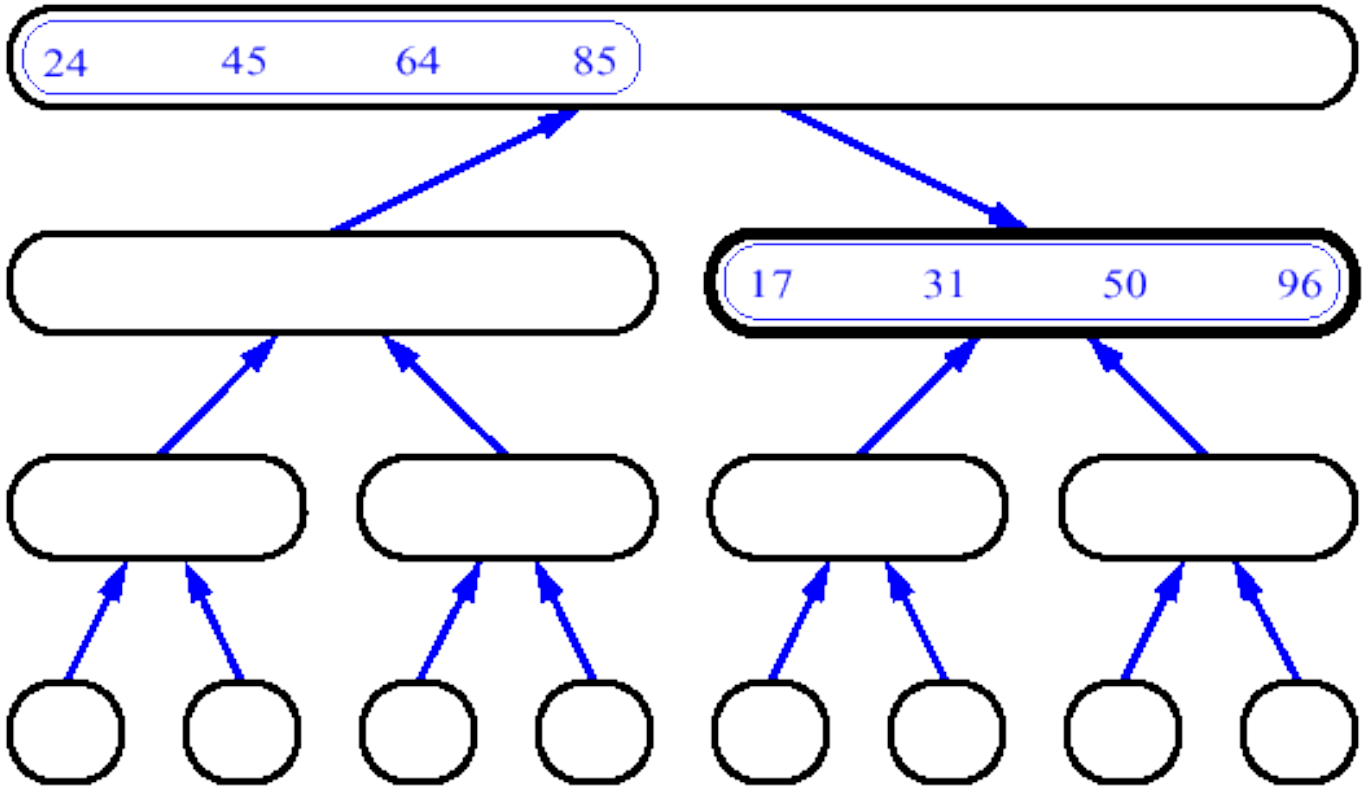
MergeSort (esempio) - 18



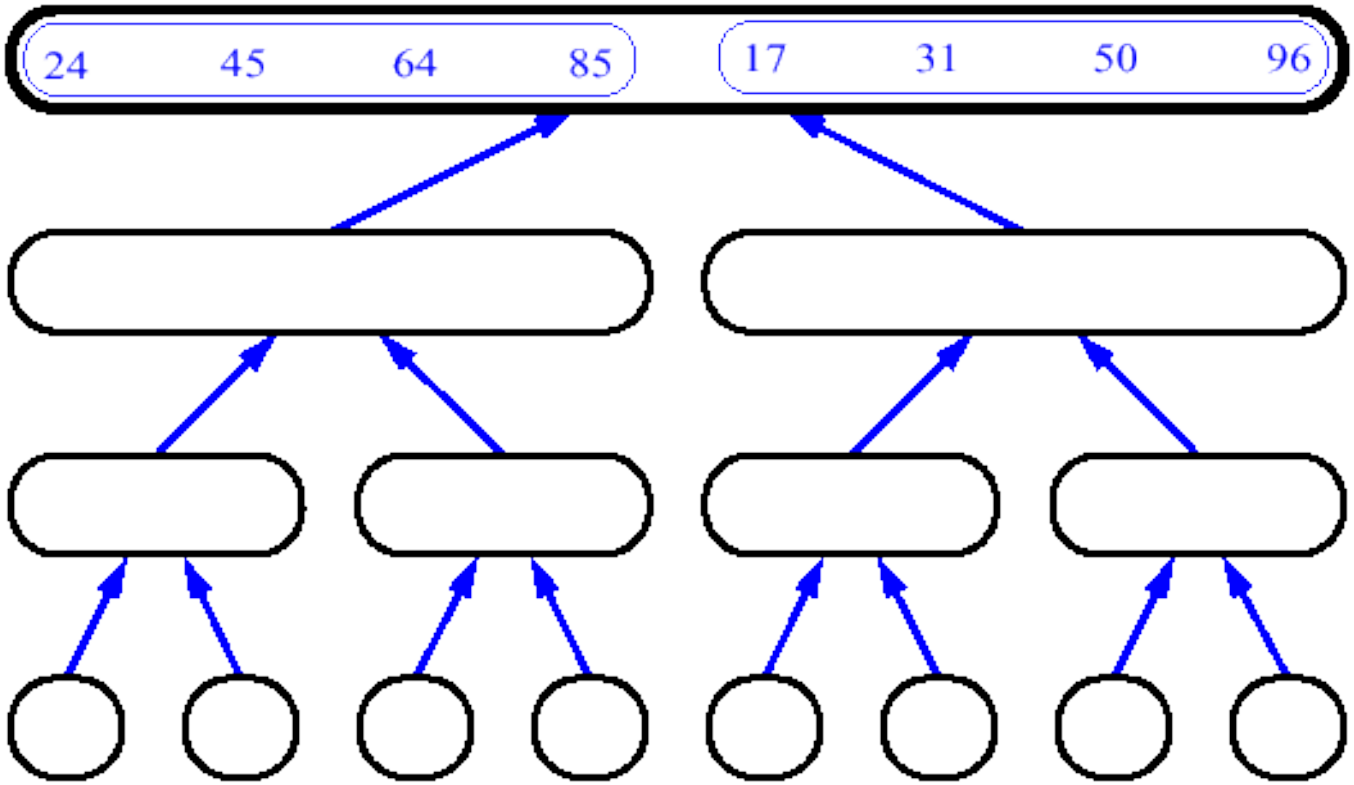
MergeSort (esempio) - 19



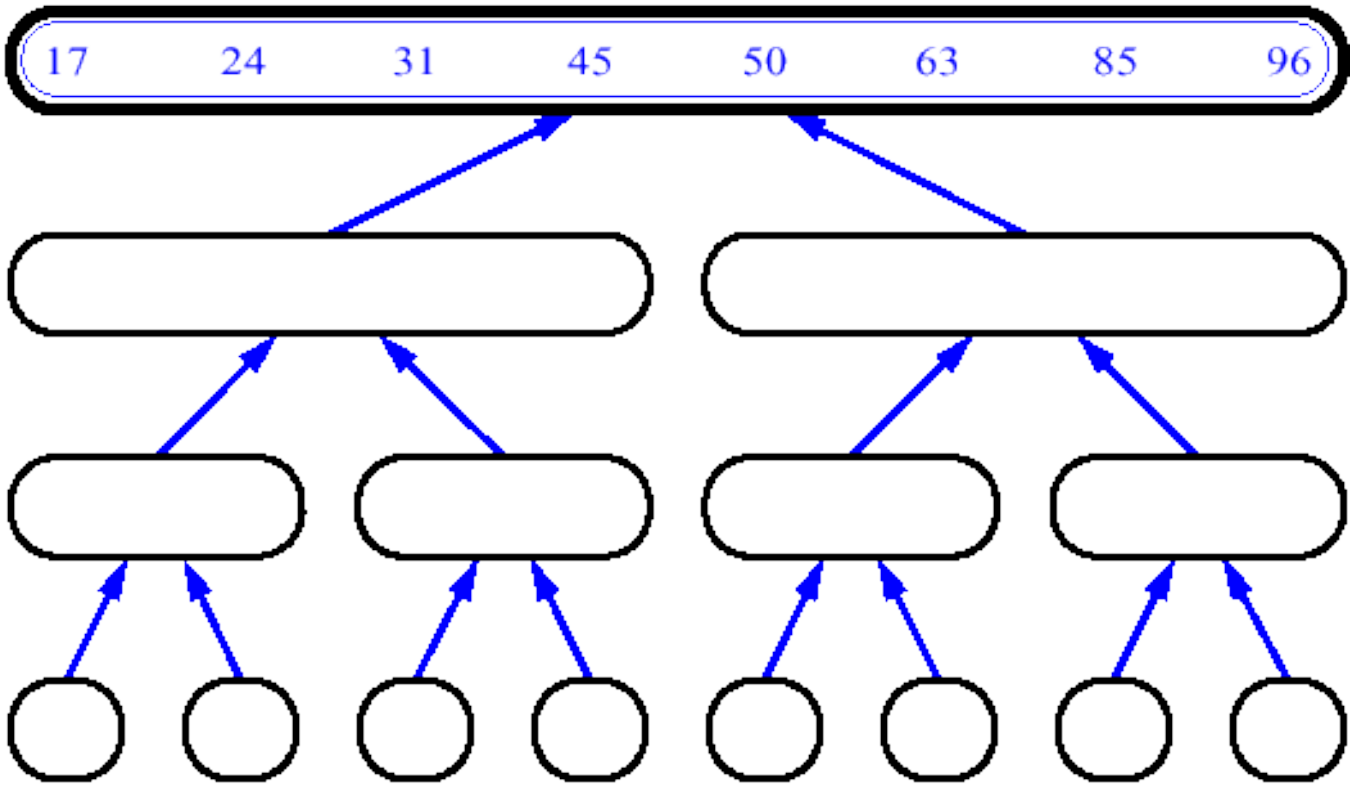
MergeSort (esempio) - 20



MergeSort (esempio) - 21



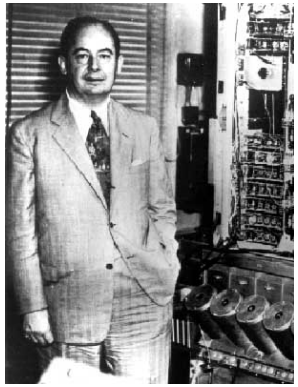
MergeSort (esempio) - 22



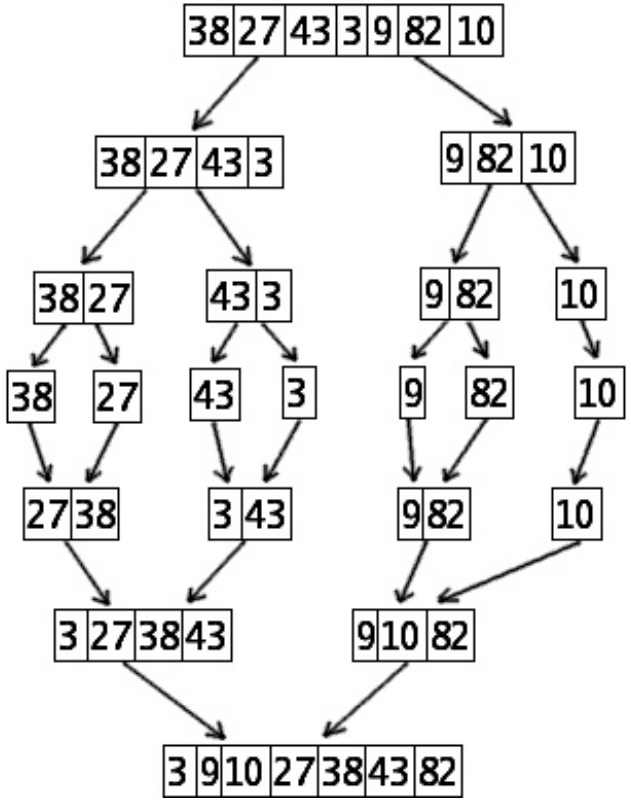
Mergesort

Mergesort.

- Dividere array in due metà.
- Ricorsivamente ordina ogni metà.
- Merge due metà per ottenere tutta la sequenza ordinata.



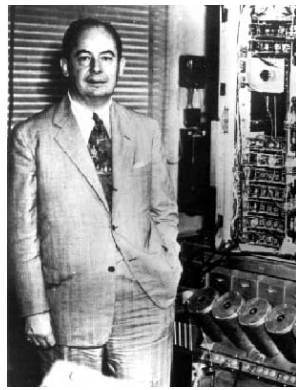
Jon von Neumann (1945)



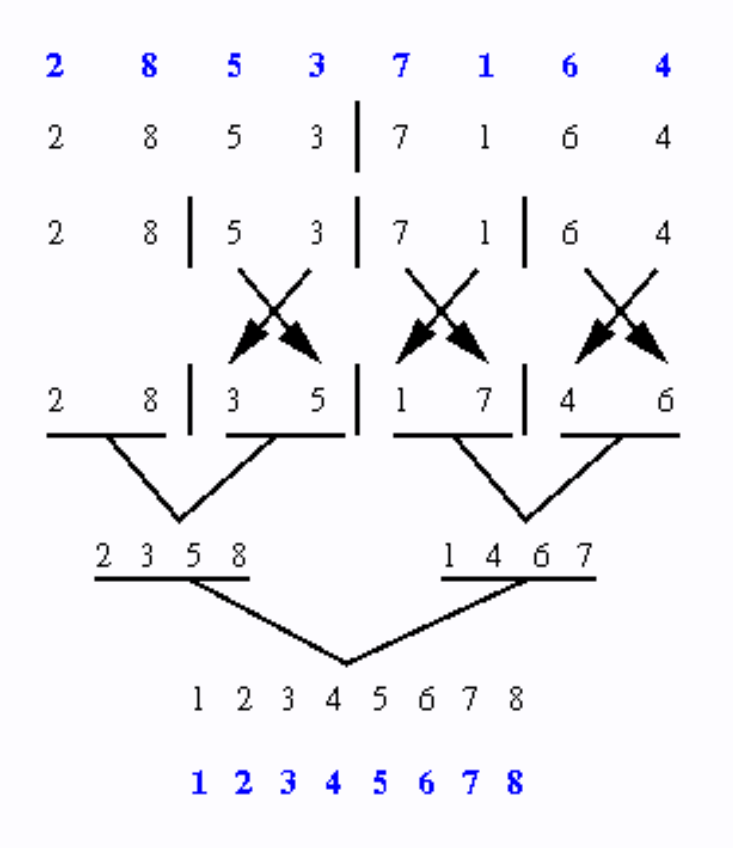
Mergesort

Mergesort.

- Dividere array in due metà.
- Ricorsivamente ordina ogni metà.
- Merge due metà per ottenere tutta la sequenza ordinata.



Jon von Neumann (1945)

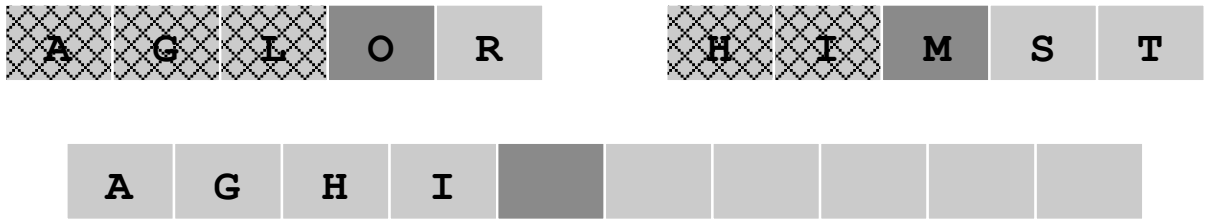


Merge

Merging. Combina due liste ordinate in un'unica lista ordinata.

Come effettuare il merge efficientemente?

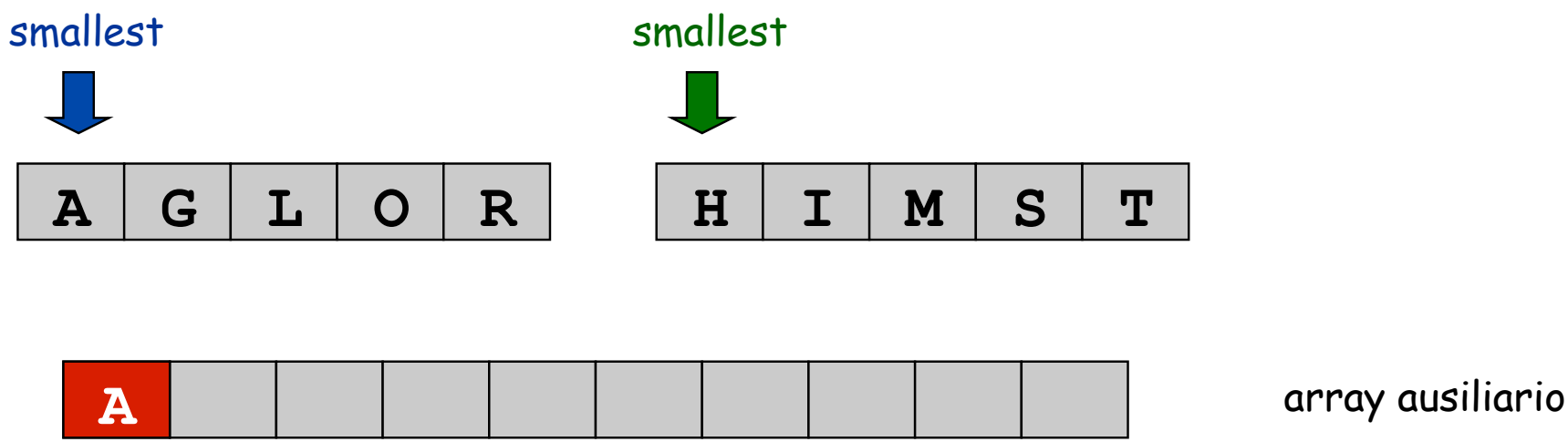
- Uso di un array temporaneo.



Merge

Merge.

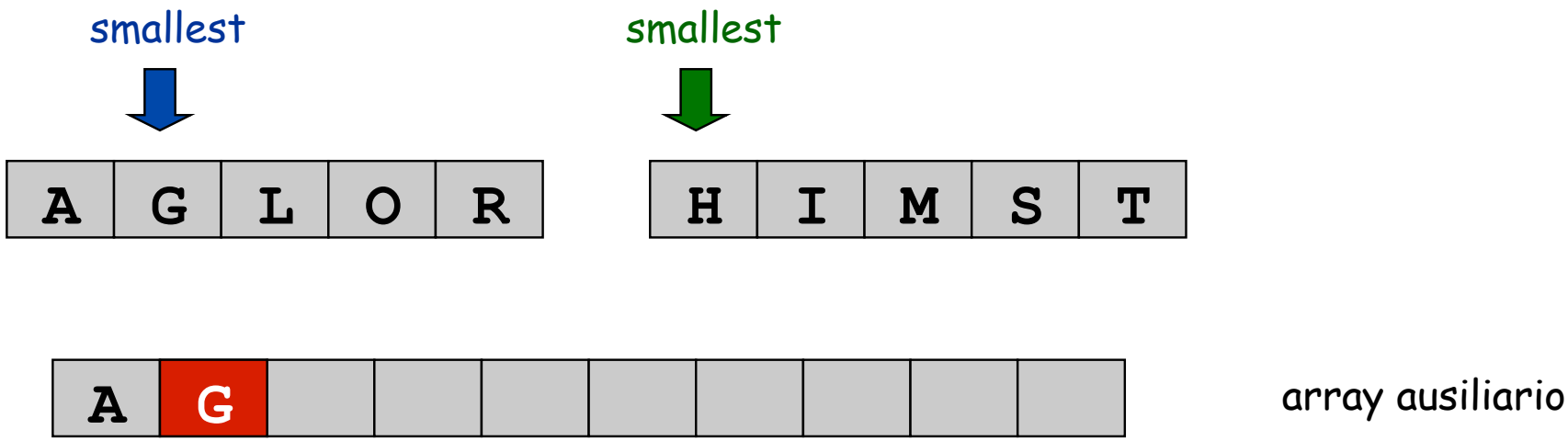
- Mantenere traccia dell' elemento più piccolo in ogni metà ordinata.
- Inserire il più piccolo dei due elementi in un array ausiliario.
- Ripetere fino a terminare gli elementi nelle due metà.



Merge

Merge.

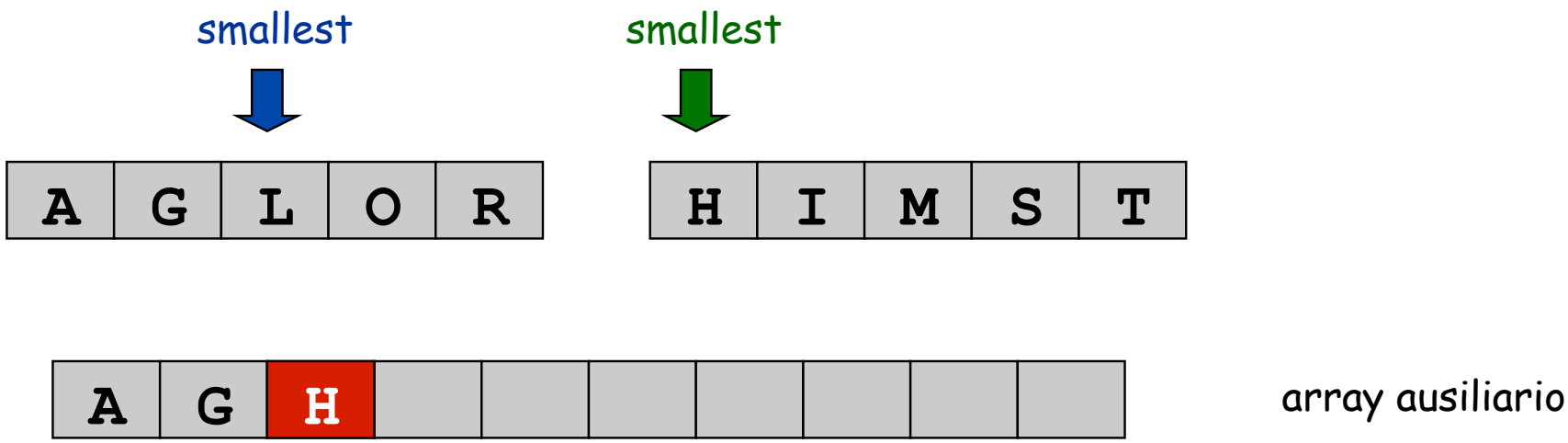
- Mantenere traccia dell' elemento più piccolo in ogni metà ordinata.
- Inserire il più piccolo dei due elementi in un array ausiliario.
- Ripetere fino a terminare gli elementi nelle due metà.



Merge

Merge.

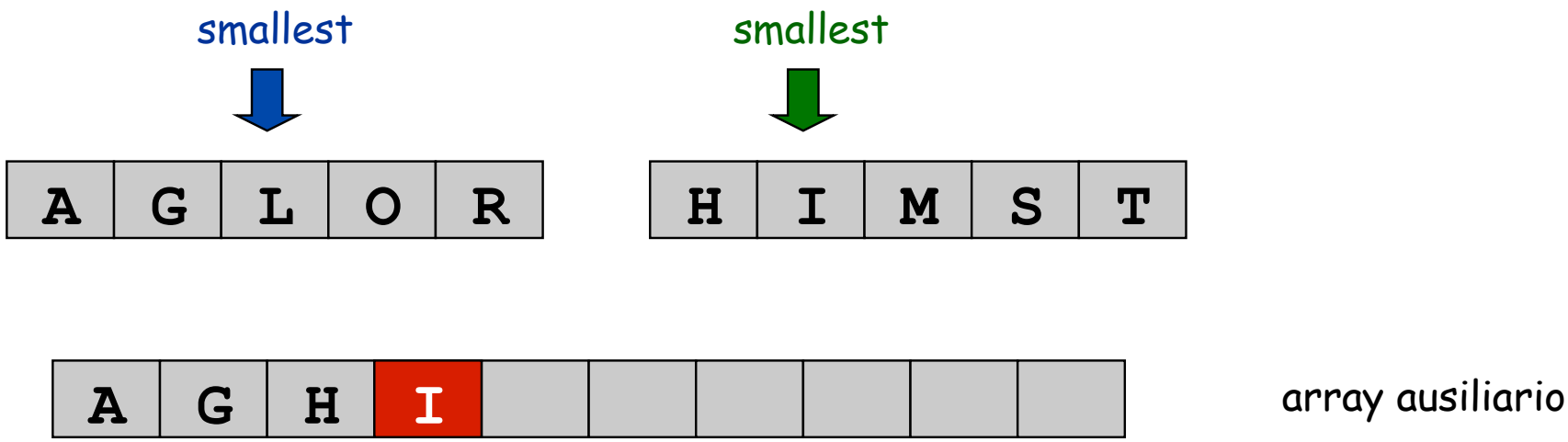
- Mantenere traccia dell' elemento più piccolo in ogni metà ordinata.
- Inserire il più piccolo dei due elementi in un array ausiliario.
- Ripetere fino a terminare gli elementi nelle due metà.



Merge

Merge.

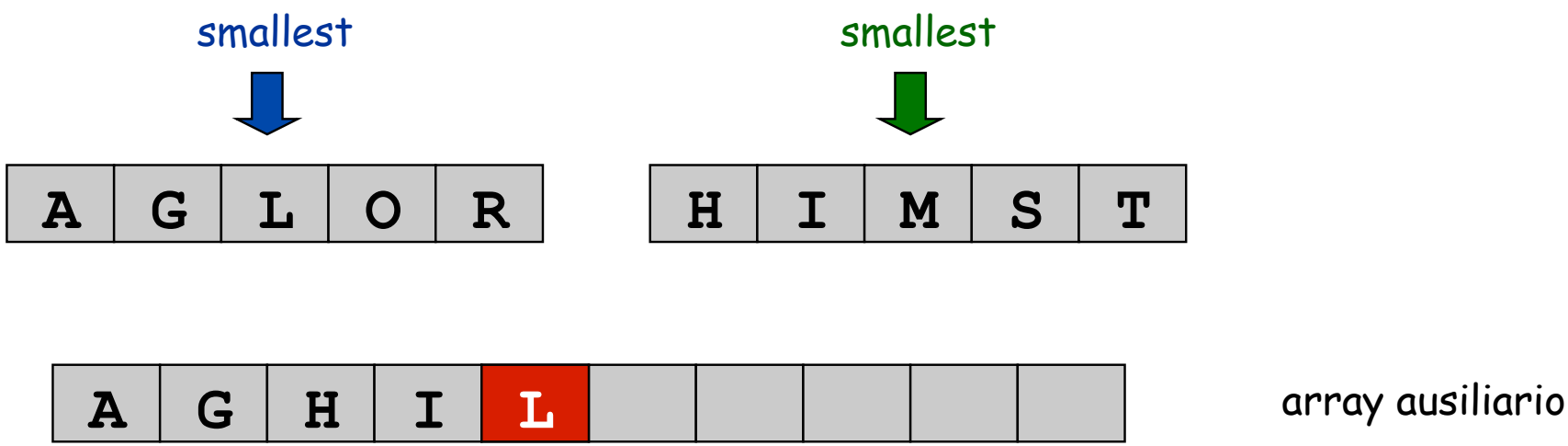
- Mantenere traccia dell' elemento più piccolo in ogni metà ordinata.
- Inserire il più piccolo dei due elementi in un array ausiliario.
- Ripetere fino a terminare gli elementi nelle due metà.



Merge

Merge.

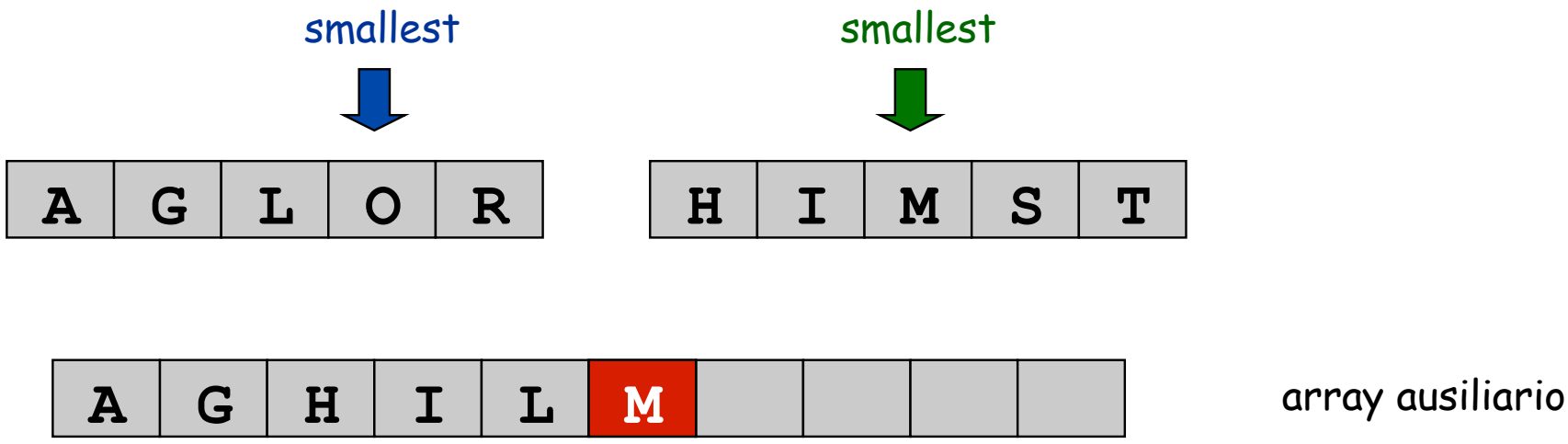
- Mantenere traccia dell' elemento più piccolo in ogni metà ordinata.
- Inserire il più piccolo dei due elementi in un array ausiliario.
- Ripetere fino a terminare gli elementi nelle due metà.



Merge

Merge.

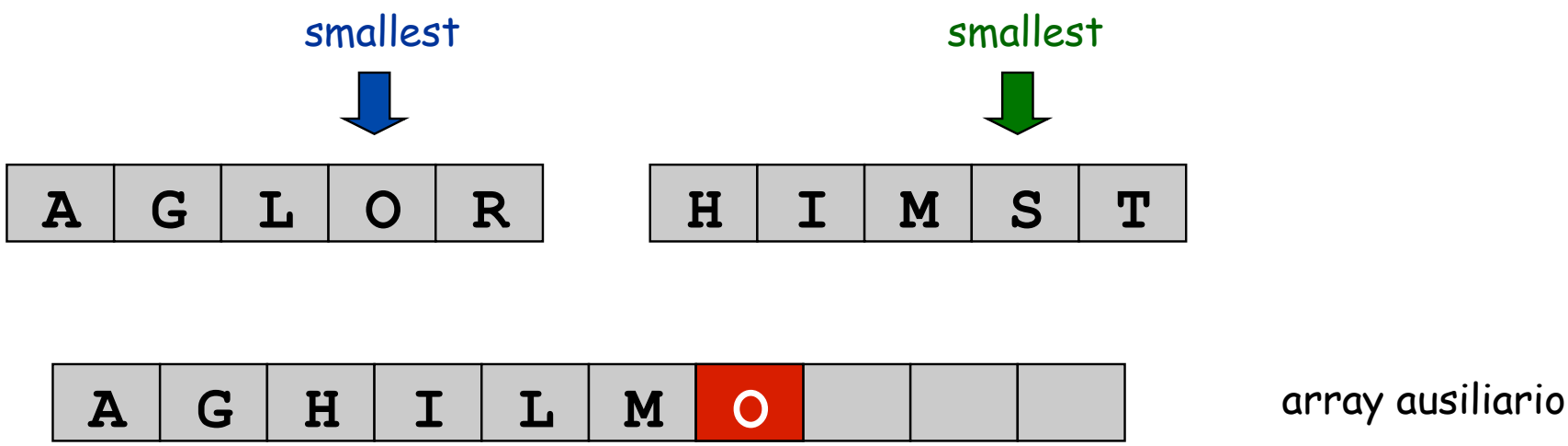
- Mantenere traccia dell' elemento più piccolo in ogni metà ordinata.
- Inserire il più piccolo dei due elementi in un array ausiliario.
- Ripetere fino a terminare gli elementi nelle due metà.



Merge

Merge.

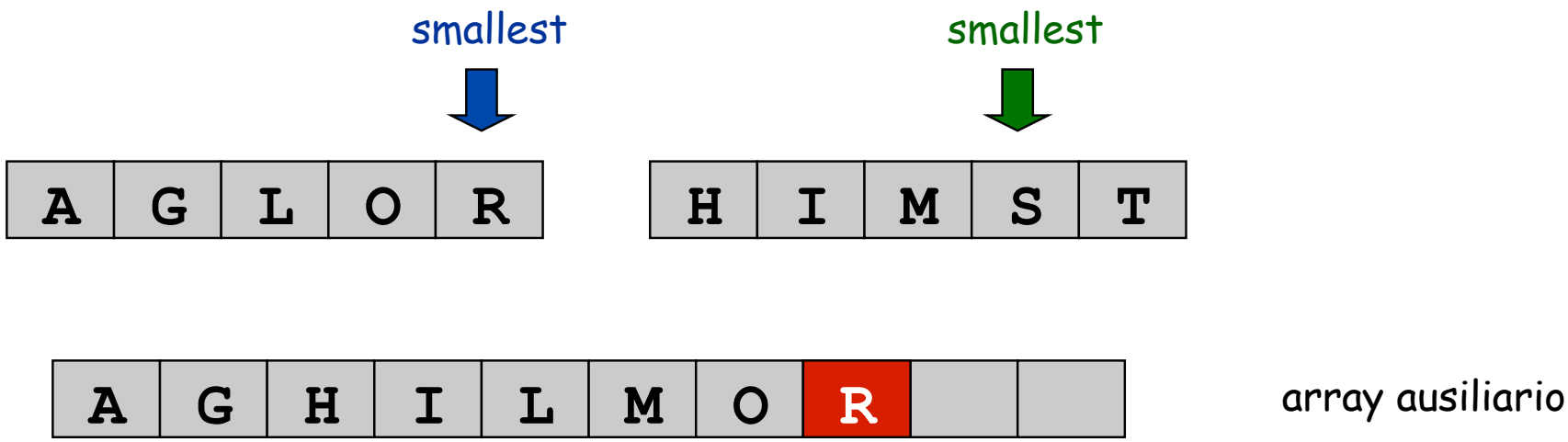
- Mantenere traccia dell' elemento più piccolo in ogni metà ordinata.
- Inserire il più piccolo dei due elementi in un array ausiliario.
- Ripetere fino a terminare gli elementi nelle due metà.



Merge

Merge.

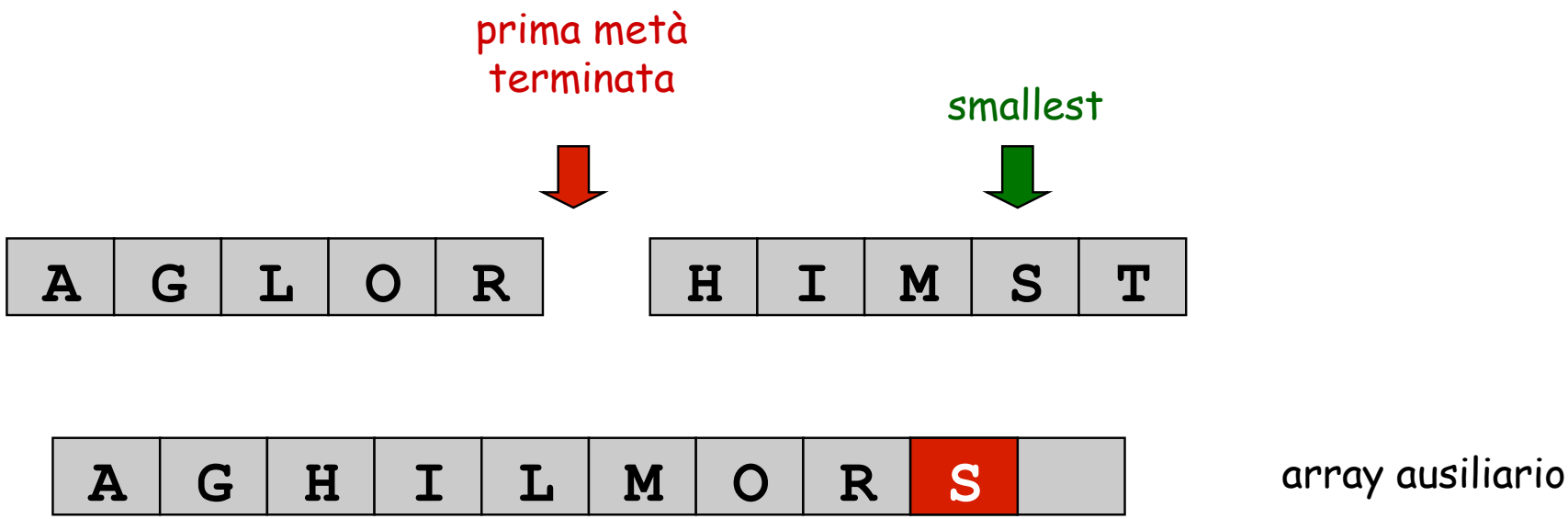
- Mantenere traccia dell' elemento più piccolo in ogni metà ordinata.
- Inserire il più piccolo dei due elementi in un array ausiliario.
- Ripetere fino a terminare gli elementi nelle due metà.



Merge

Merge.

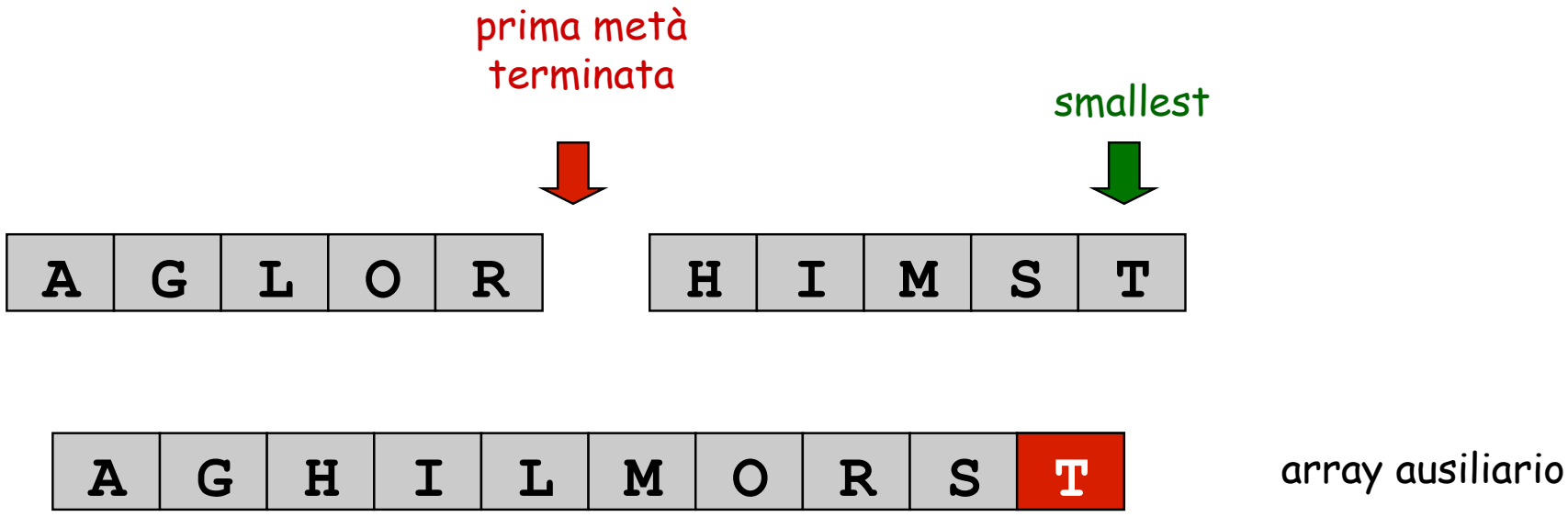
- Mantenere traccia dell' elemento più piccolo in ogni metà ordinata.
- Inserire il più piccolo dei due elementi in un array ausiliario.
- Ripetere fino a terminare gli elementi nelle due metà.



Merge

Merge.

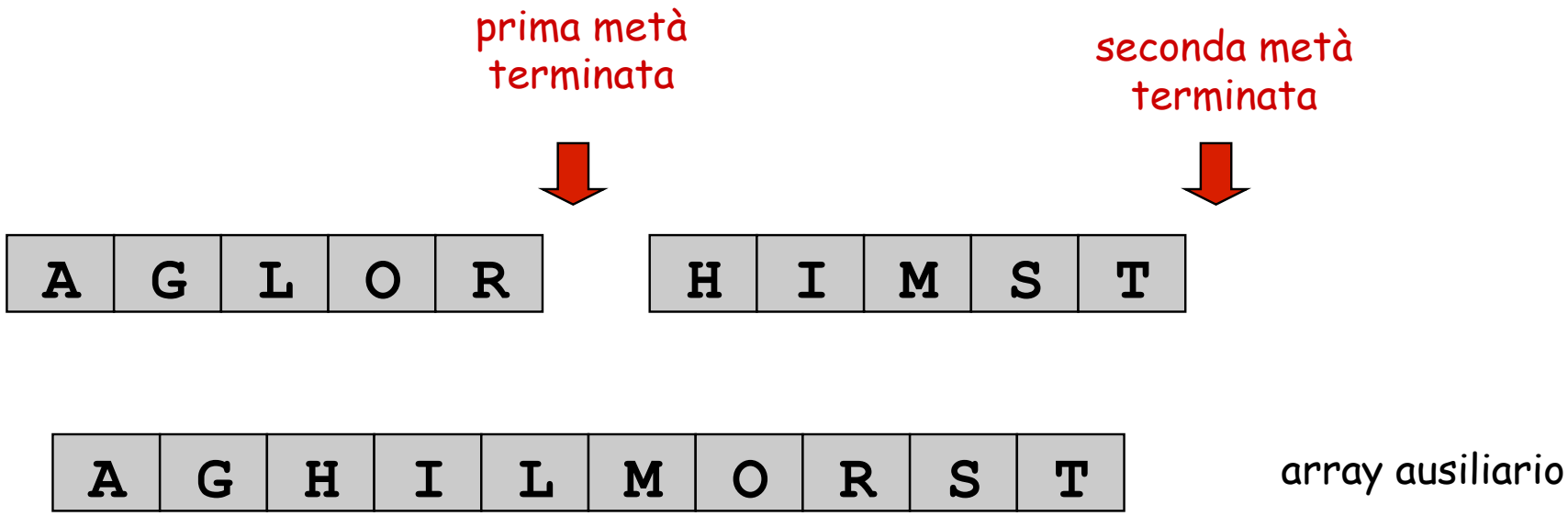
- Mantenere traccia dell' elemento più piccolo in ogni metà ordinata.
- Inserire il più piccolo dei due elementi in un array ausiliario.
- Ripetere fino a terminare gli elementi nelle due metà.



Merge

Merge.

- Mantenere traccia dell' elemento più piccolo in ogni metà ordinata.
- Inserire il più piccolo dei due elementi in un array ausiliario.
- Ripetere fino a terminare gli elementi nelle due metà.

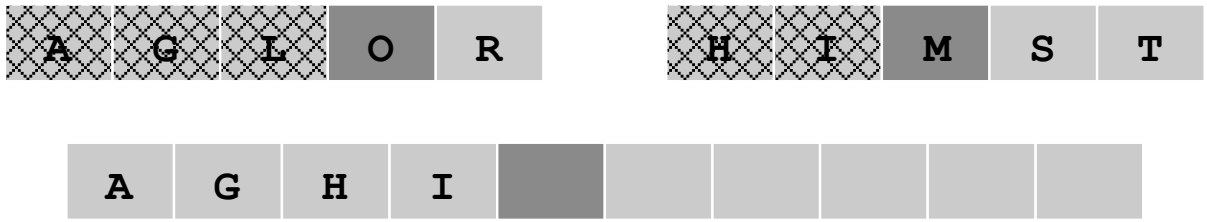


Merge

Merging. Combina due liste ordinate in un'unica lista ordinata.

Come effettuare il merge efficientemente?

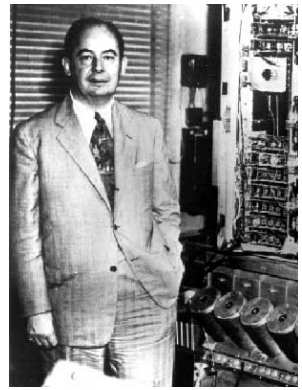
- Numero lineare di confronti.
- Uso di un array temporaneo.



Mergesort

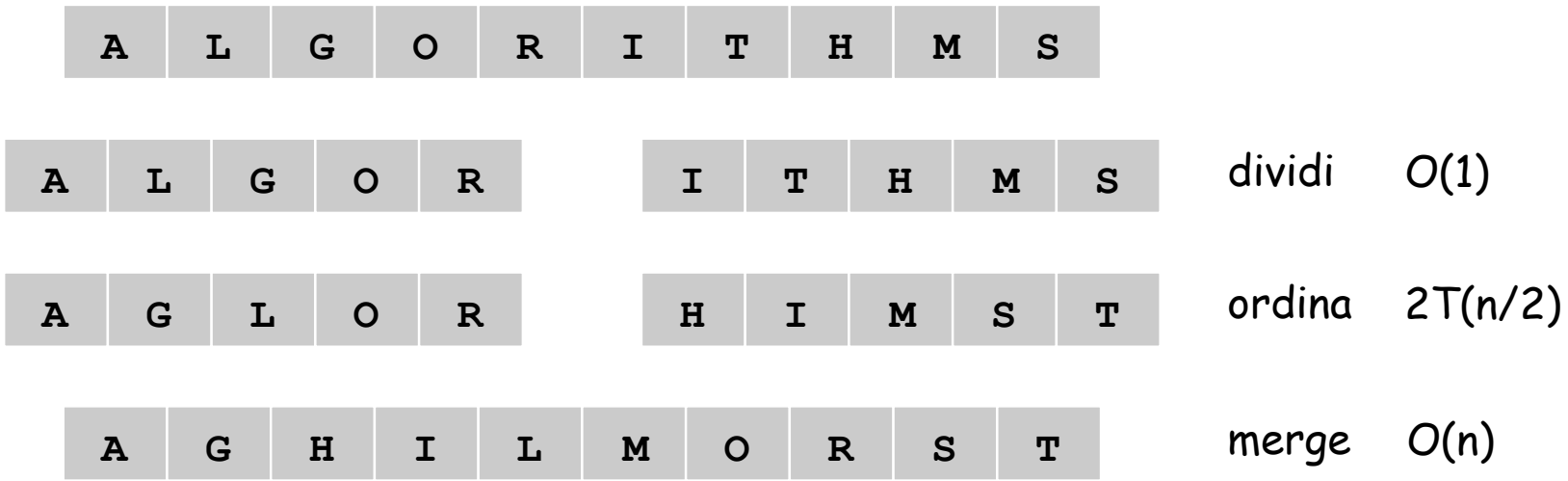
Mergesort.

- Dividere array in due metà.
- Ricorsivamente ordina ogni metà.
- Merge due metà per ottenere tutta la sequenza ordinata.



Jon von Neumann (1945)

Definizione. T(n) = numero di confronti mergesort per input di n elementi.



Una utile relazione di ricorrenza

Definizione. T(n) = numero di confronti mergesort per input n elementi.

Ricorrenza del Mergesort.

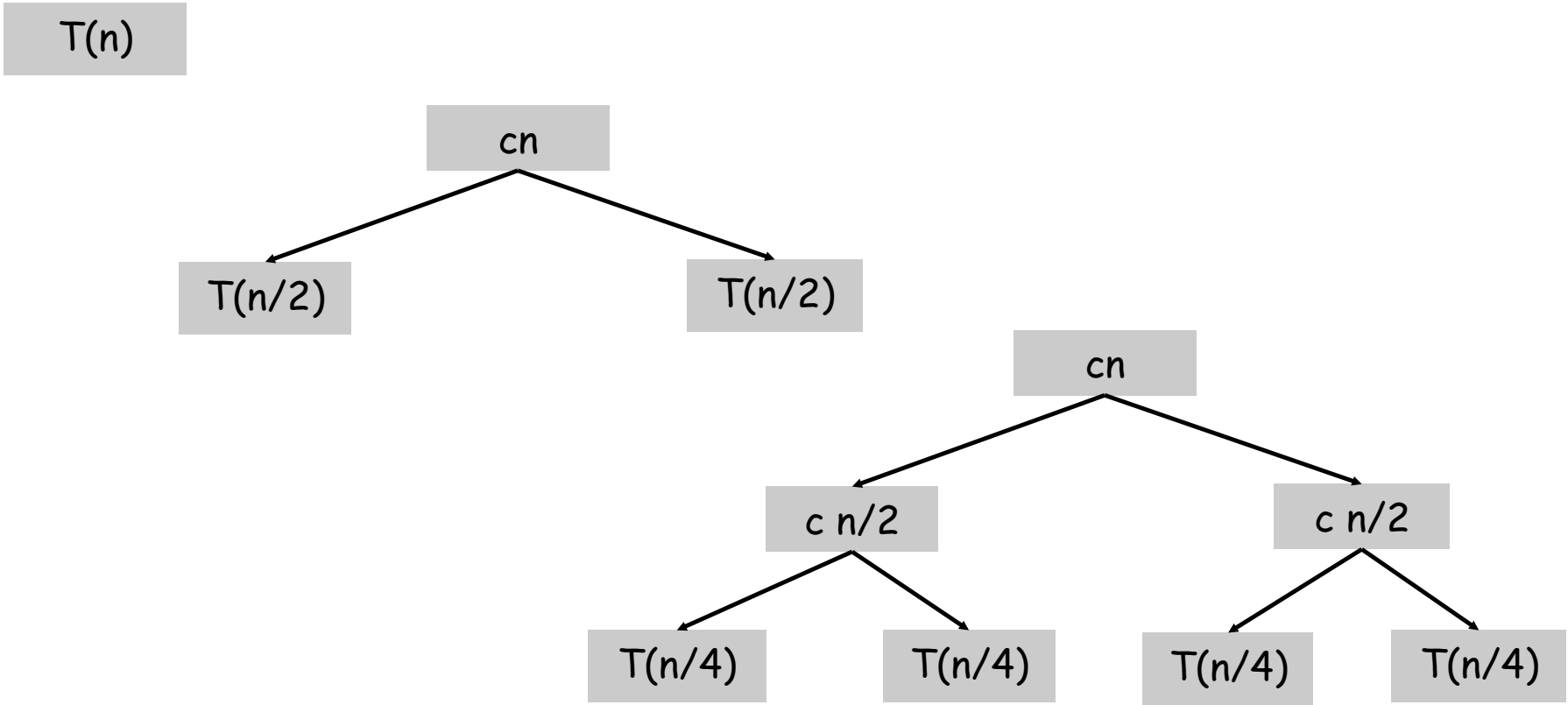
$$T(n) \leq \begin{cases} 0 & \text{se } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{risolvi parte sinistra}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{risolvi parte destra}} + \underbrace{cn}_{\text{merge}} & \text{altrimenti} \end{cases}$$

Soluzione. $T(n) = O(n \log_2 n)$.

Varie prove. Descriviamo vari modi per risolvere questa ricorrenza.
Iniziamente assumiamo che n è una potenza di 2 e sostituiamo ≤ con =.

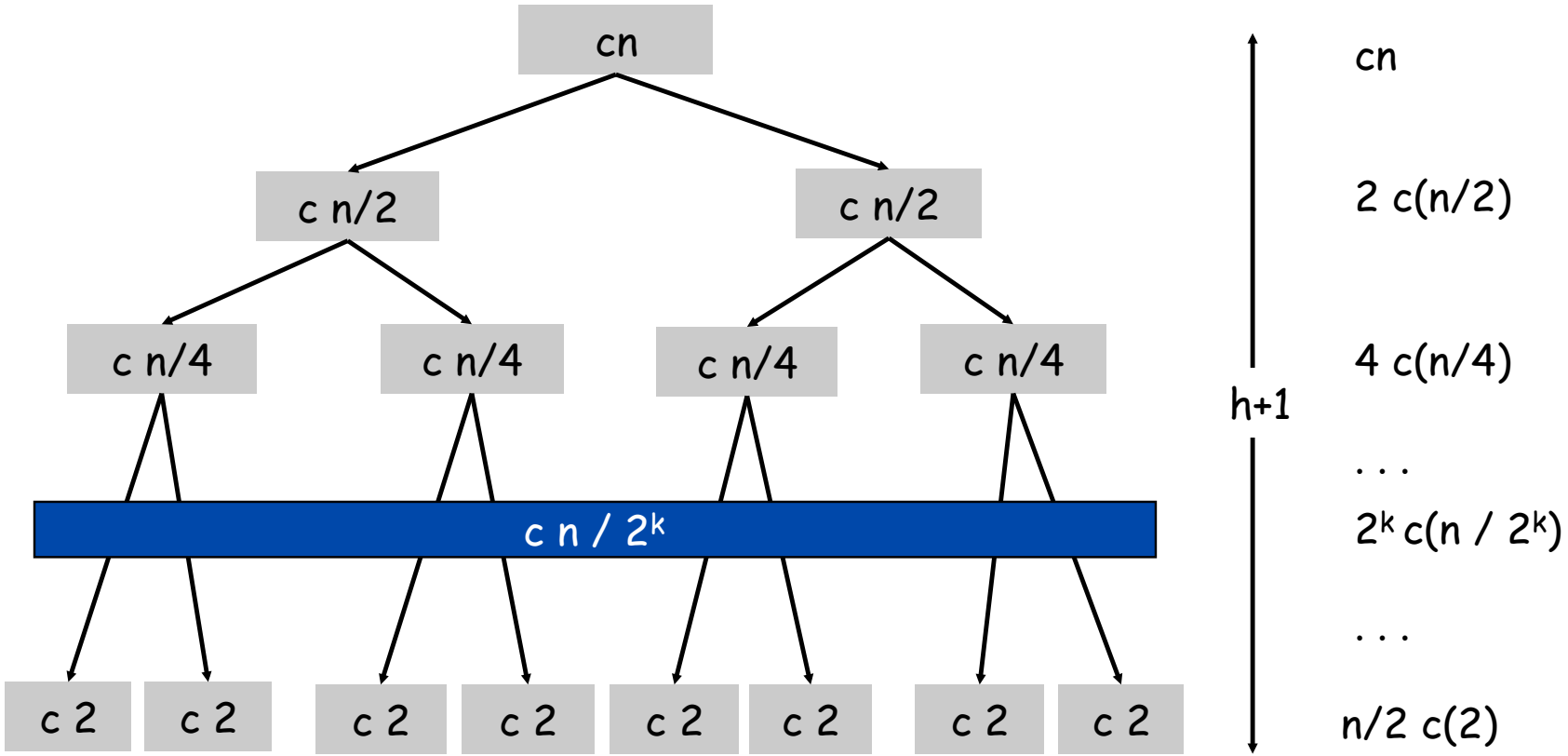
Prova con Albero di Ricorsione

$$T(n) \leq \begin{cases} 0 & \text{se } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{risolvi parte sinistra}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{risolvi parte destra}} + \underbrace{cn}_{\text{merge}} & \text{altrimenti} \end{cases}$$



Prova con Albero di Ricorsione

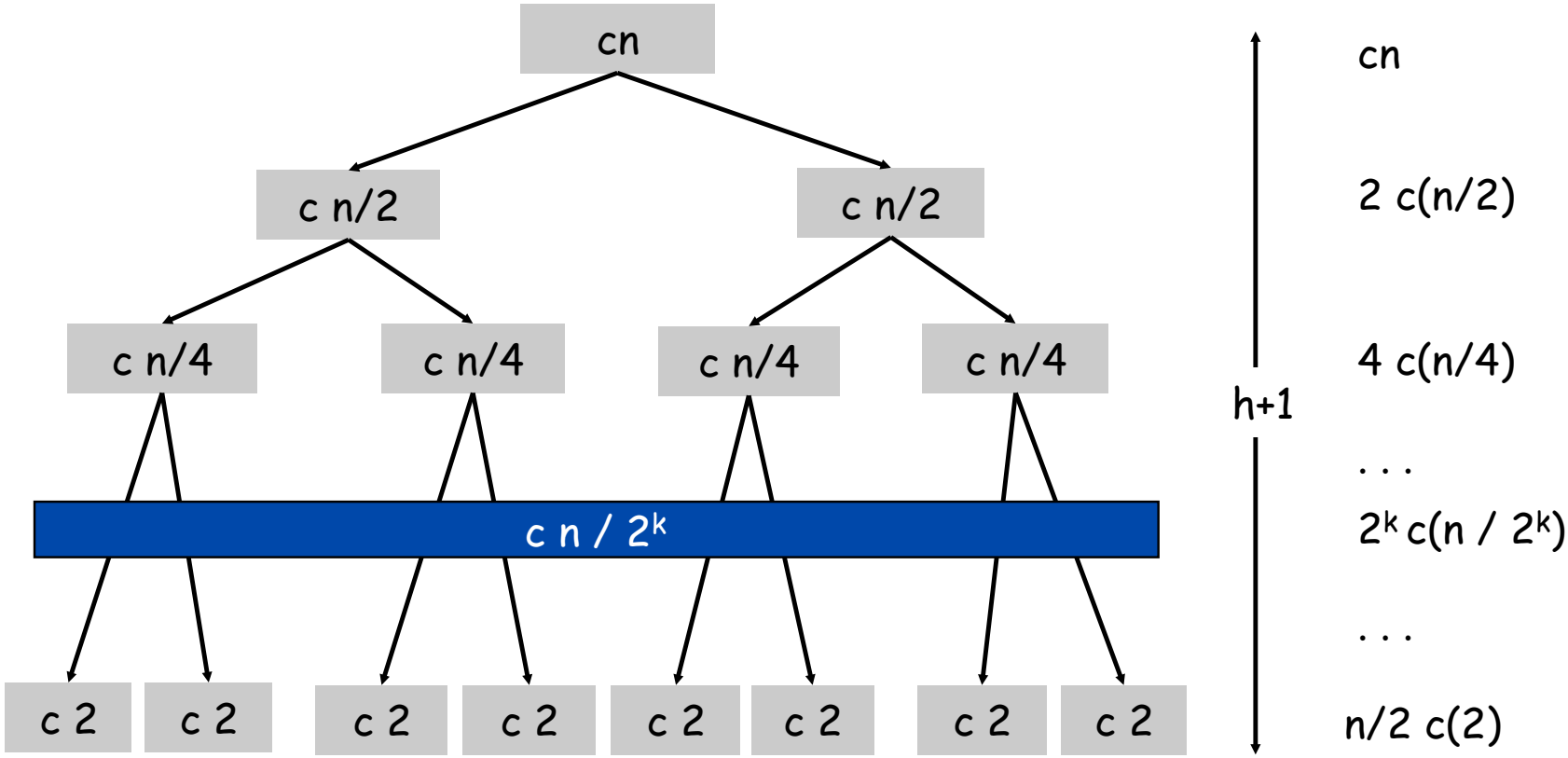
$$T(n) \leq \begin{cases} 0 & \text{se } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{risolvi parte sinistra}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{risolvi parte destra}} + \underbrace{cn}_{\text{merge}} & \text{altrimenti} \end{cases}$$



Altezza albero h?

Prova con Albero di Ricorsione

$$T(n) \leq \begin{cases} 0 & \text{se } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{risolvi parte sinistra}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{risolvi parte destra}} + \underbrace{cn}_{\text{merge}} & \text{altrimenti} \end{cases}$$



Altezza albero h? $n / 2^h = 2$ cioè $h = \log_2 n - 1$

Prova con sequenza telescopica

Claim. $T(n) = c\, n \log_2 n$. (assumiamo n potenza di 2)

$$T(n) \le \begin{cases} 0 & \text{se } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{risolvi parte sinistra}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{risolvi parte destra}} + \underbrace{cn}_{\text{merge}} & \text{altrimenti} \end{cases}$$

Prova. Per n > 1:

$$\begin{aligned} \frac{T(n)}{n} &= \frac{2T(n/2)}{n} + c \\ &= \frac{T(n/2)}{n/2} + c \\ &= \frac{T(n/4)}{n/4} + c + c \\ &\cdots \\ &= \frac{T(n/n)}{n/n} + \underbrace{c + \cdots + c}_{\log_2 n} \\ &= c \log_2 n \end{aligned}$$

$$\frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + c$$

Prova per induzione

Claim. T(n) = n $\log_2$ n. (assumiamo n potenza di 2)

$$T(n) \le \begin{cases} 0 & \text{se } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{risolvi parte sinistra}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{risolvi parte destra}} + \underbrace{cn}_{\text{merge}} & \text{altrimenti} \end{cases}$$

Prova. (per induzione su n)

- Caso base: n = 1.
- Ipotesi induttiva: T(n) = c n $\log_2$ n.
- Obiettivo: mostrare che T(2n) = c 2n $\log_2$ (2n).

$$\begin{aligned} T(2n) &= 2T(n) + c2n \\ &= c2n\log_2 n + c2n \\ &= c2n(\log_2(2n) - 1) + c2n \\ &= c2n\log_2(2n) \end{aligned}$$

Analisi della ricorrenza del Mergesort

Claim. $T(n) \le c\, n \lceil \log n \rceil$. (logaritmo in base 2, cioè $\log = \log_2$)

$$T(n) \le \begin{cases} 0 & \text{se } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{risolvi parte sinistra}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{risolvi parte destra}} + \underbrace{cn}_{\text{merge}} & \text{altrimenti} \end{cases}$$

Prova. (per induzione su n)

- Caso base: n = 1.
- Definiamo $n_1 = \lfloor n / 2 \rfloor$, $n_2 = \lceil n / 2 \rceil$.
- Passo induzione: assumiamo che sia vero per 1, 2, ... , n–1.

$$\begin{aligned} T(n) &\le T(n_1) + T(n_2) + cn \\ &\le cn_1 \lceil \log n_1 \rceil + cn_2 \lceil \log n_2 \rceil + cn \\ &\le cn_1 \lceil \log n_2 \rceil + cn_2 \lceil \log n_2 \rceil + cn \\ &= cn \lceil \log n_2 \rceil + cn \\ &\le cn(\lceil \log n \rceil - 1) + cn \\ &= cn \lceil \log n \rceil \end{aligned}$$

$$\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\le \lceil 2^{\lceil \log n \rceil} / 2 \rceil \\ &= 2^{\lceil \log n \rceil} / 2 \end{aligned}$$

$$\Rightarrow \log n_2 \le \lceil \log n \rceil - 1$$

5.3 Counting Inversions

Conteggio Inversioni

Sito musica cerca di fare un match delle preferenze musicali di utenti.

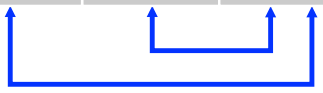
- Ognuno classifica n brani.
- Sito musica consulta il data base per cercare liste di preferenze simili di utenti.

Metrica di similarità: numero di inversioni tra due liste di preferenza.

- Primo rank: 1, 2, …, n.
- Secondo rank: $a_1, a_2, \ldots, a_n$.
- Brano i and j invertiti se i < j, ma $a_i > a_j$.

Brani

	A	B	C	D	E
io	1	2	3	4	5
tu	1	3	4	2	5



Inversioni

3-2, 4-2

Algoritmo "Brute force": controlla tutte le $\Theta(n^2)$ coppie i e j.

Applicazioni

Applicazioni.

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).

Conteggio Inversioni

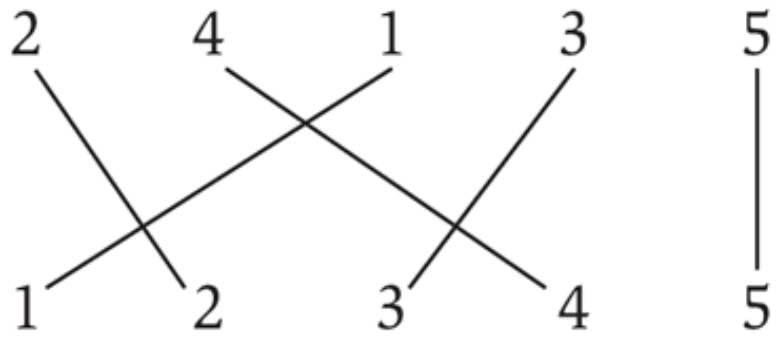


Figure 5.4 Counting the number of inversions in the sequence 2, 4, 1, 3, 5. Each crossing pair of line segments corresponds to one pair that is in the opposite order in the input list and the ascending list—in other words, an inversion.

Inversioni

2-1, 4-1, 4-3

Conteggio Inversioni: Divide-and-Conquer

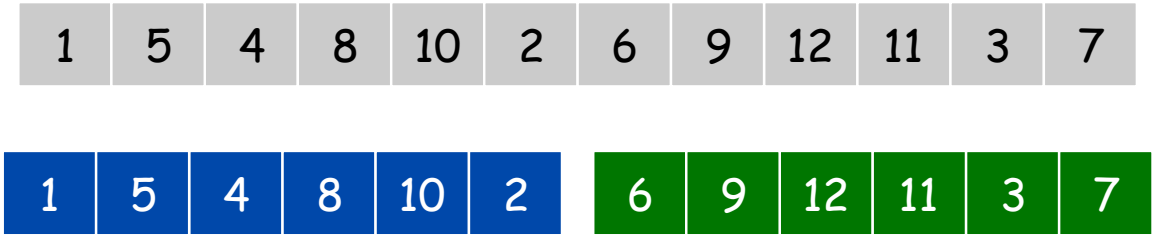
Divide-and-conquer.

1	5	4	8	10	2	6	9	12	11	3	7

Conteggio Inversioni: Divide-and-Conquer

Divide-and-conquer.

- Divide: dividi lista in due parti.

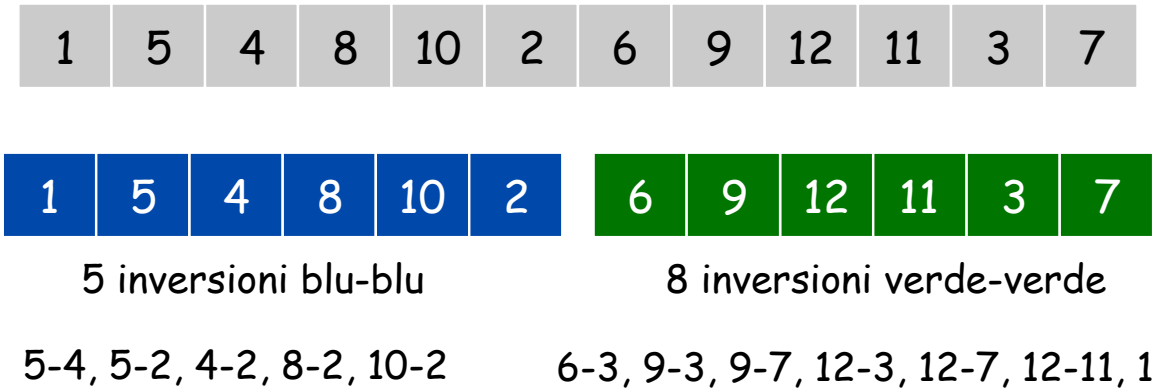


Divide: $O(1)$.

Conteggio Inversioni: Divide-and-Conquer

Divide-and-conquer.

- Divide: dividi lista in due parti.
- Conquer: conta ricorsivamente le inversioni in ogni parte.



Divide: $O(1)$.

Conquer: $2\,T(n / 2)$

Conteggio Inversioni: Divide-and-Conquer

Divide-and-conquer.

- Divide: dividi lista in due parti.
- Conquer: conta ricorsivamente le inversioni in ogni parte.
- Combine: conta le inversioni dove a_i e a_j sono in parti diverse, e restituisci la somma dele tre quantità.

Divide: $O(1)$.

Conquer: 2 T(n / 2)

5 inversioni blu-blu

8 inversioni verde-verde

9 inversioni blu-verde
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Combine: ???

Total = 5 + 8 + 9 = 22.

Conteggio Inversioni: Combine

Combine: conteggio inversioni blu-verde

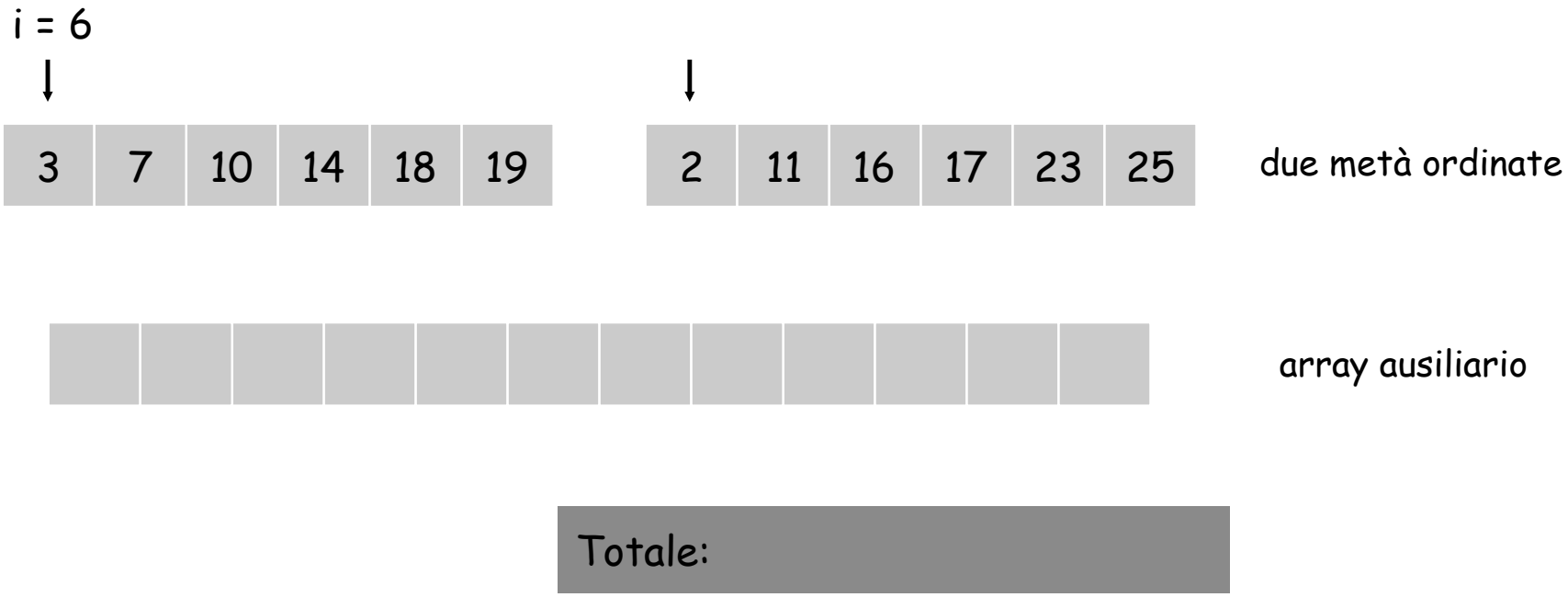
- Assumiamo che ogni metà è ordinata.
- Contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Merge due parti ordinate in una sola sequenza ordinata.

per mantenere l' invariante dell' ordinamento

Merge and Count

Passo di "merge and count".

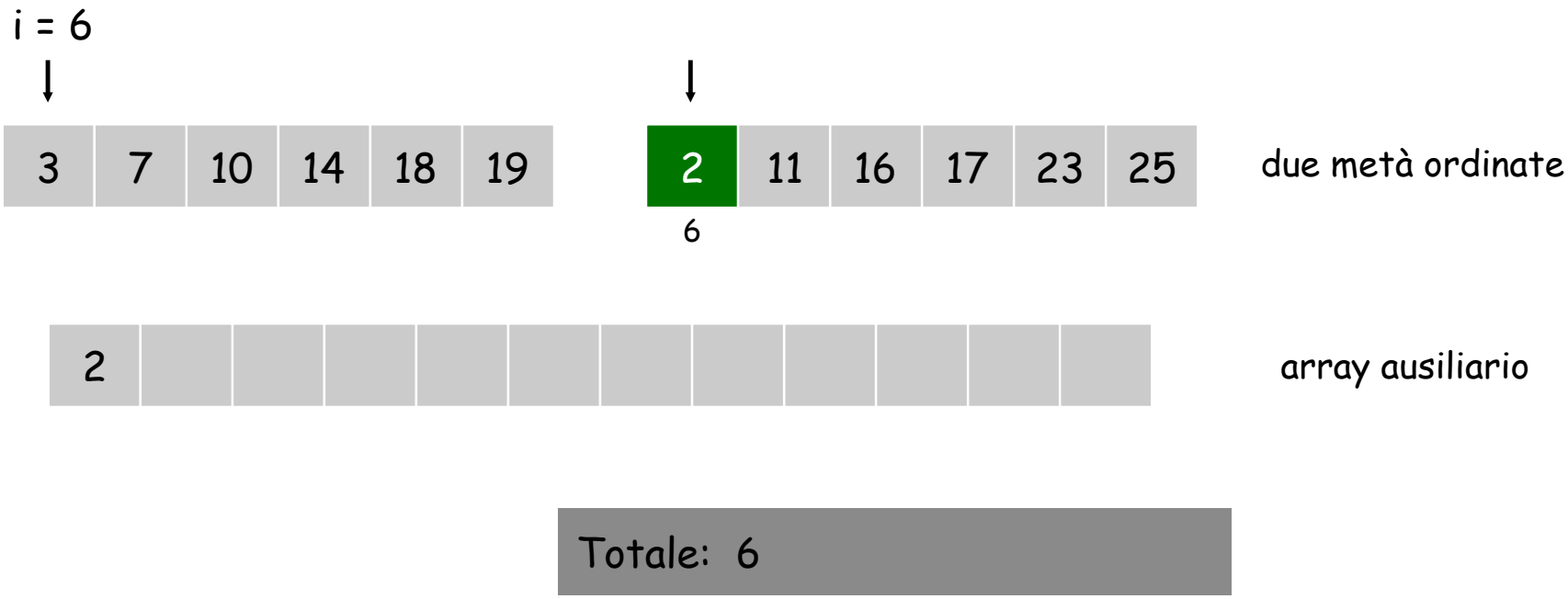
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

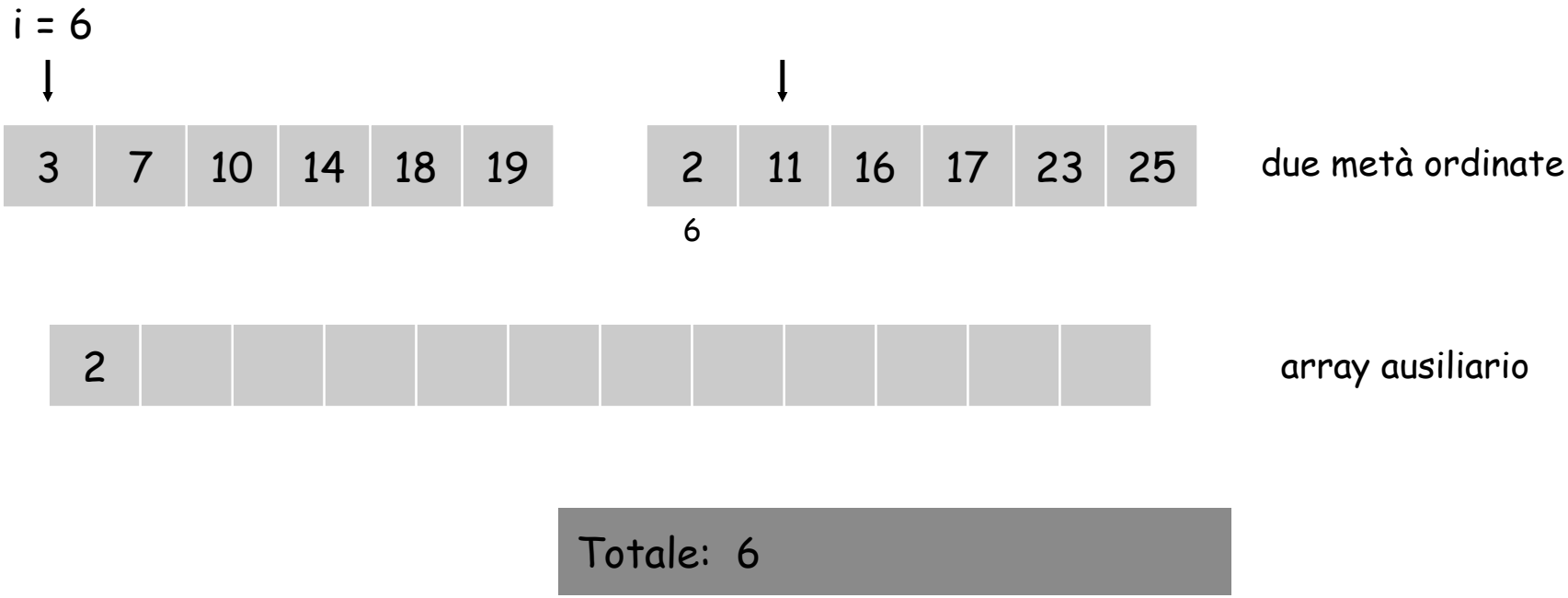
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

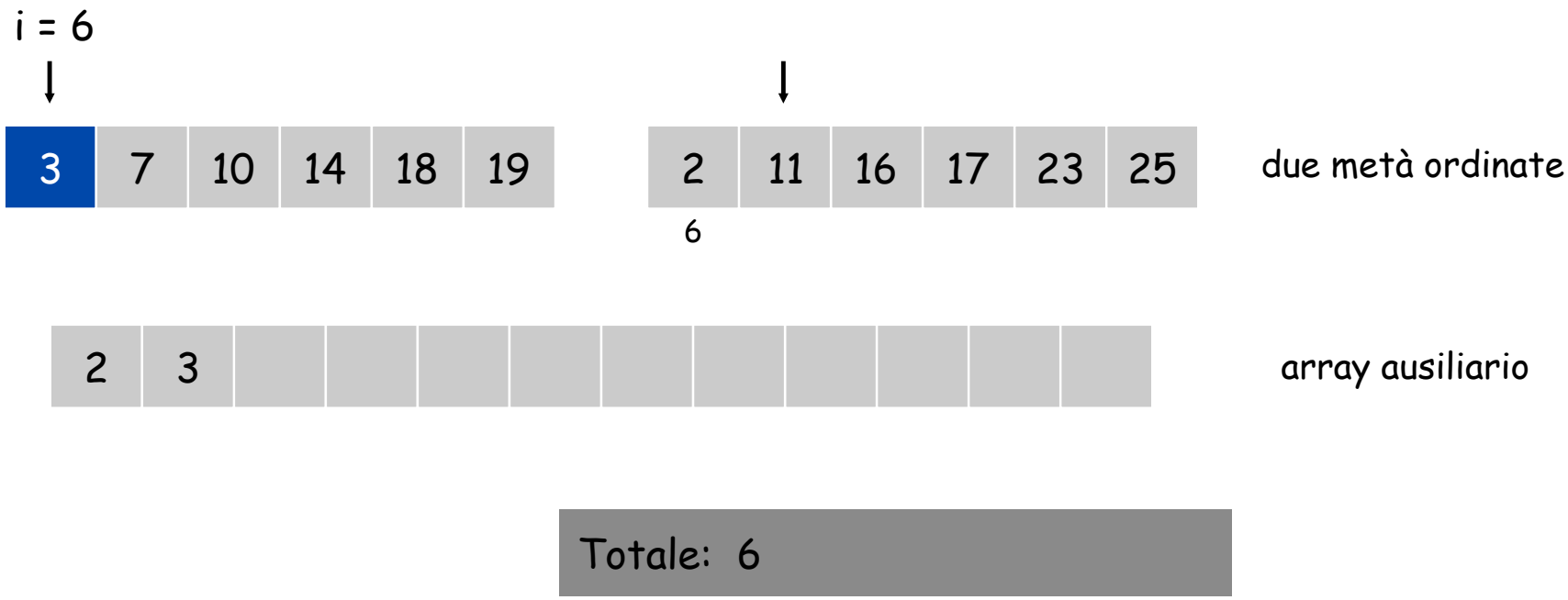
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

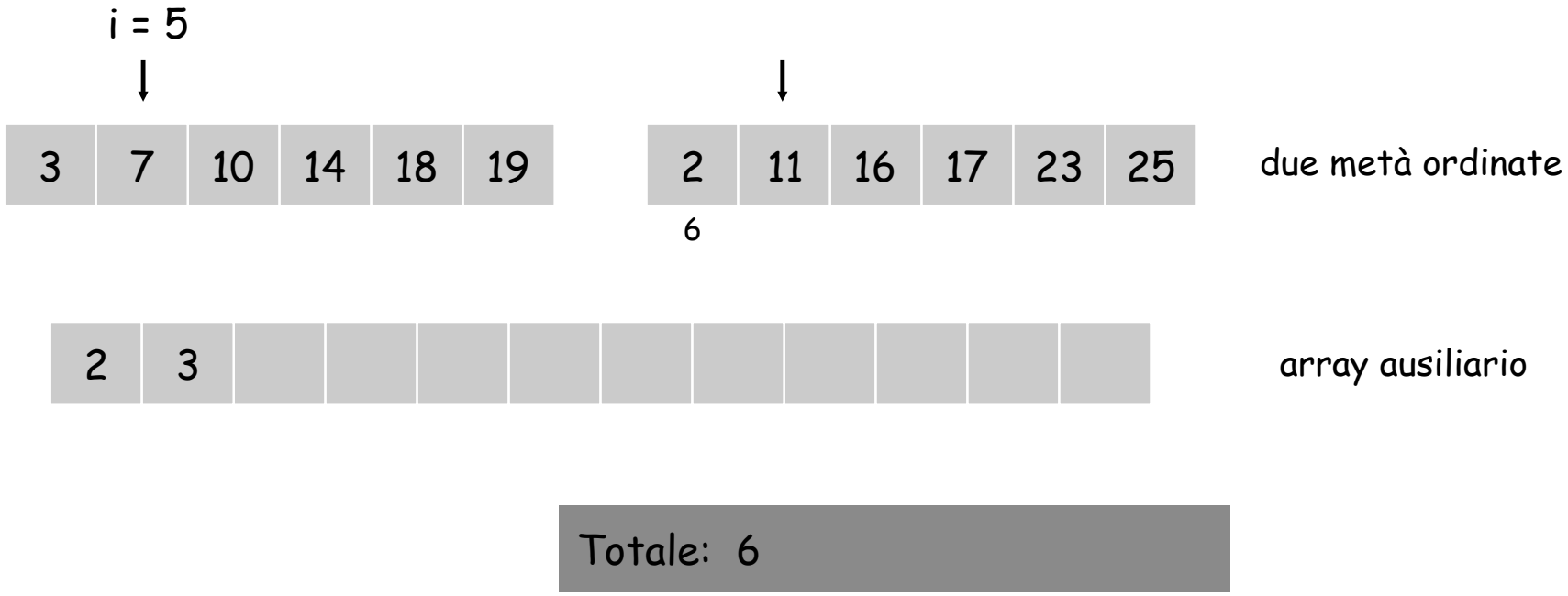
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

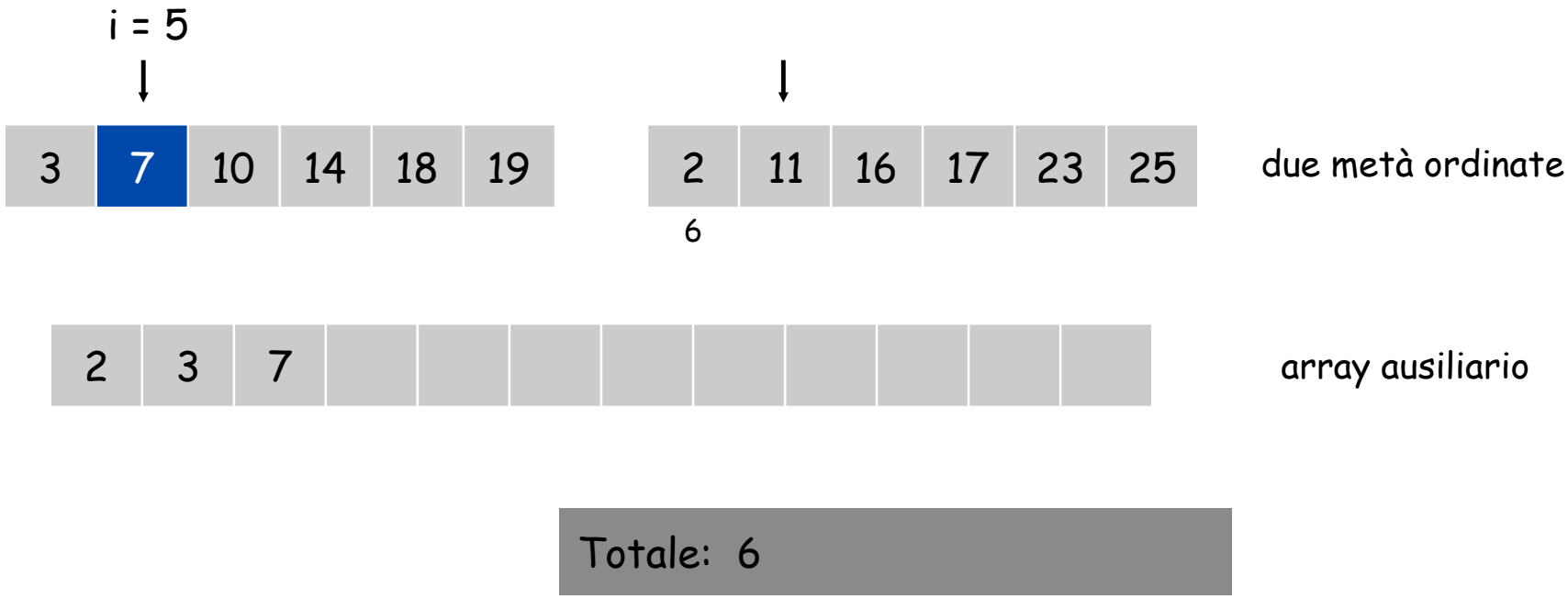
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

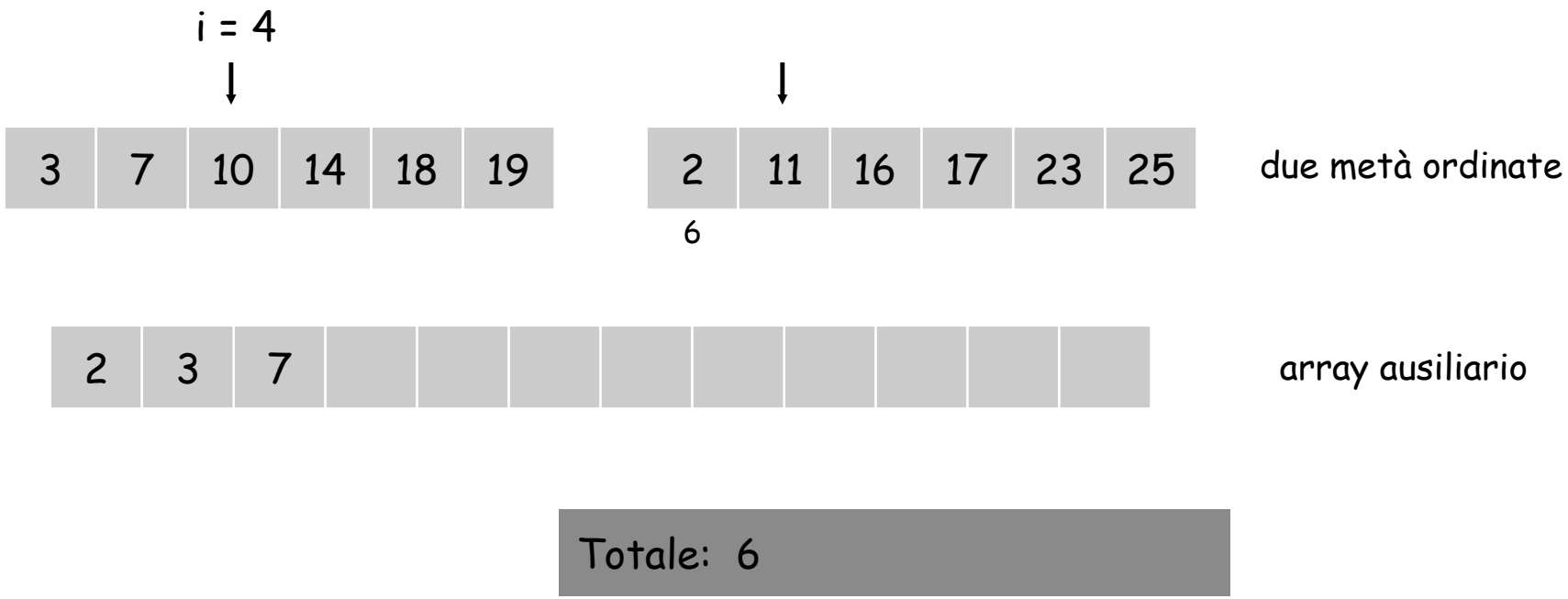
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

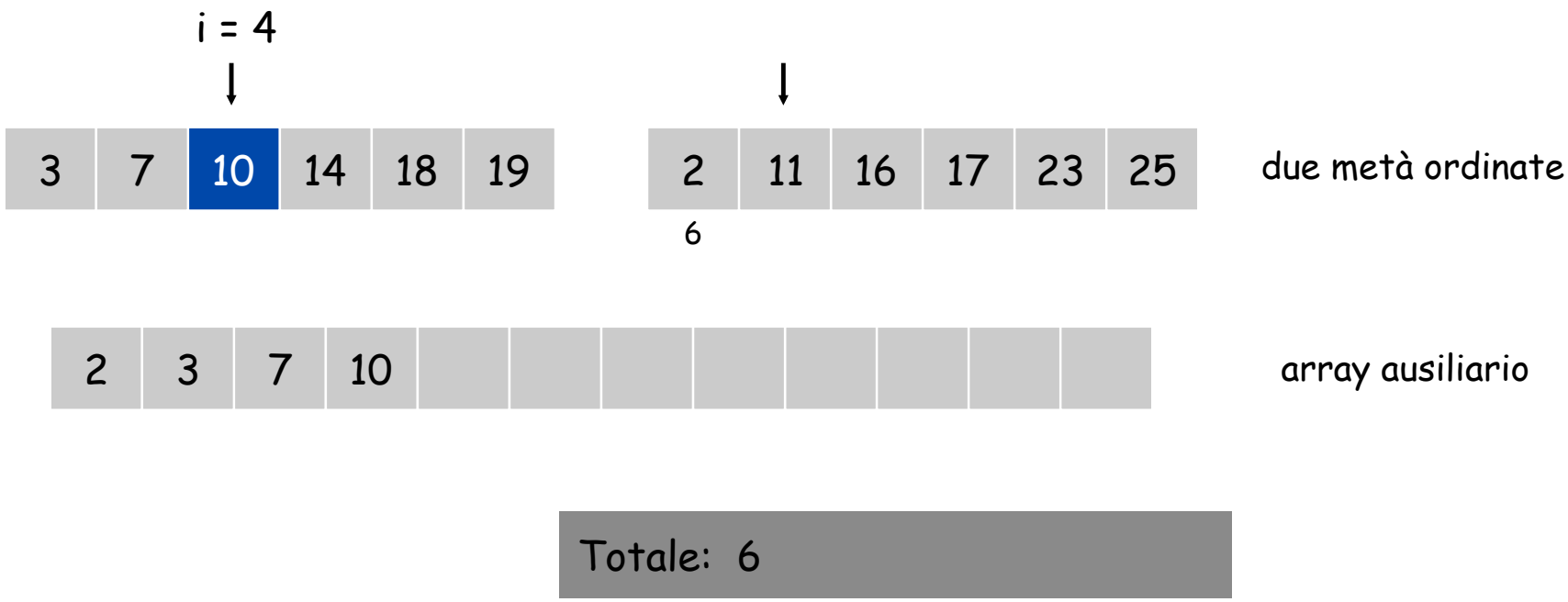
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

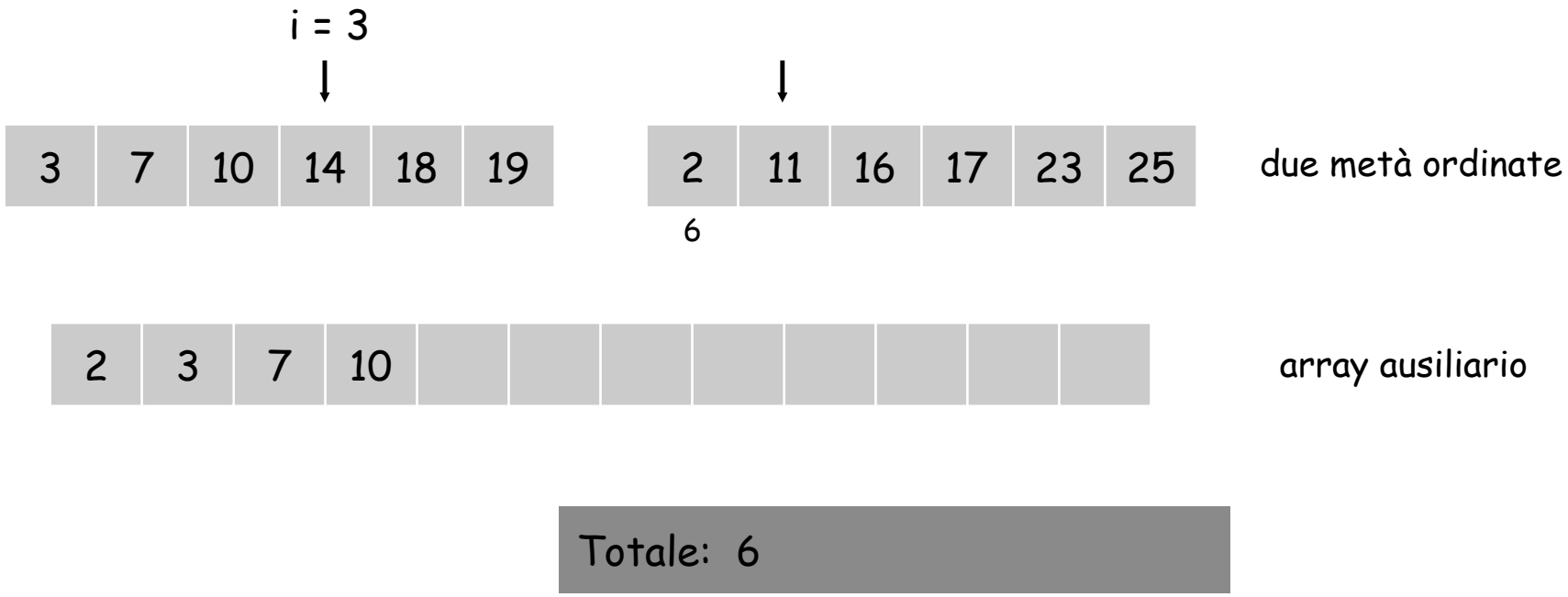
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

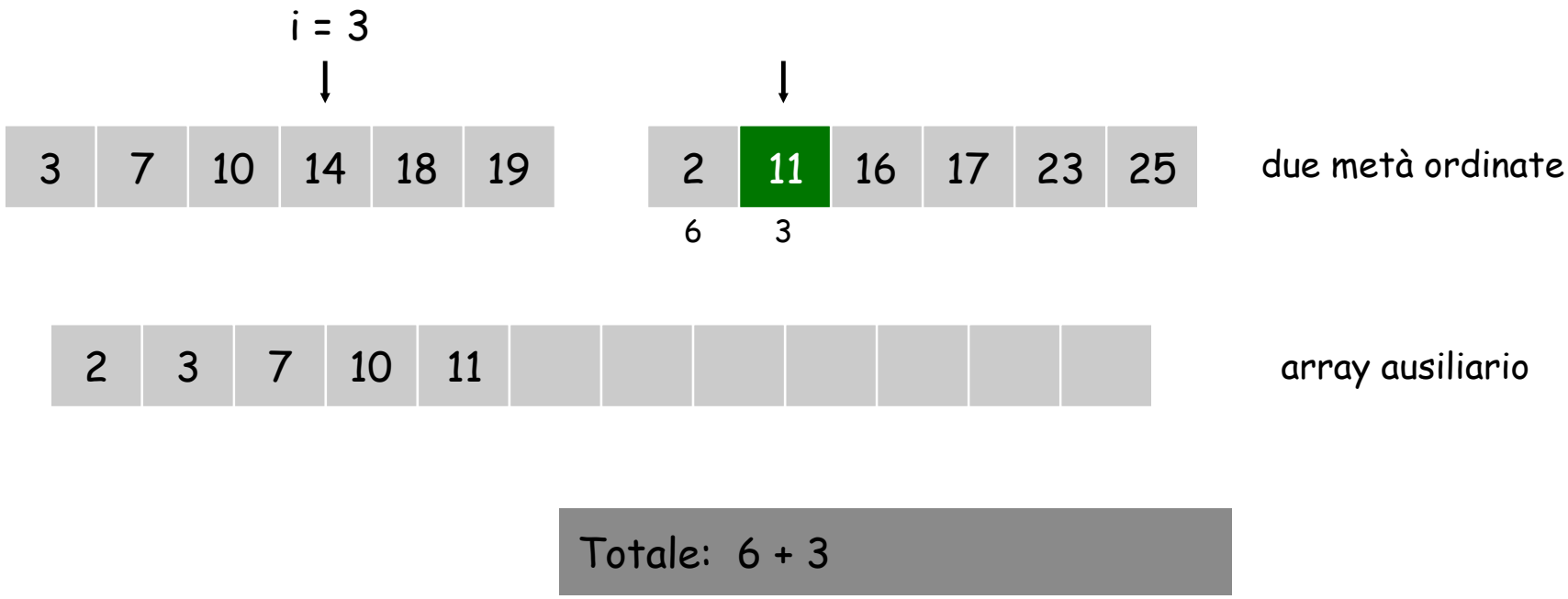
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

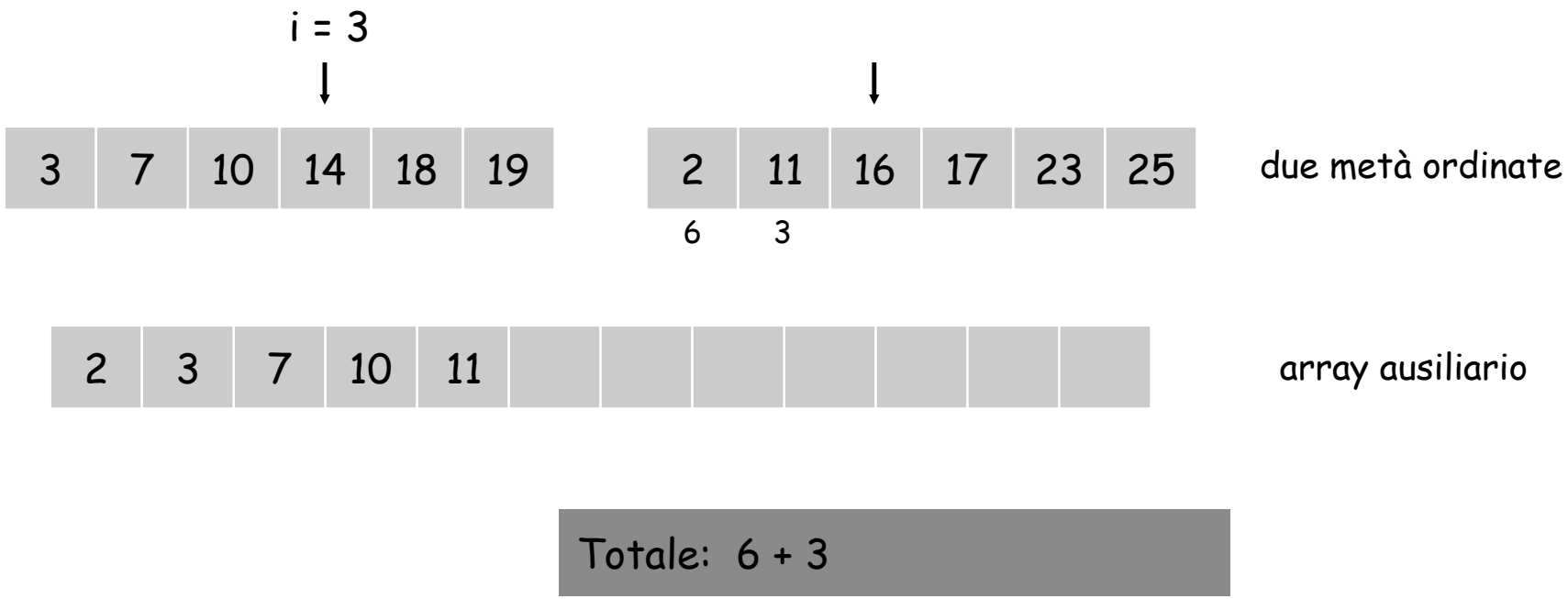
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

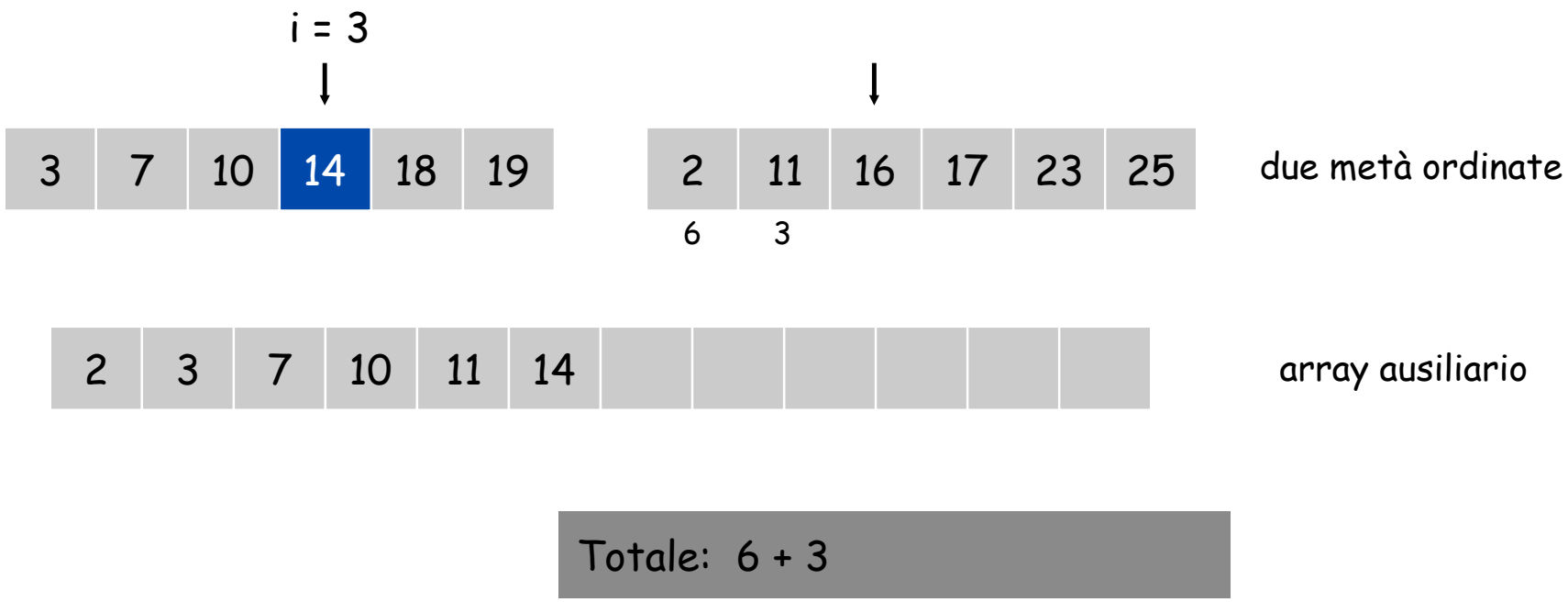
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

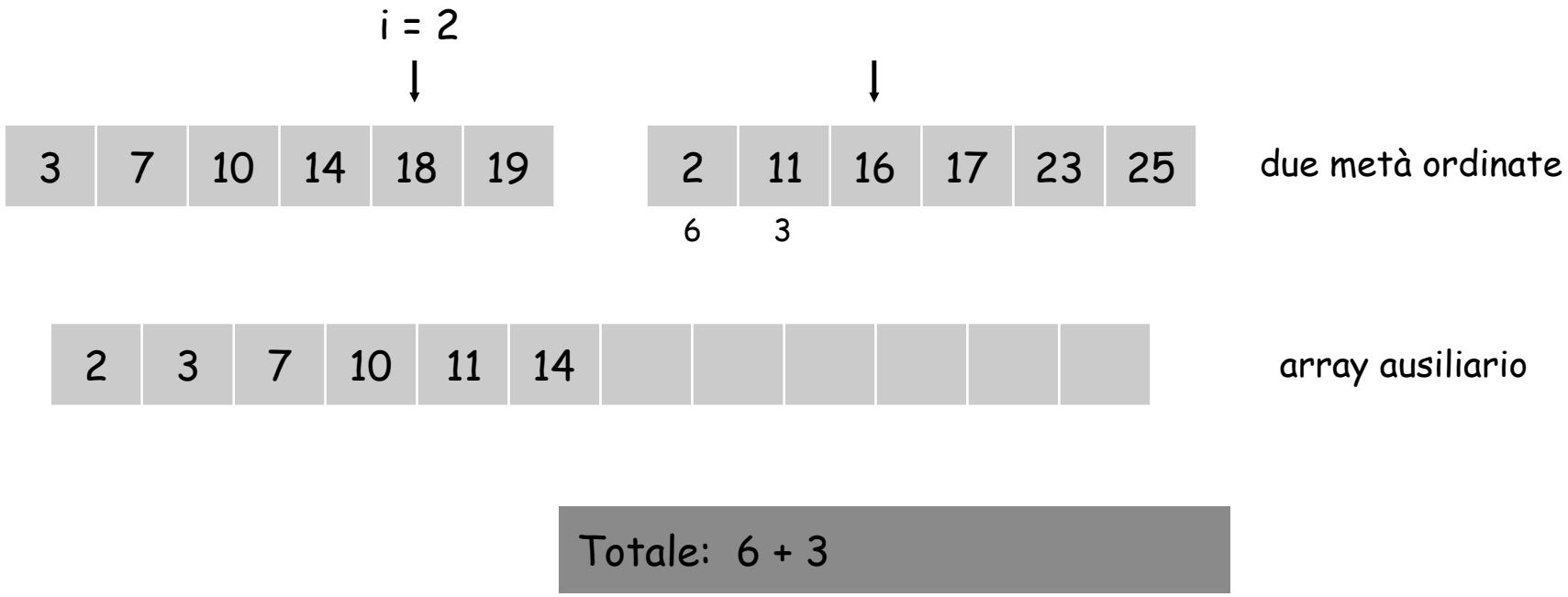
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

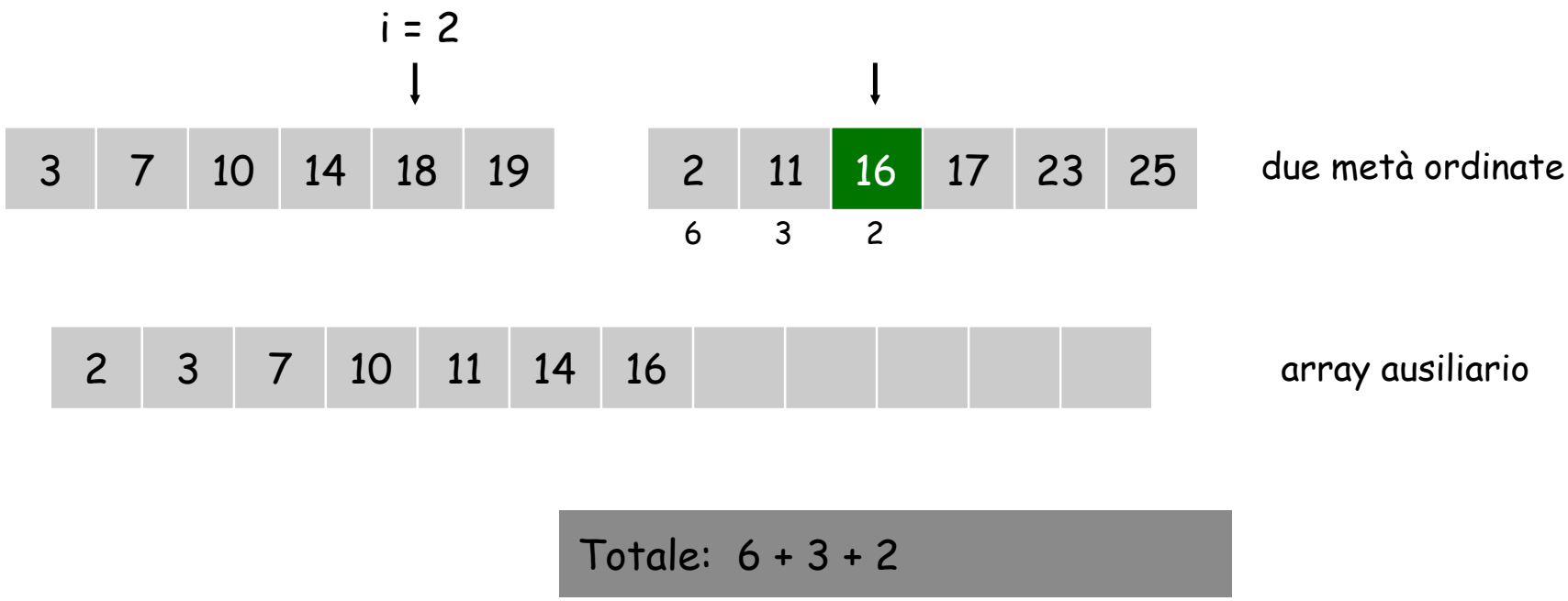
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

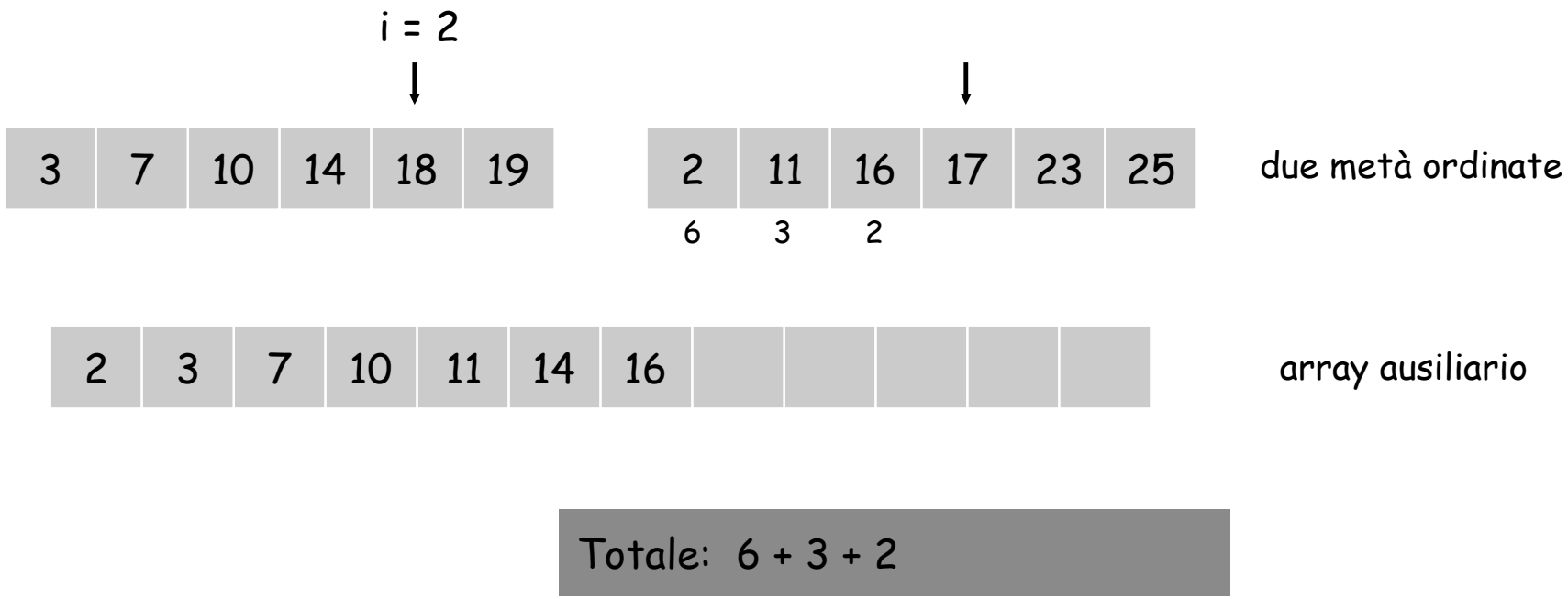
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

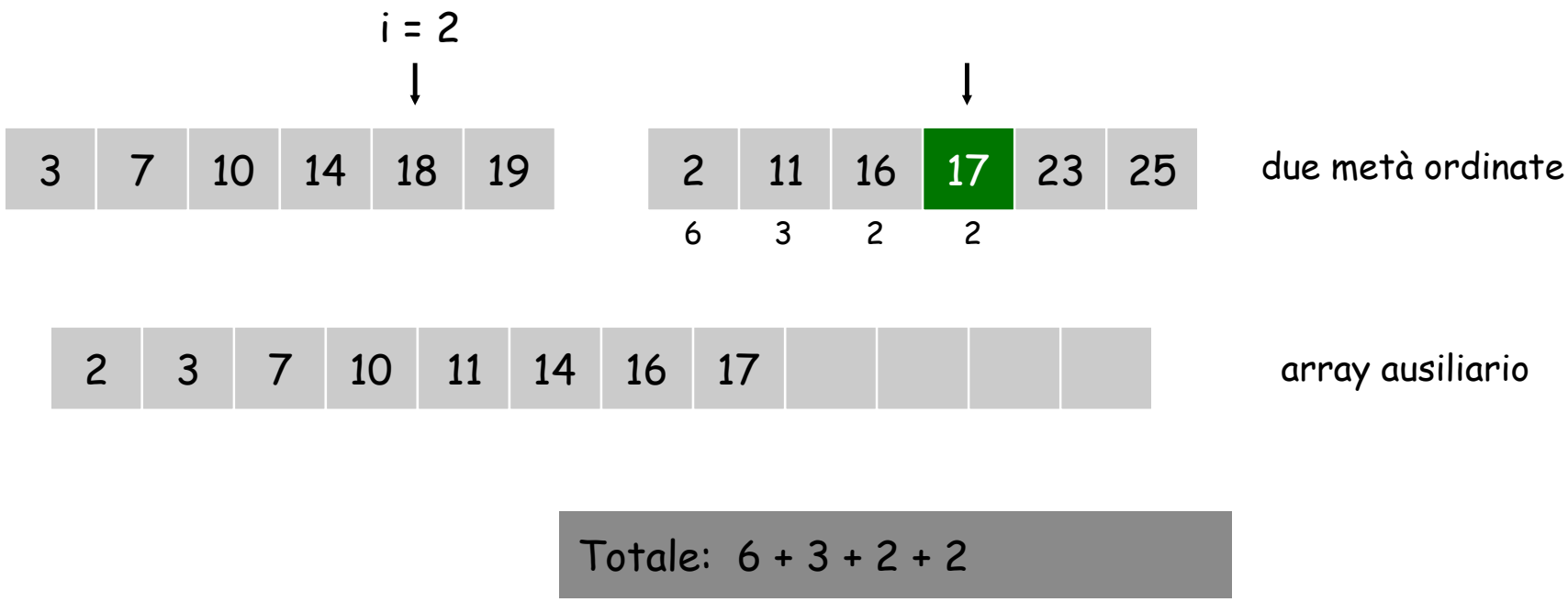
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

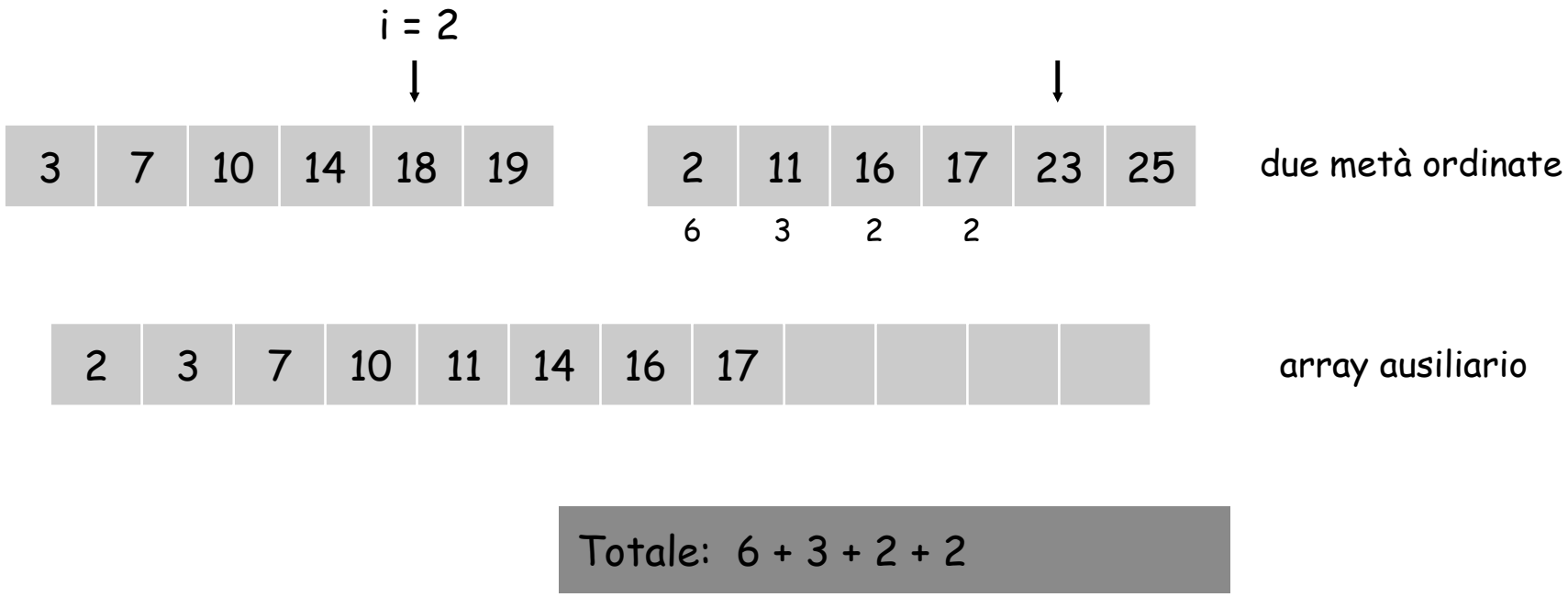
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

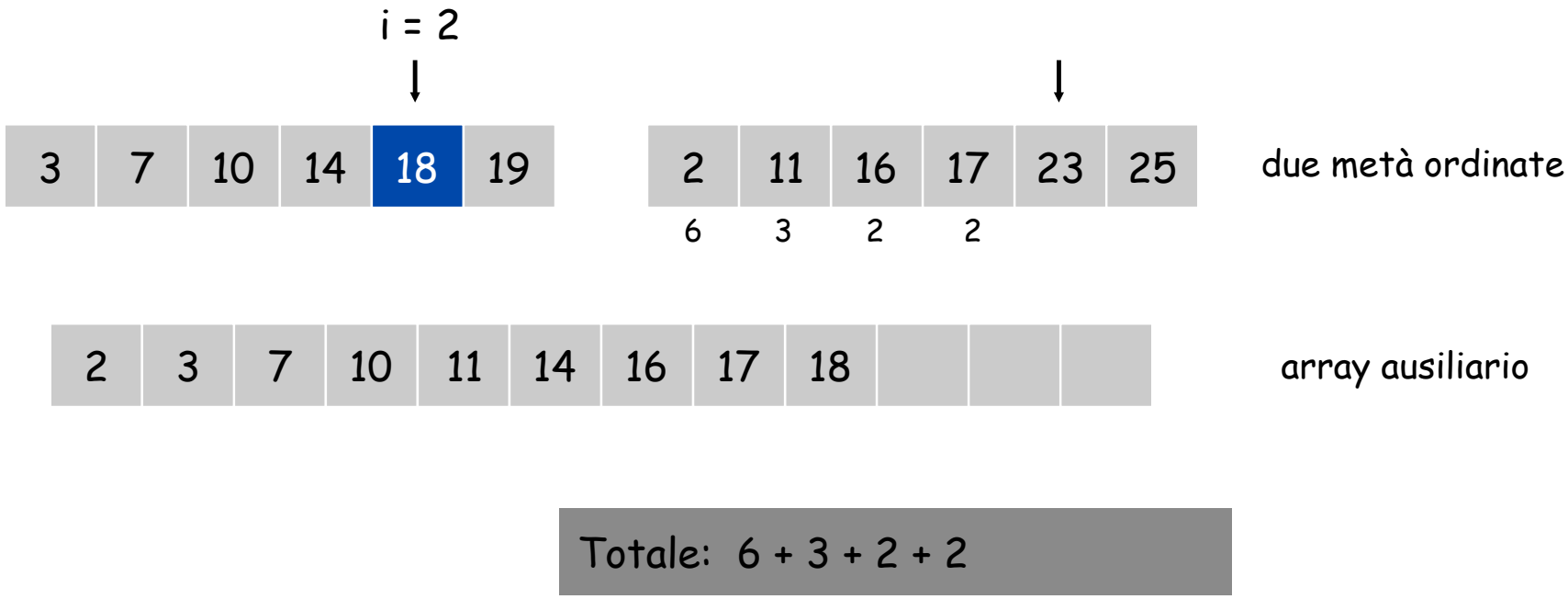
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

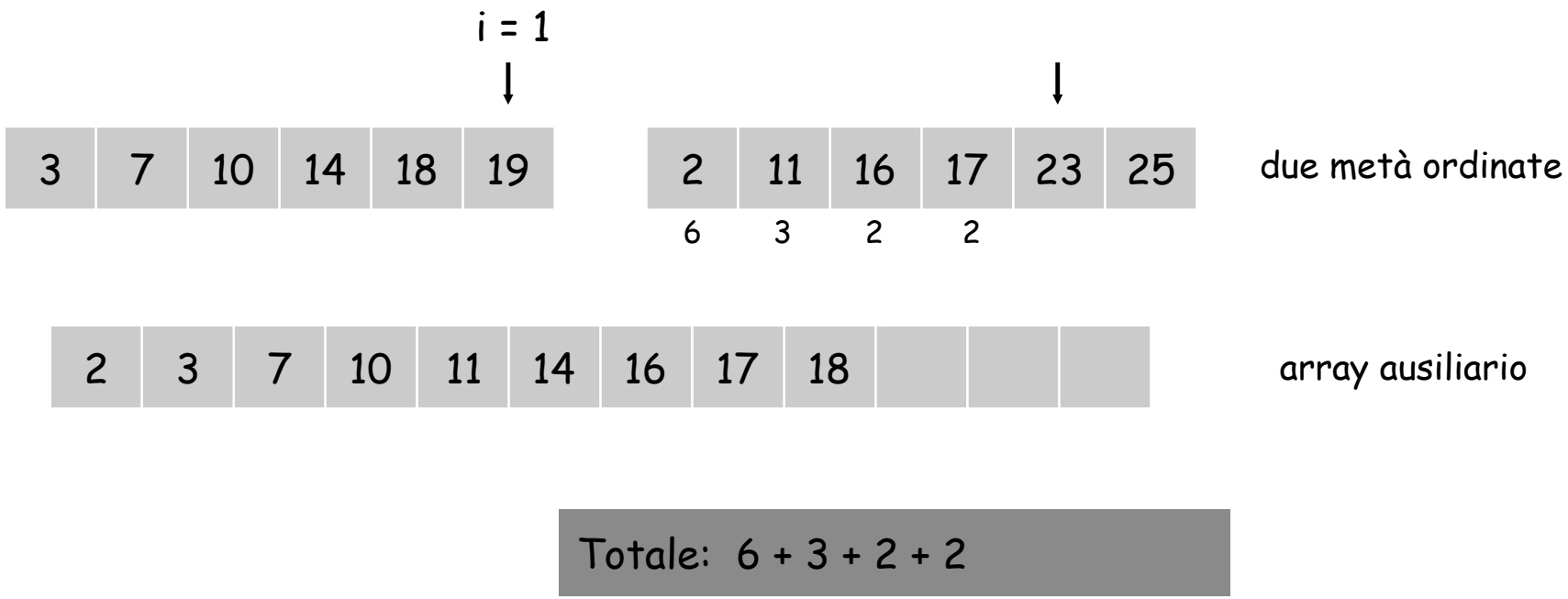
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

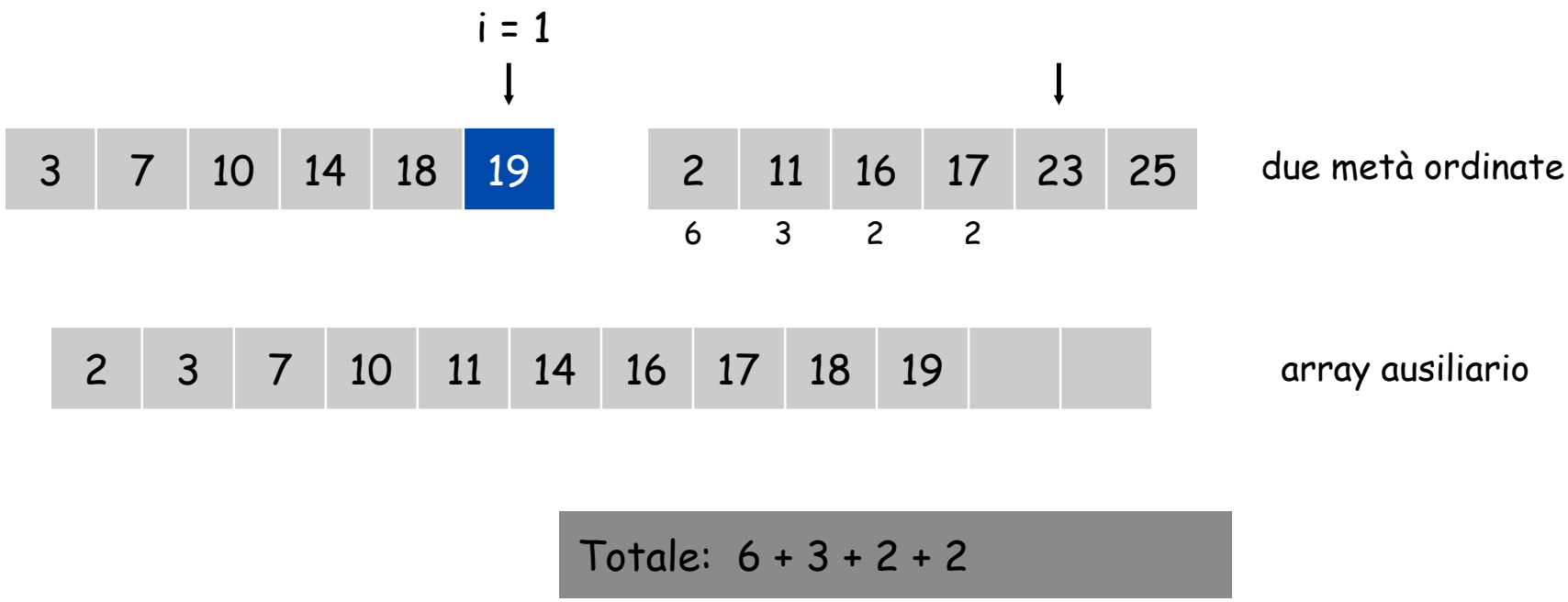
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

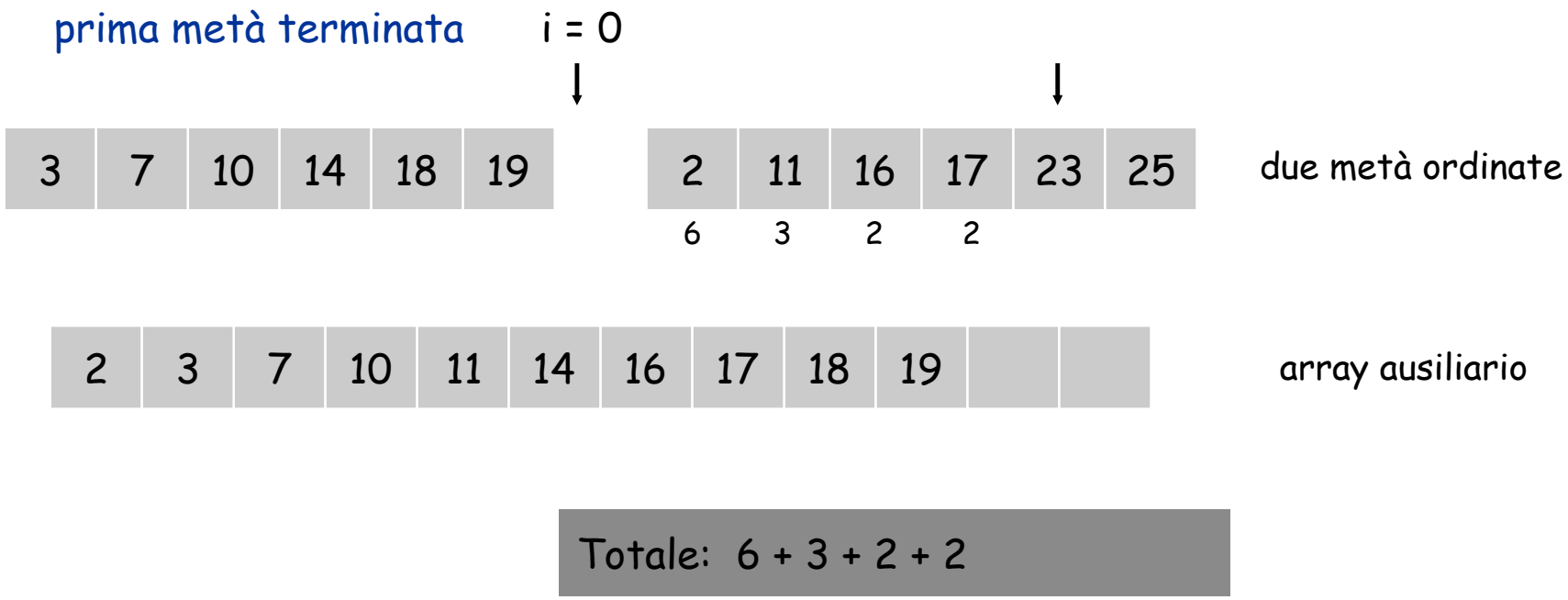
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

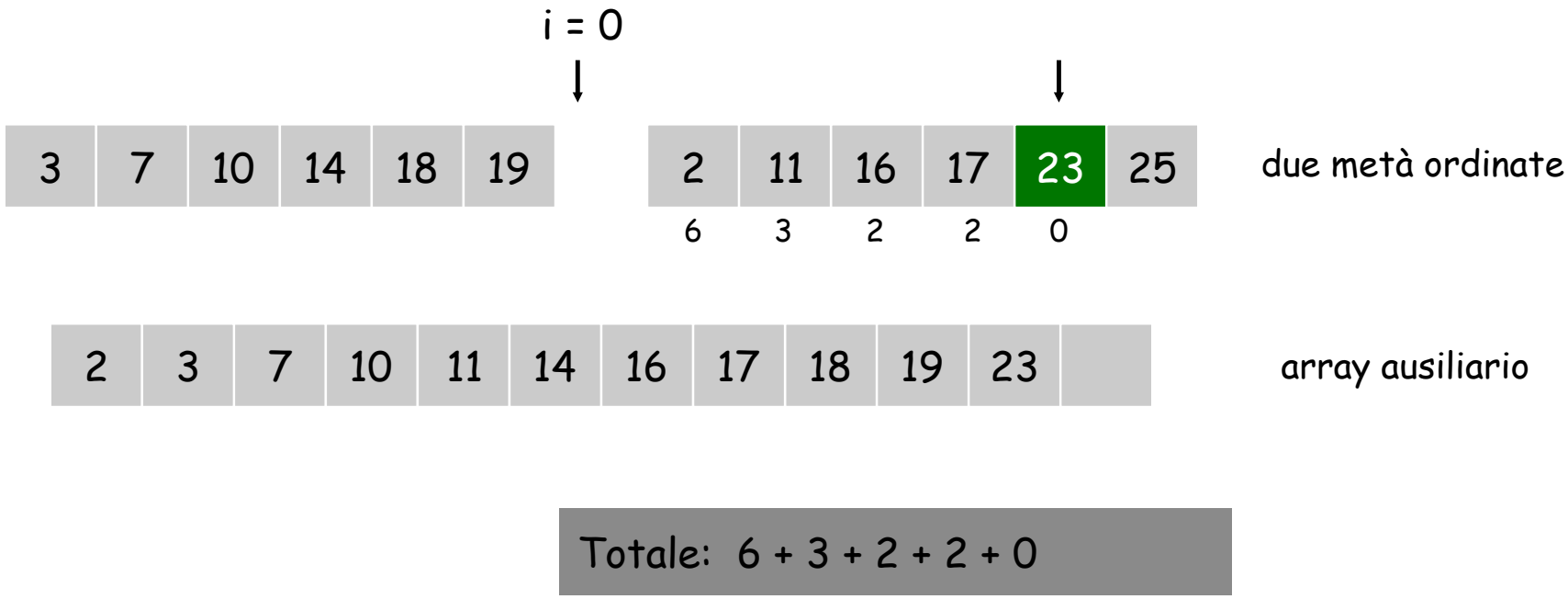
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

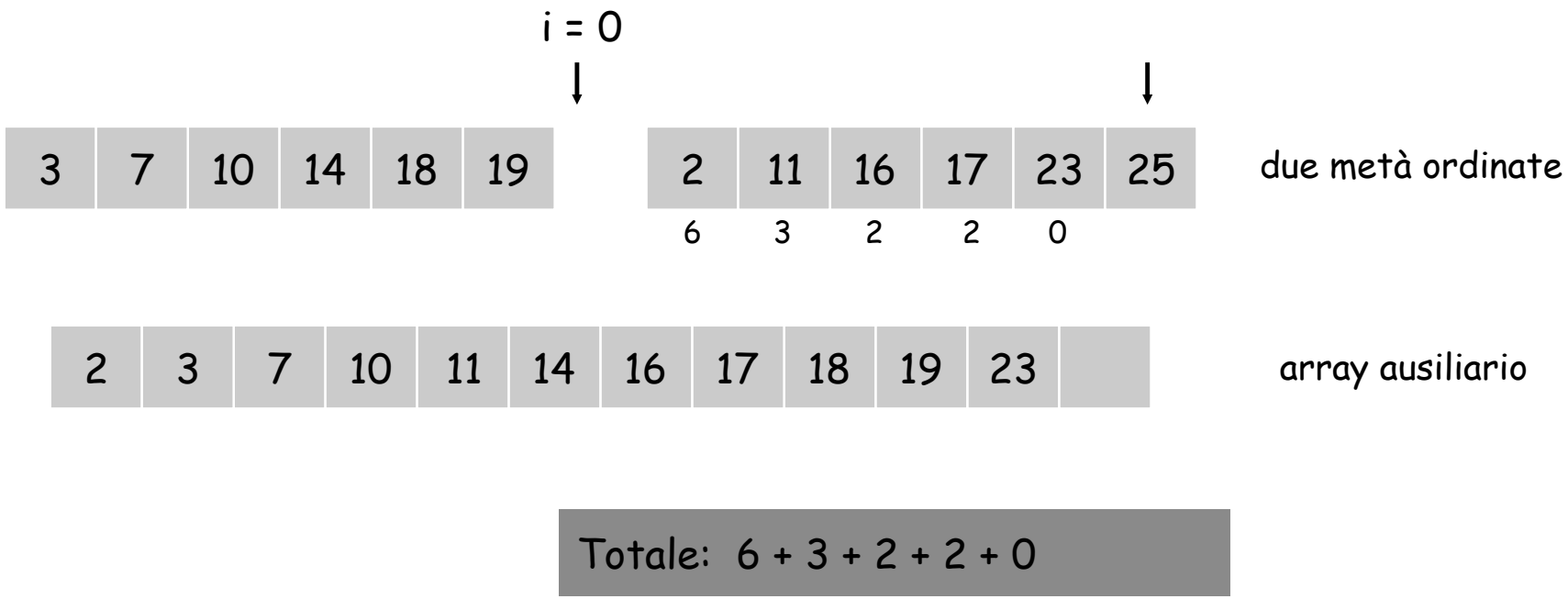
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

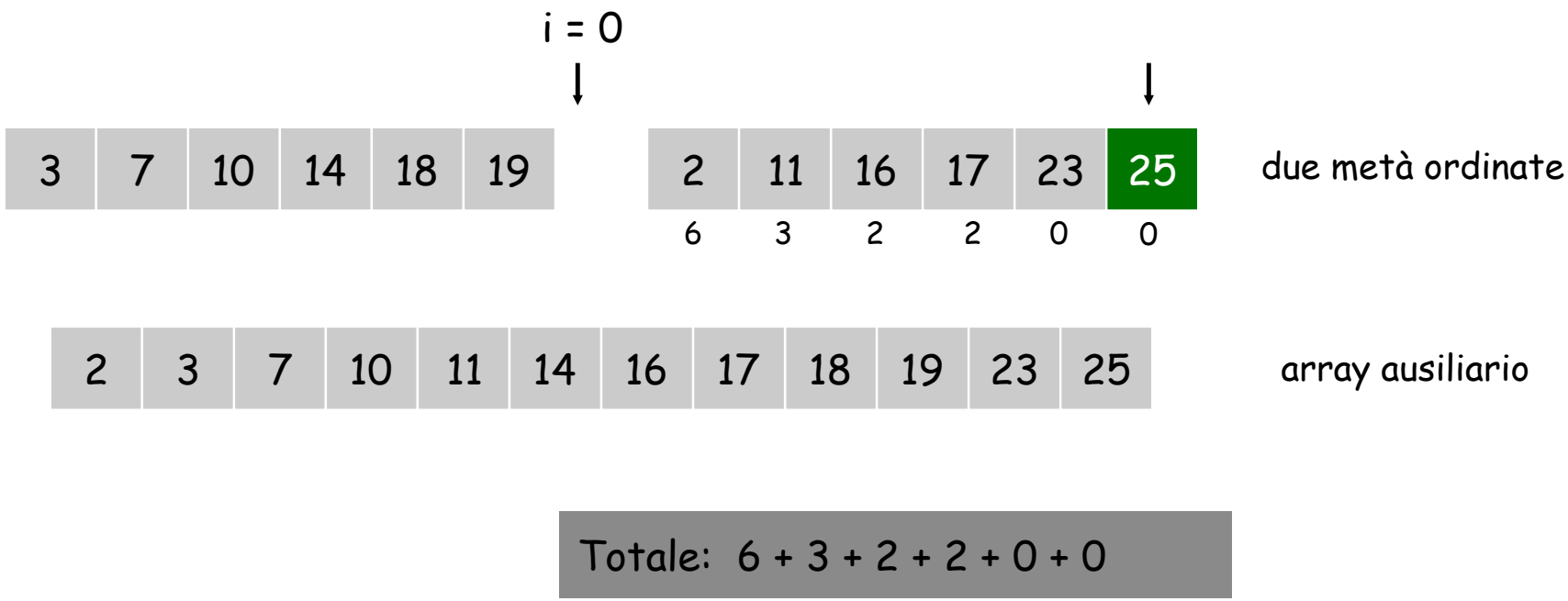
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

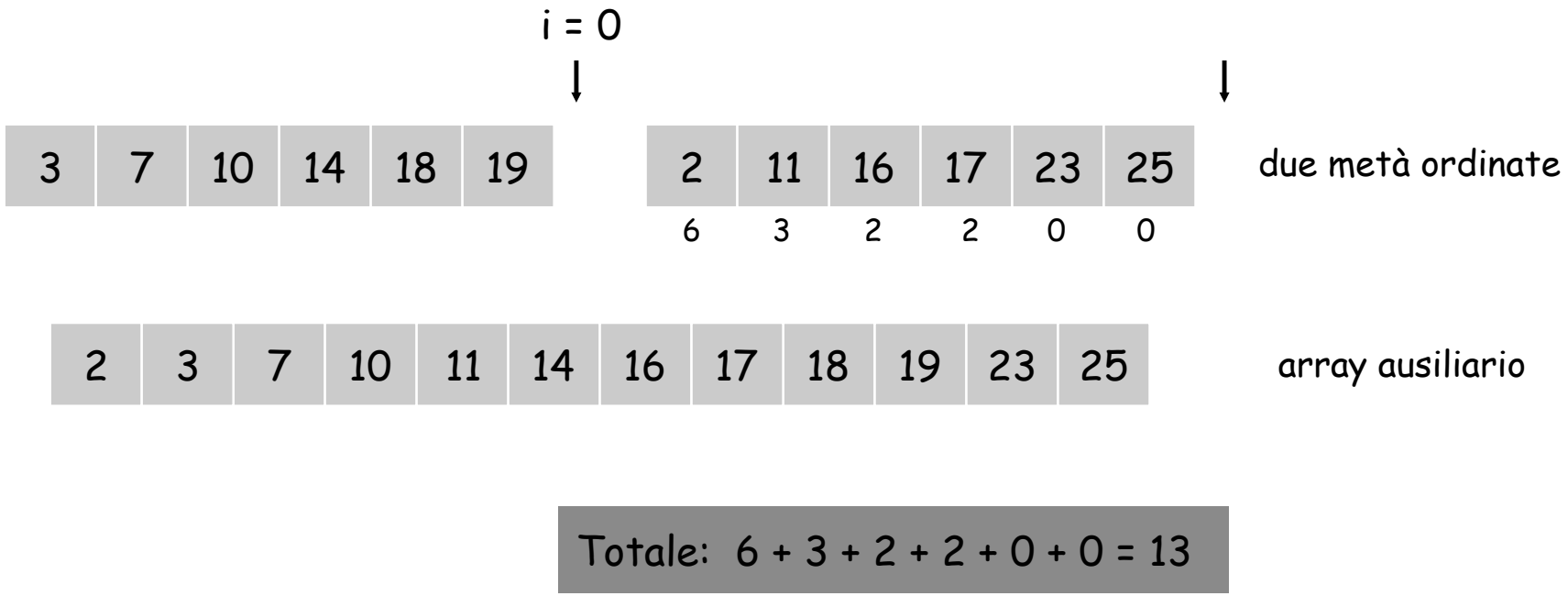
- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Merge and Count

Passo di "merge and count".

- Date due parti ordinate, contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Combiniamo due parti ordinate in una sola sequenza ordinata.



Conteggio Inversioni: Combine

Combine: conteggio inversioni blu-verde

- Assumiamo che ogni metà è ordinata.
- Contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Merge due parti ordinate in una sola sequenza ordinata.

per mantenere l'invariante dell' ordinamento

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

13 inversioni blu-verde: 6 + 3 + 2 + 2 + 0 + 0

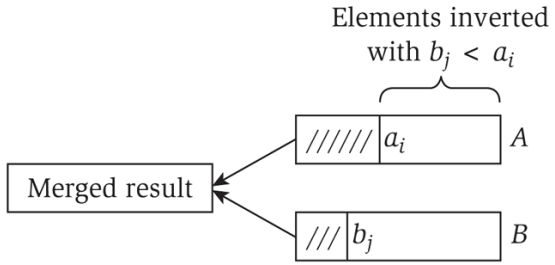


Figure 5.5 Merging two sorted lists while also counting the number of inversions between them.

Conteggio Inversioni: Combine

Combine: conteggio inversioni blu-verde

- Assumiamo che ogni metà è ordinata.
- Contiamo le inversioni dove a_i e a_j sono in parti diverse.
- Merge due parti ordinate in una sola sequenza ordinata.

per mantenere l' invariante dell' ordinamento

13 inversioni blu-verde: 6 + 3 + 2 + 2 + 0 + 0

Conteggio: O(n)

Merge: O(n)

$$T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \Rightarrow T(n) = O(n \log n)$$

Conteggio Inversioni

Algorithm 5.1, page 224

```
Merge-and-Count(A,B)
  Maintain a Current pointer into each list, initialized to
    point to the front elements
  Maintain a variable Count for the number of inversions,
    initialized to 0
  While both lists are nonempty:
    Let a_i and b_j be the elements pointed to by the Current pointer
    Append the smaller of these two to the output list
    If b_j is the smaller element then
      Increment Count by the number of elements remaining in A
    Endif
    Advance the Current pointer in the list from which the
      smaller element was selected.
  EndWhile
  Once one list is empty, append the remainder of the other list
      to the output
  Return Count and the merged list
```

Conteggio Inversioni: Implementation

Pre-condizione. [Merge-and-Count] A e B sono ordinati.
Post-condizione. [Sort-and-Count] L è ordinato.

```
Sort-and-Count(L) {
   if lista L ha un solo elemento
      return 0 e la lista L

   Divide la lista in due metà A e B
   (r_A, A) ← Sort-and-Count(A)
   (r_B, B) ← Sort-and-Count(B)
   (r  , L) ← Merge-and-Count(A, B)

   return r = r_A + r_B + r e la lista ordinata L
}
```

5.4 Closest Pair of Points

Coppia di punti più vicini

Coppia più vicina. Dati n punti nel piano, trovare una coppia con la più piccola distanza euclidea tra loro.

Primitiva geometrica fondamentale.
Grafica, computer vision, sistemi informativi geografici, modellazione molecolare, controllo traffico aereo.

Brute force. Controllare tutte le coppie di punti p e q. Complessità $\Theta(n^2)$.

Coppia di punti più vicini

Coppia più vicina. Dati n punti nel piano, trovare una coppia con la più piccola distanza euclidea tra loro.

Primitiva geometrica fondamentale.
Grafica, computer vision, sistemi informativi geografici, modellazione molecolare, controllo traffico aereo.

Brute force. Controllare tutte le coppie di punti p e q. Complessità $\Theta(n^2)$.

Versione 1-D. Facile se i punti sono su una linea, $O(n \log n)$.

Come?

Coppia di punti più vicini

Coppia più vicina. Dati n punti nel piano, trovare una coppia con la più piccola distanza euclidea tra loro.

Primitiva geometrica fondamentale.
Grafica, computer vision, sistemi informativi geografici, modellazione molecolare, controllo traffico aereo.

Brute force. Controllare tutte le coppie di punti p e q. Complessità $\Theta(n^2)$.

Versione 1-D. Facile se i punti sono su una linea, $O(n \log n)$.

Come?

1. Ordinamento
2. Minimo tra tutte le distanze tra punti successivi

Esempio
- Input: 5, 20, 10, 7, 2, 13
- Ordinamento: 2, 5, 7, 10, 13, 20
- Minimo tra tutte le distanze tra ogni punto ed il successivo: min {3, 2, 3, 3, 7} = 2

Coppia di punti più vicini

Coppia più vicina. Dati n punti nel piano, trovare una coppia con la più piccola distanza euclidea tra loro.

Primitiva geometrica fondamentale.
Grafica, computer vision, sistemi informativi geografici, modellazione molecolare, controllo traffico aereo.

Brute force. Controllare tutte le coppie di punti p e q. Complessità $\Theta(n^2)$.

Versione 1-D. Facile se i punti sono su una linea, O(n log n).

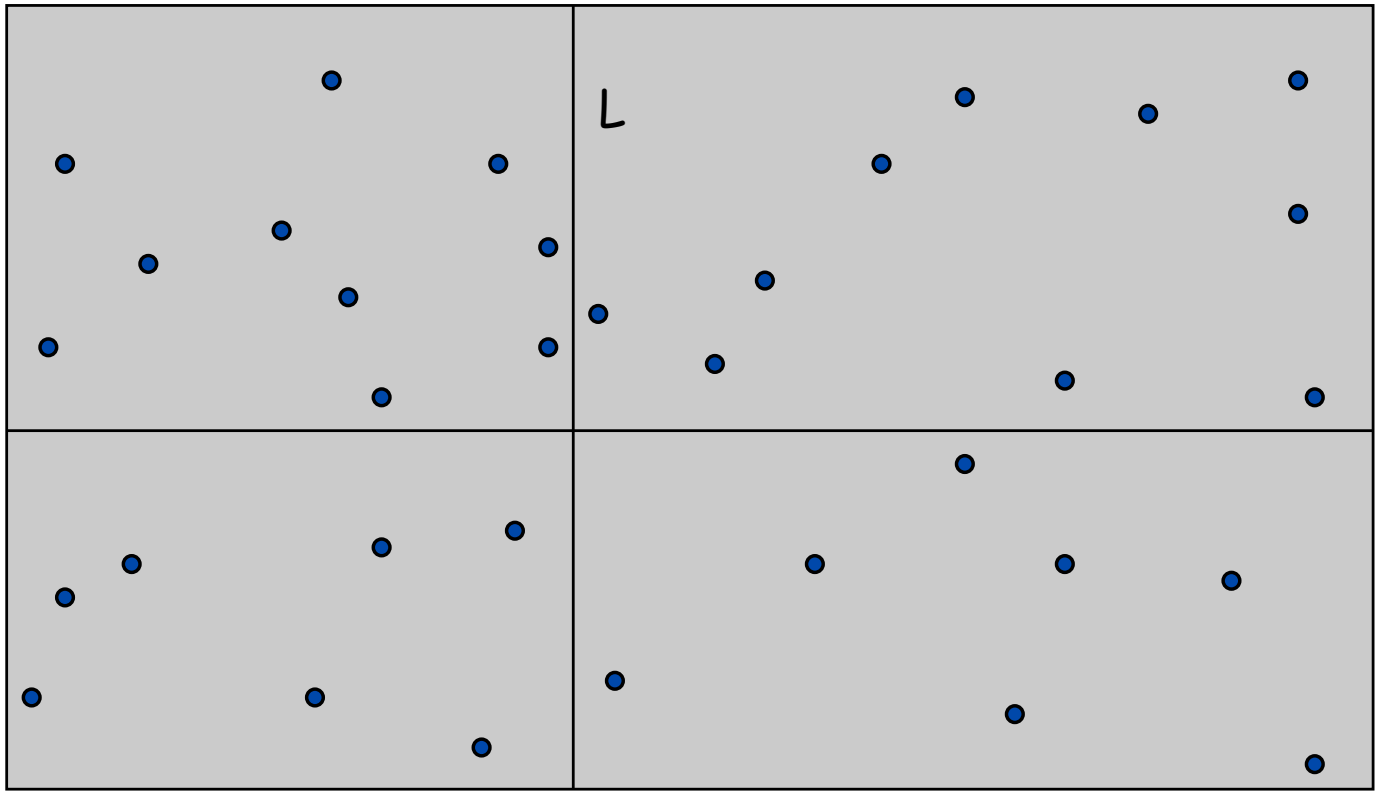
Assunzione. Punti con diverse coordinate x.

↑
per semplificare la presentazione. Altrimenti rotazione oppure estendere algoritmo

Problema considerato da M. I. Shamos e D. Hoey, inizio anni '70.
L' algoritmo O(n log n) che vedremo è essenzialmente la loro soluzione.

Coppia di punti più vicini: Primo tentativo

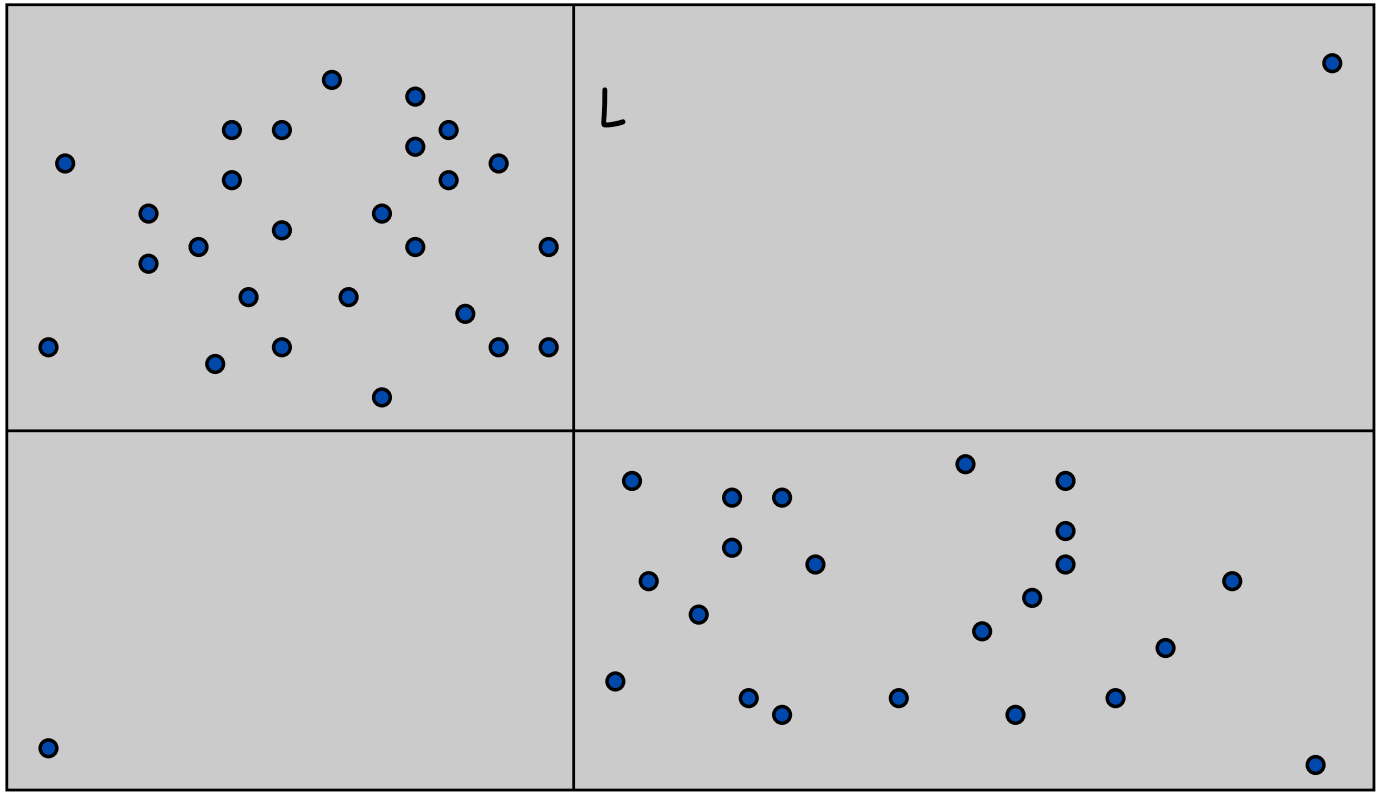
Divide. Dividiamo la regione in 4 quadranti.



Coppia di punti più vicini: Primo tentativo

Divide. Dividiamo la regione in 4 quadranti

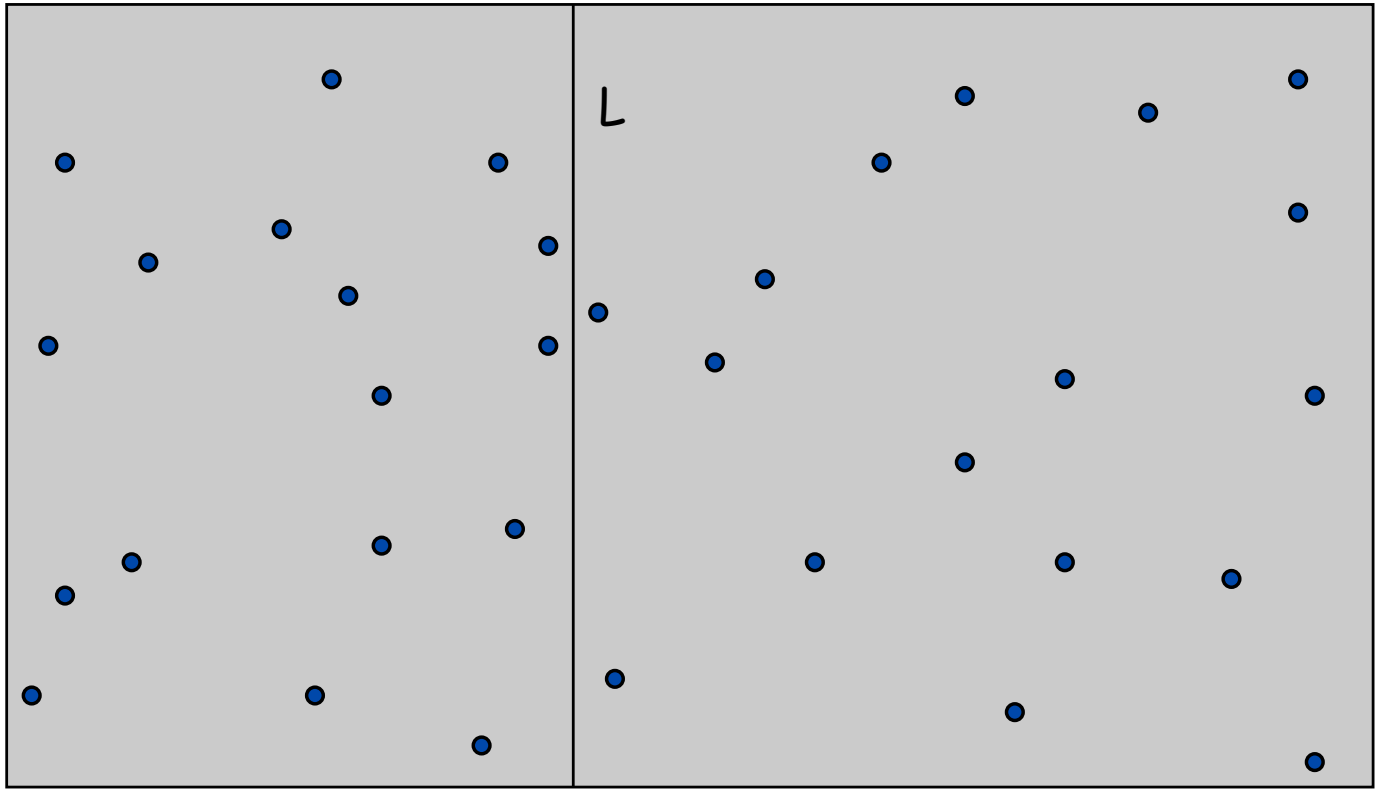
Ostacolo. Impossibile assicurare n/4 punti in ogni parte.



Coppia di punti più vicini

Algoritmo.

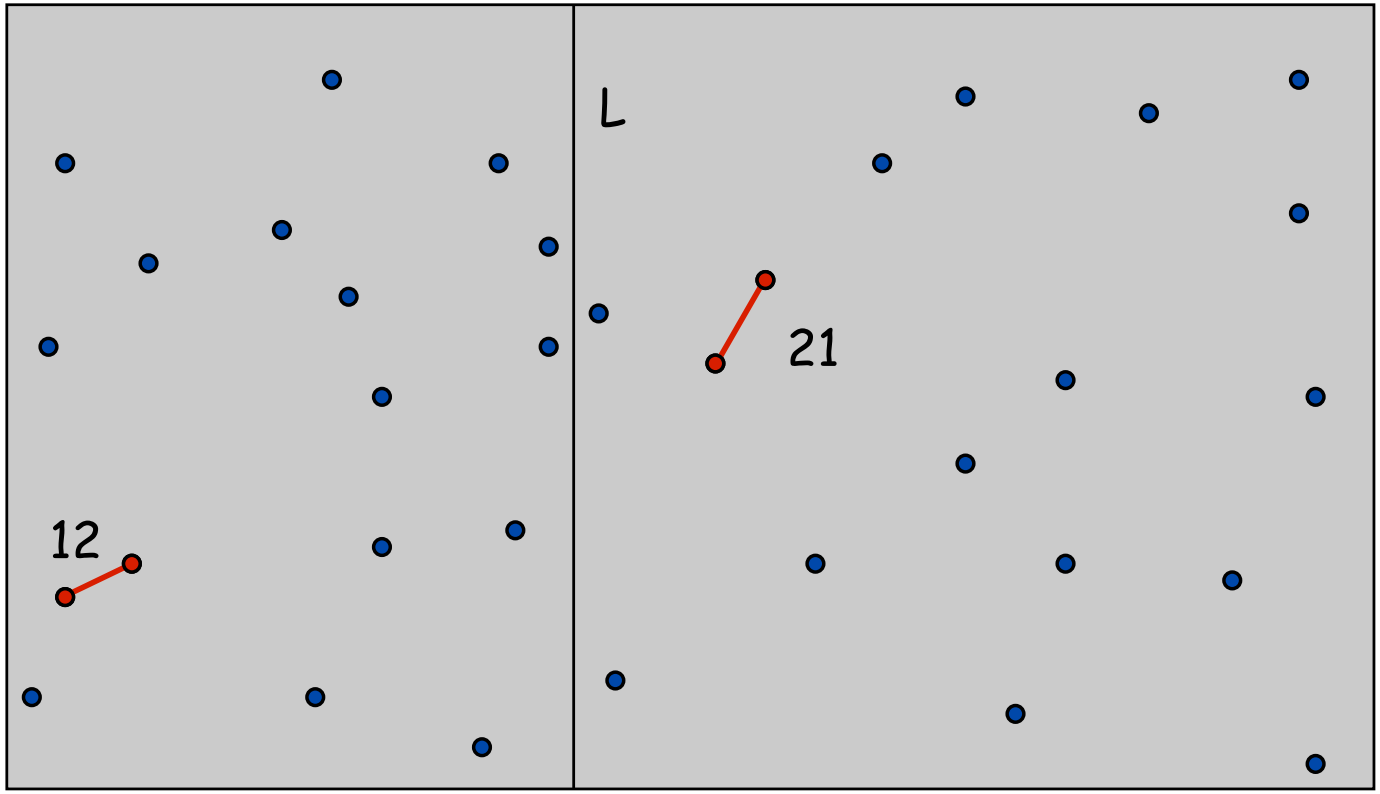
- Divide: disegnare linea verticale L cosi che ci sono circa n/2 punti in ogni parte.



Coppia di punti più vicini

Algoritmo.

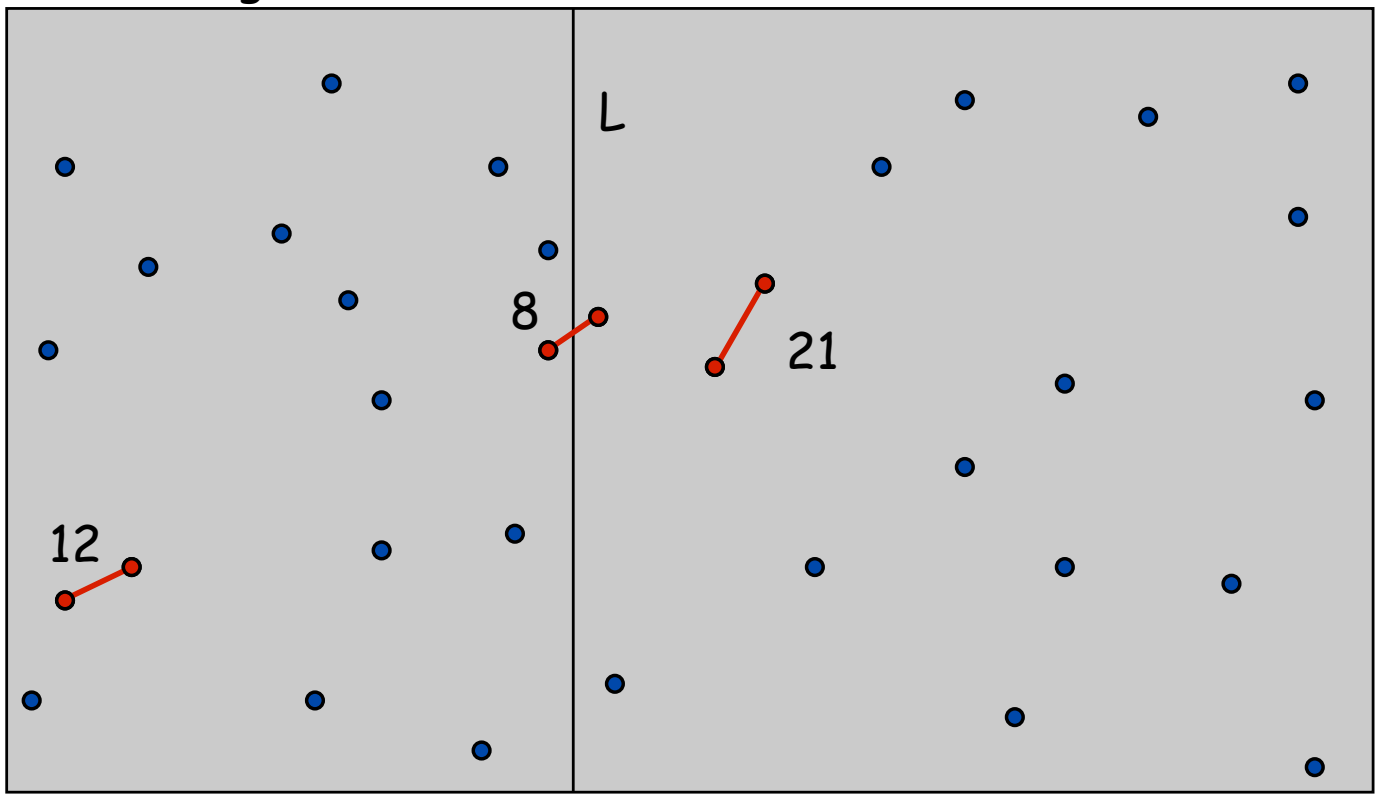
- Divide: disegnare linea verticale L cosi che ci sono circa n/2 punti in ogni parte.
- Conquer: trovare ricorsivamente la coppia più vicina in ogni parte.



Coppia di punti più vicini

Algoritmo.

- Divide: disegnare linea verticale L cosi che ci sono circa n/2 punti in ogni parte.
- Conquer: trovare ricorsivamente la coppia più vicina in ogni parte.
- Combine: trovare la coppia più vicina con un punto in ogni parte.
- Restituisci la migliore delle 3 soluzioni.

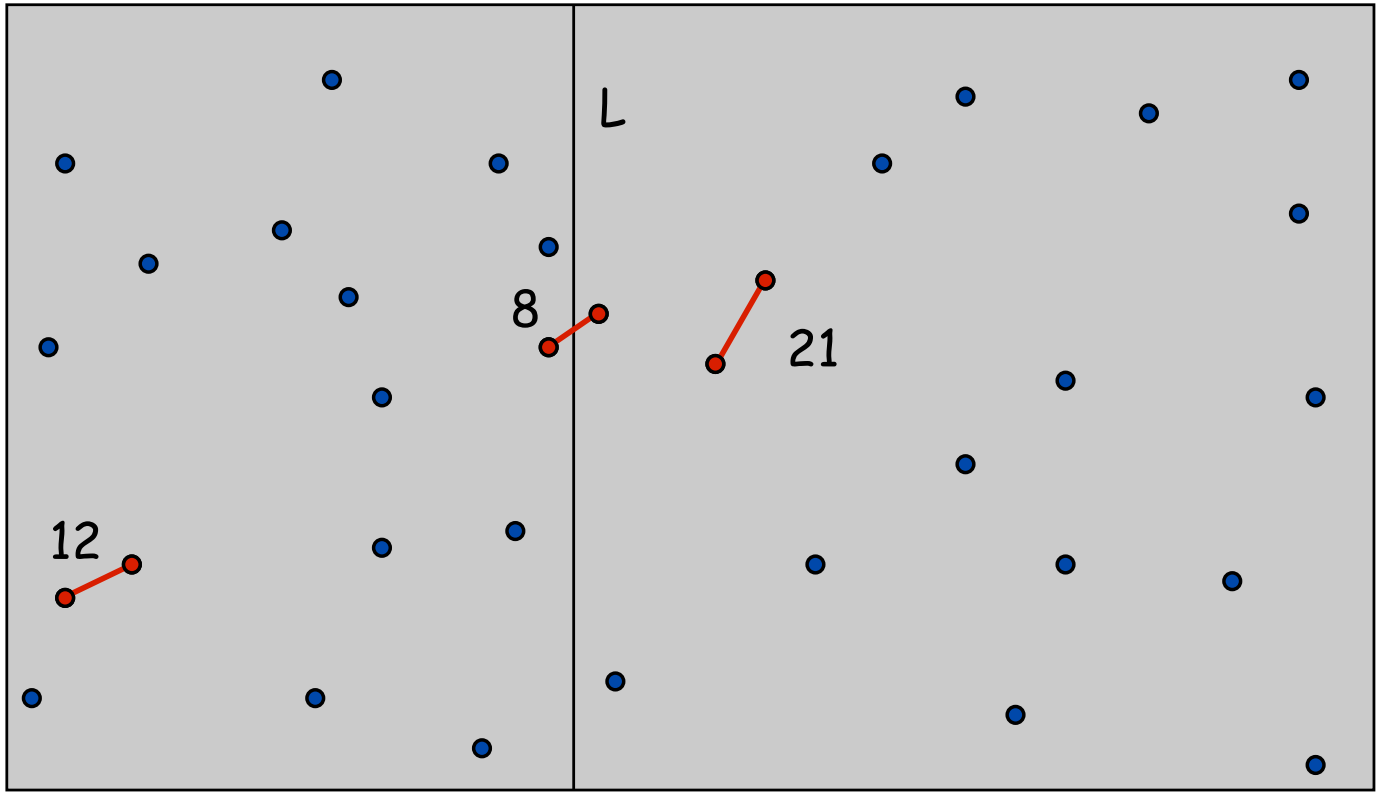


Coppia di punti più vicini

Algoritmo.

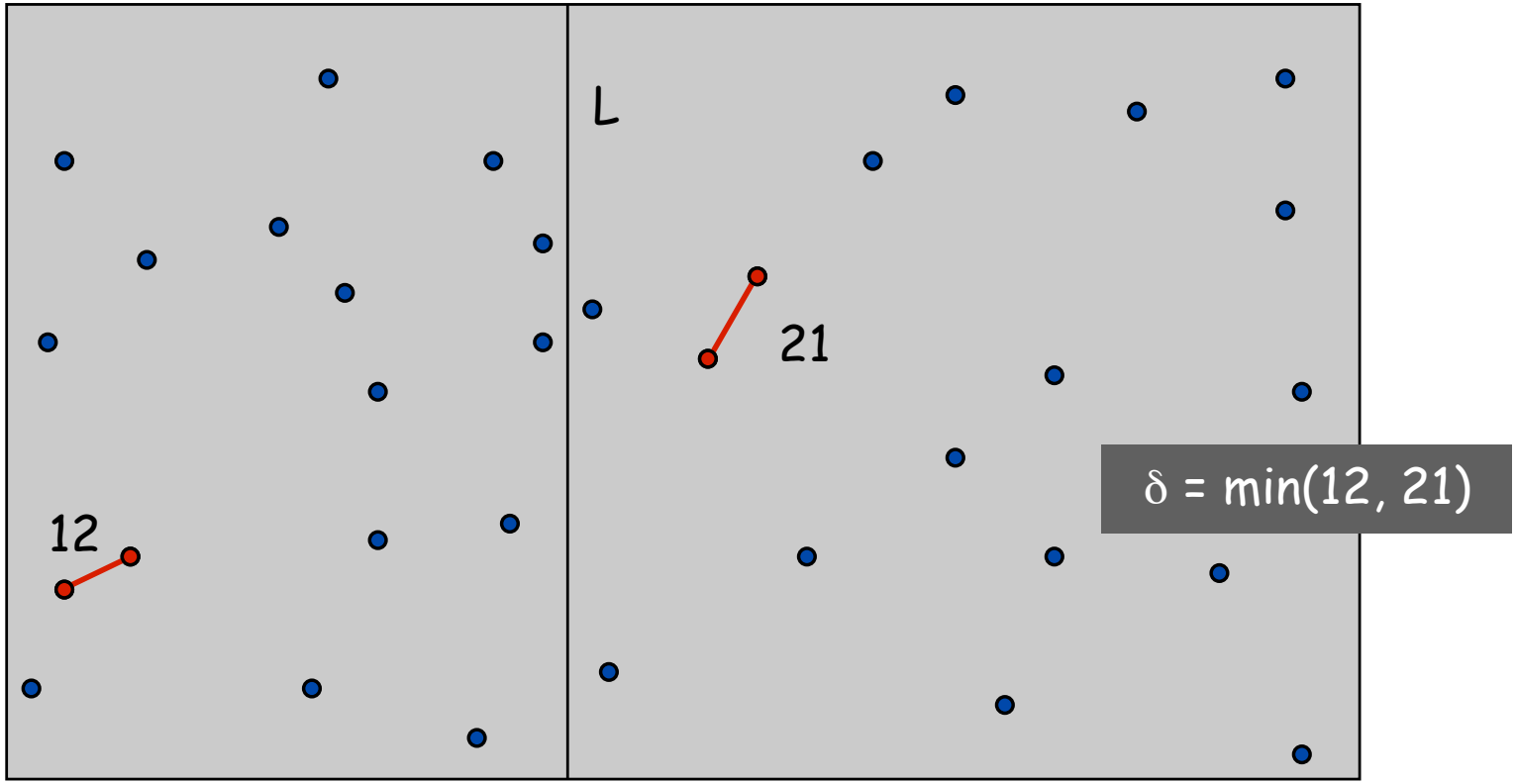
- Divide: disegnare linea verticale L cosi che ci sono circa n/2 punti in ogni parte.
- Conquer: trovare ricorsivamente la coppia più vicina in ogni parte.
- Combine: trovare la coppia più vicina con un punto in ogni parte.
- Restituisci la migliore delle 3 soluzioni.

sembra $\Theta(n^2)$



Coppia di punti più vicini

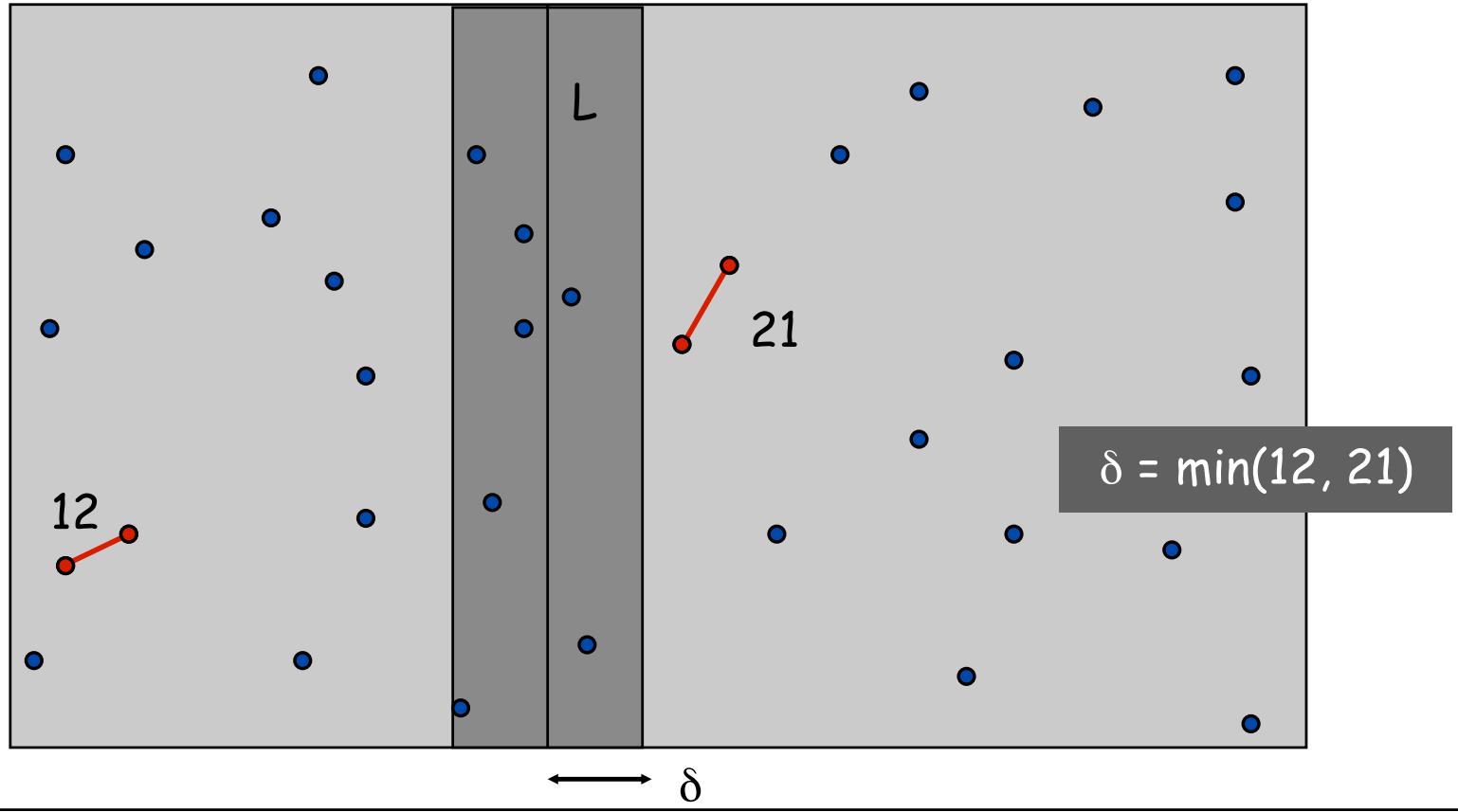
Trovare la coppia più vicina con un punto in ogni parte, assumendo che la distanza sia < δ.



Coppia di punti più vicini

Trovare la coppia più vicina con un punto in ogni parte, assumendo che la distanza sia < δ.

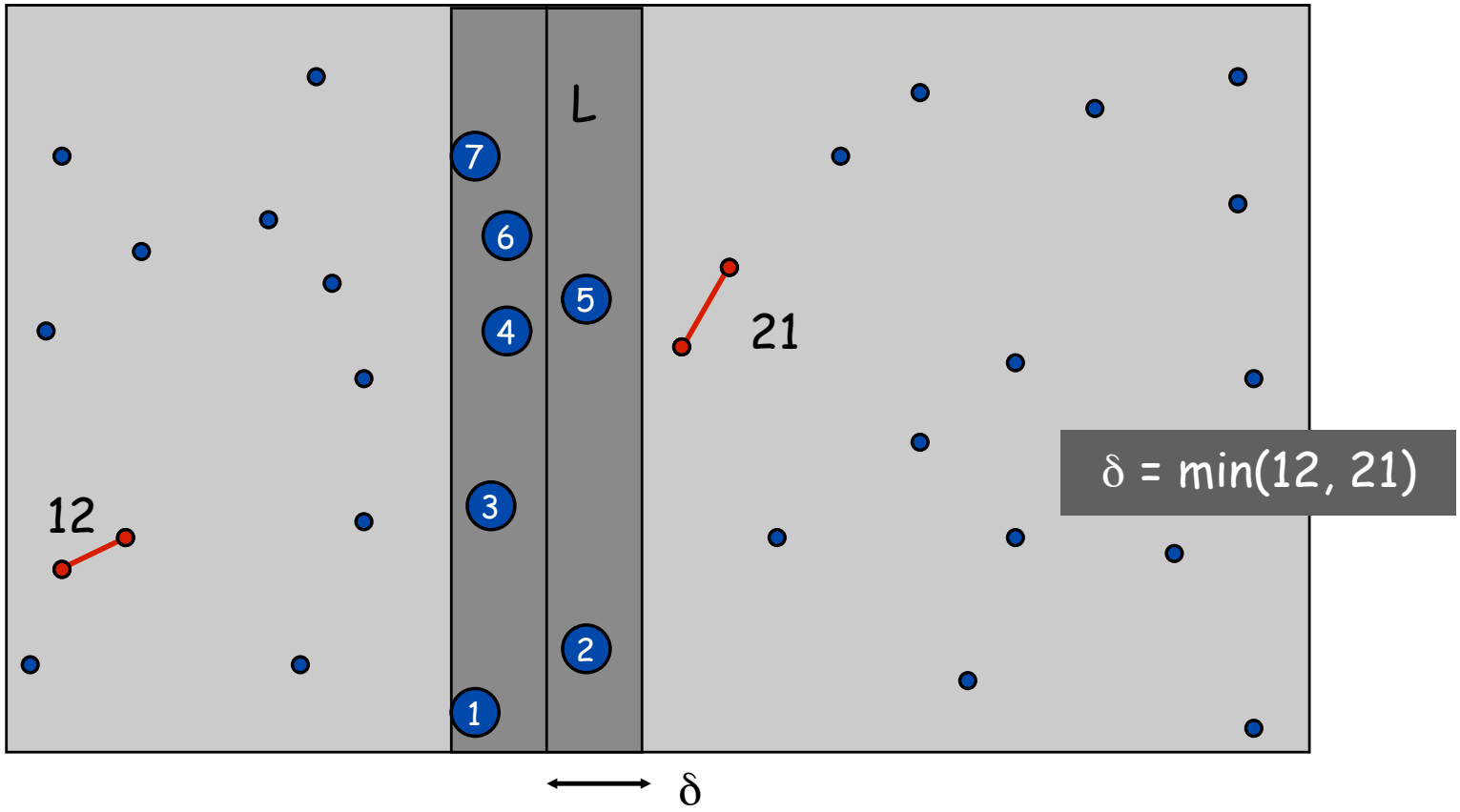
- Observazione: è inutile considerare punti oltre distanza δ da L.



Coppia di punti più vicini

Trovare la coppia più vicina con un punto in ogni parte, assumendo che la distanza sia < δ.

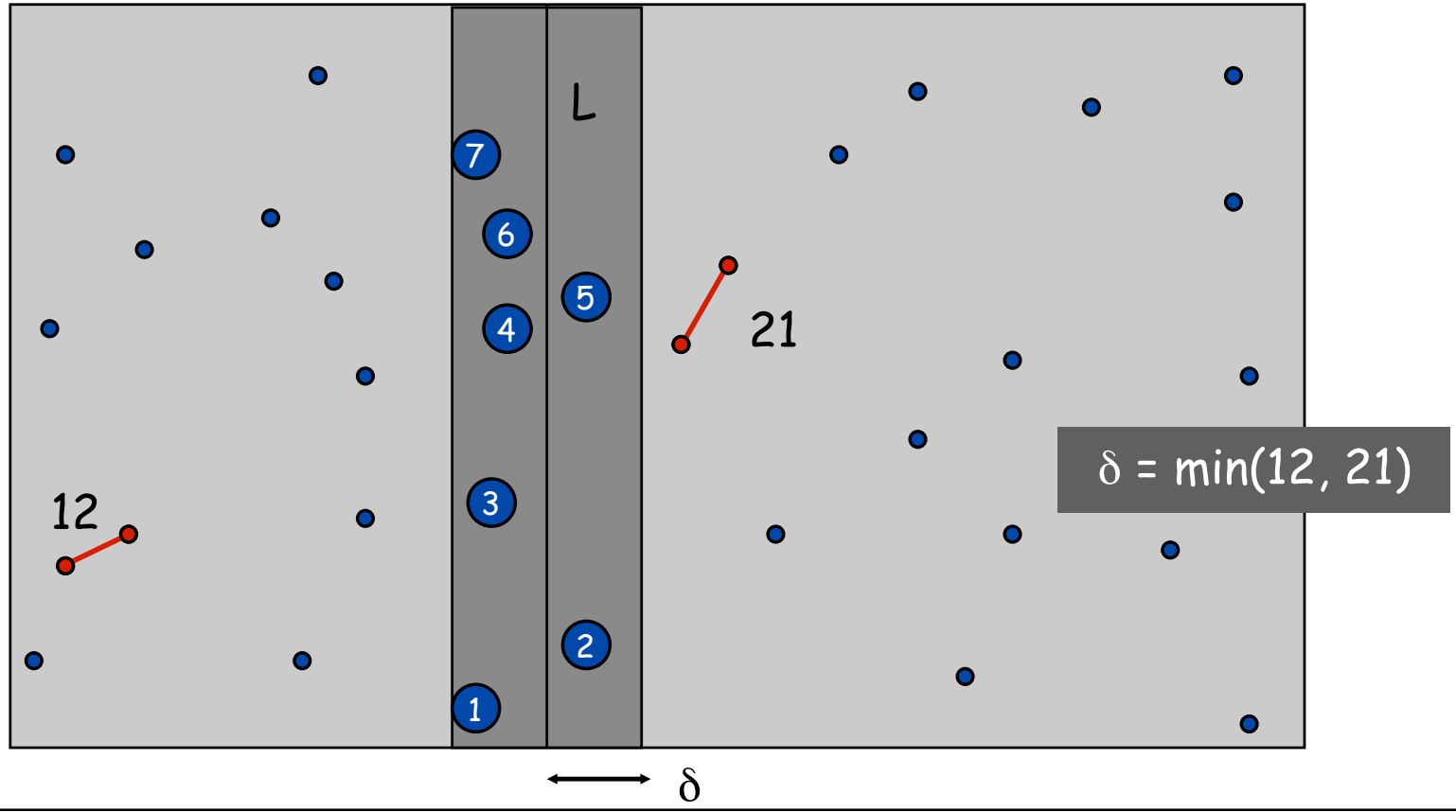
- Observazione: è inutile considerare punti oltre distanza δ da L.
- Ordinare punti nella striscia 2δ per coordinata y.



Coppia di punti più vicini

Trovare la coppia più vicina con un punto in ogni parte, assumendo che la distanza sia < δ.

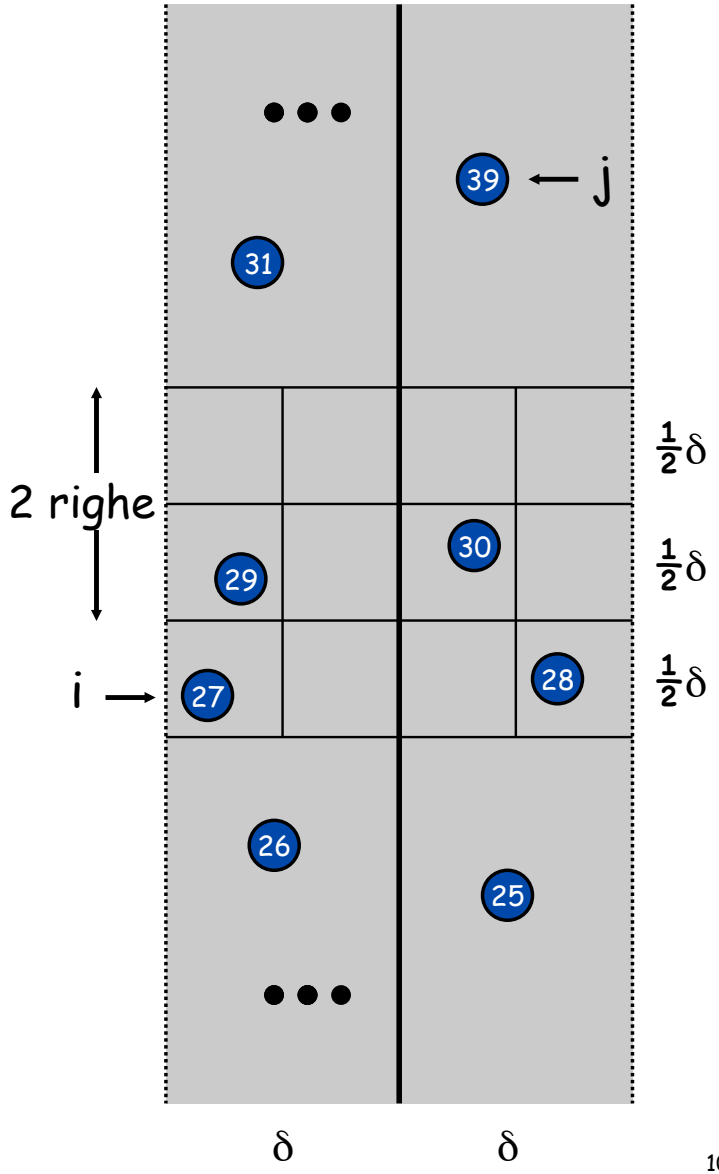
- Observazione: è inutile considerare punti oltre distanza δ da L.
- Ordinare punti nella striscia 2δ per coordinata y.
- Considerare solo i punti entro 11 posizioni nella lista ordinata!



Coppia di punti più vicini

Definizione. Sia s_i il punto nella striscia 2δ, con la i-esima più piccola coordinata y.

Claim. Se $|i - j| \geq 12$, allora la distanza tra s_i ed s_j è almeno δ.



Coppia di punti più vicini

Definizione. Sia s_i il punto nella striscia 2δ, con la i-esima più piccola coordinata y.

Claim. Se $|i - j| \geq 12$, allora la distanza tra s_i ed s_j è almeno δ.

Prova.

- Non ci sono due punti nello stesso quadrato $\frac{1}{2}\delta$-per-$\frac{1}{2}\delta$ (distanza max $\frac{1}{2}\delta \cdot \sqrt{2} < \delta$).

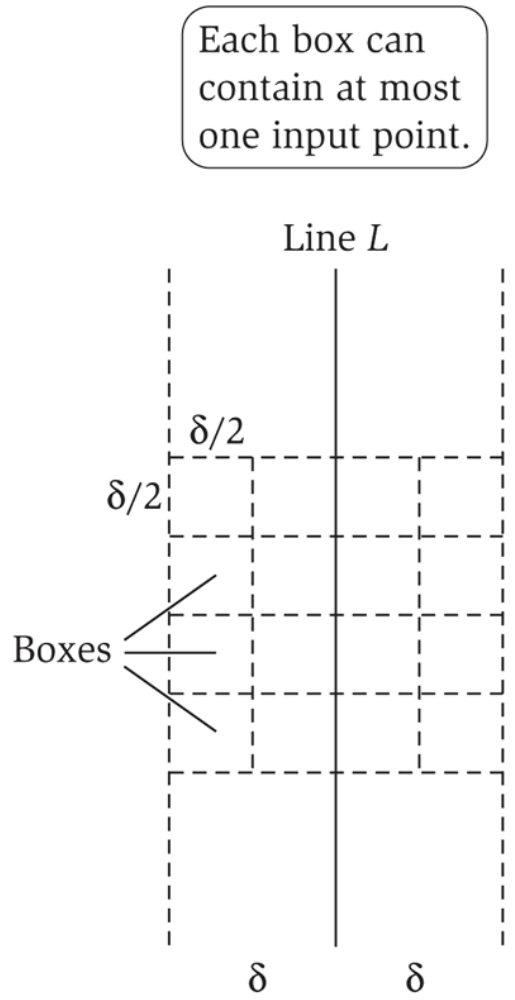


Figure 5.7 The portion of the plane close to the dividing line L, as analyzed in the proof of (5.10).

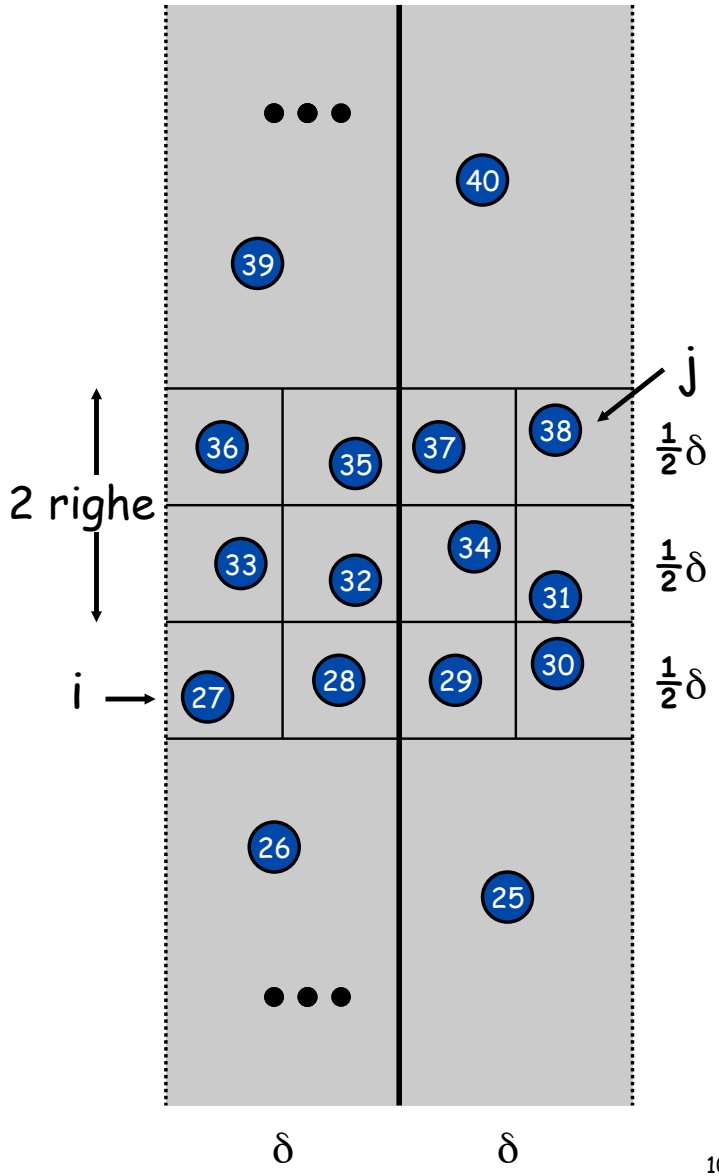
Coppia di punti più vicini

Definizione. Sia s_i il punto nella striscia 2δ, con la i-esima più piccola coordinata y.

Claim. Se $|i - j| \geq 12$, allora la distanza tra s_i ed s_j è almeno δ.

Prova.

- Non ci sono due punti nello stesso quadrato $\frac{1}{2}\delta$-per-$\frac{1}{2}\delta$ (distanza max $\frac{1}{2}\delta \cdot \sqrt{2} < \delta$).
- Due punti con almeno due righe tra loro hanno distanza $\geq 2(\frac{1}{2}\delta)$.
- Se vi è al massimo una riga tra loro, allora $|i - j| \leq 11$. ▪



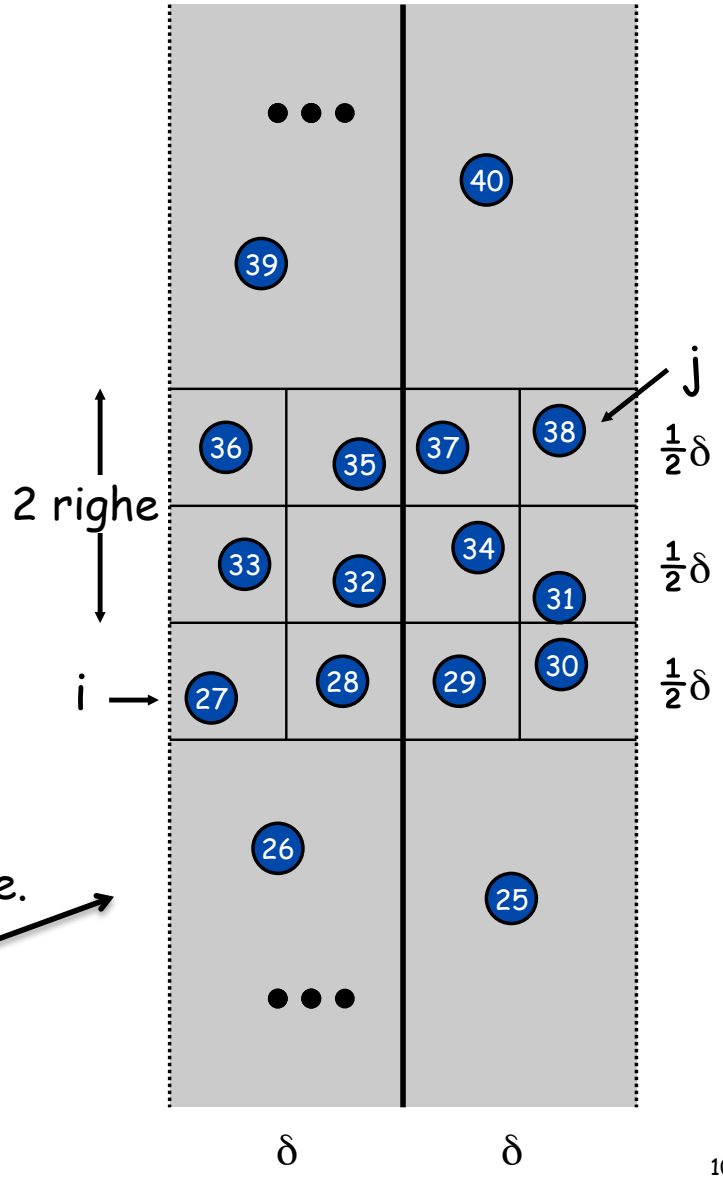
Coppia di punti più vicini

Definizione. Sia s_i il punto nella striscia 2δ, con la i-esima più piccola coordinata y.

Claim. Se $|i - j| \geq 12$, allora la distanza tra s_i ed s_j è almeno δ.

Prova.

- Non ci sono due punti nello stesso quadrato $\frac{1}{2}\delta$-per-$\frac{1}{2}\delta$ (distanza max $\frac{1}{2}\delta \cdot \sqrt{2} < \delta$).
- Due punti con almeno due righe tra loro hanno distanza $\geq 2(\frac{1}{2}\delta)$.
- Se vi è al massimo una riga tra loro, allora $|i - j| \leq 11$. ▪



Nota. Il valore della costante non è molto importante.

Claim ancora vero se sostituiamo 11 con 6.

L'esempio della figura non può accadere

Sul testo troviamo 15 al posto di 11.

Coppia di punti più vicini: algoritmo

```
Closest-Pair(p1, …, pn) {

   Calcolare linea di separazione L tale che divide i
   punti in due parti uguali.

   δ1 = Closest-Pair(metà sinistra)
   δ2 = Closest-Pair(metà destra)
   δ  = min(δ1, δ2)

   Cancellare tutti punti con distanza >δ da L

   Ordinare i punti rimanenti per coordinata y.

   Scansione punti ordinati rispetto ad y e confrontare
   distanza tra ogni punto ed i successivi 11 punti.
   Se una distanza è <δ, aggiornare δ.

   return δ.
}
```

Coppia di punti più vicini: complessità algoritmo

Indichiamo con T(n) la complessità dell' algoritmo per n punti

```
Closest-Pair(p1, …, pn) {

   Calcolare linea di separazione L tale che divide i
   punti in due parti uguali.                                  O(n log n)

   δ1 = Closest-Pair(metà sinistra)
   δ2 = Closest-Pair(metà destra)                              2T(n / 2)
   δ  = min(δ1, δ2)

   Cancellare tutti punti con distanza >δ da L                 O(n)

   Ordinare i punti rimanenti per coordinata y.                O(n log n)

   Scansione punti ordinati rispetto ad y e confrontare
   distanza tra ogni punto ed i successivi 11 punti.           O(n)
   Se una distanza è <δ, aggiornare δ.

   return δ.
}
```

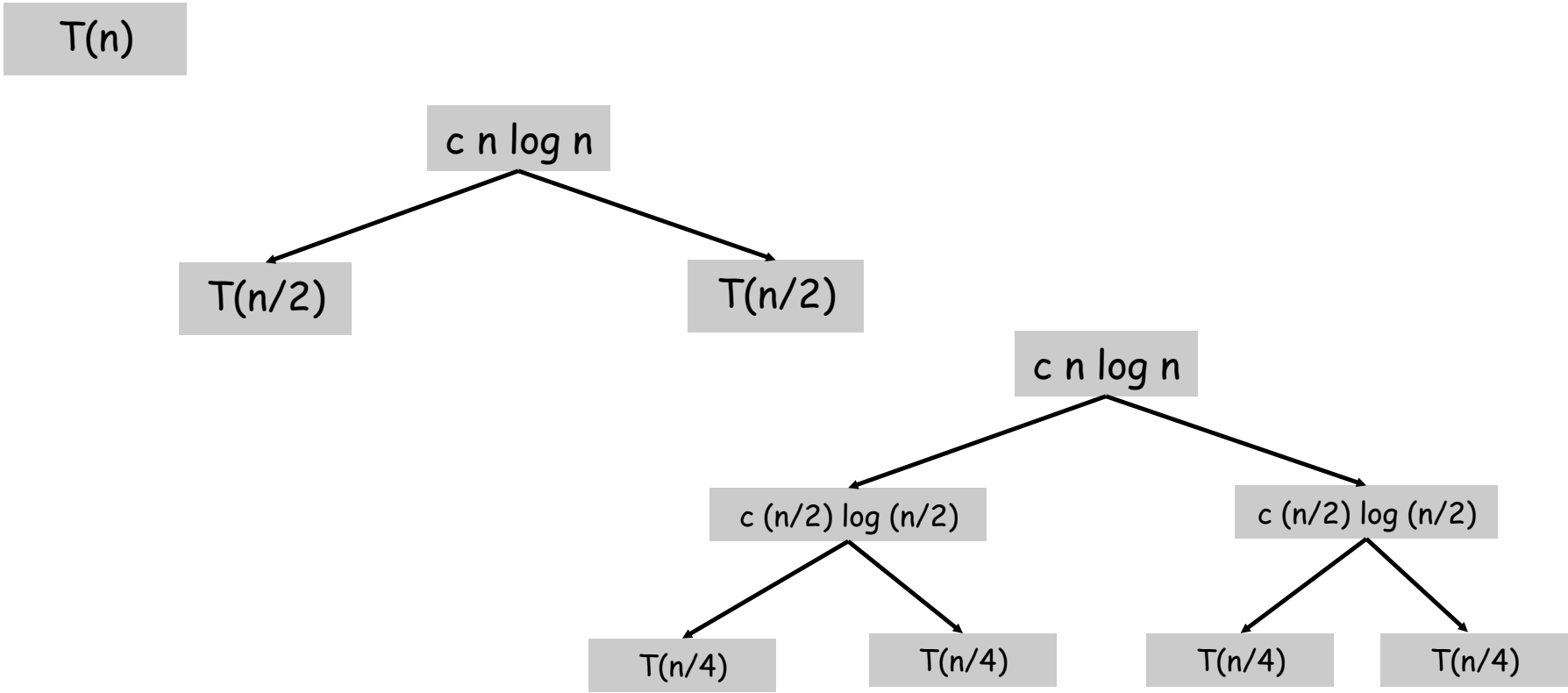
Coppia di punti più vicini: complessità algoritmo

Complessità algoritmo.

$$\mathrm{T}(n) \leq 2T(n/2) + O(n \log n) \Rightarrow \mathrm{T}(n) = O(n \log^2 n)$$

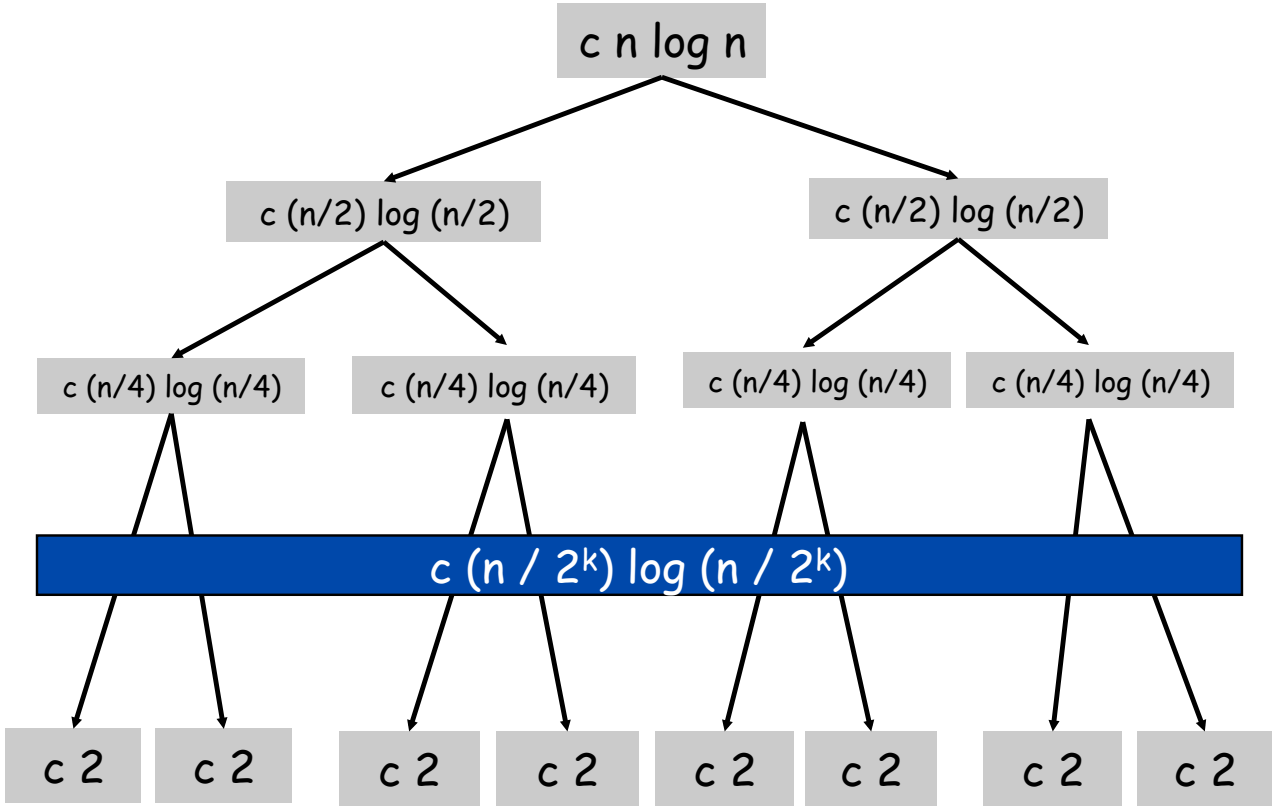
Prova con Albero di Ricorsione

$$\mathrm{T}(n) = 2T(n/2) + O(n \log n) \Rightarrow \mathrm{T}(n) = O(n \log^2 n)$$



Prova con Albero di Ricorsione

$$\mathrm{T}(n) \leq 2T(n/2) + O(n \log n) \Rightarrow \mathrm{T}(n) = O(n \log^2 n)$$



h+1

c n log n

2 c (n/2) log (n/2)

4 c (n/4) log (n/4)

2^k c (n/2^k) log (n/2^k)

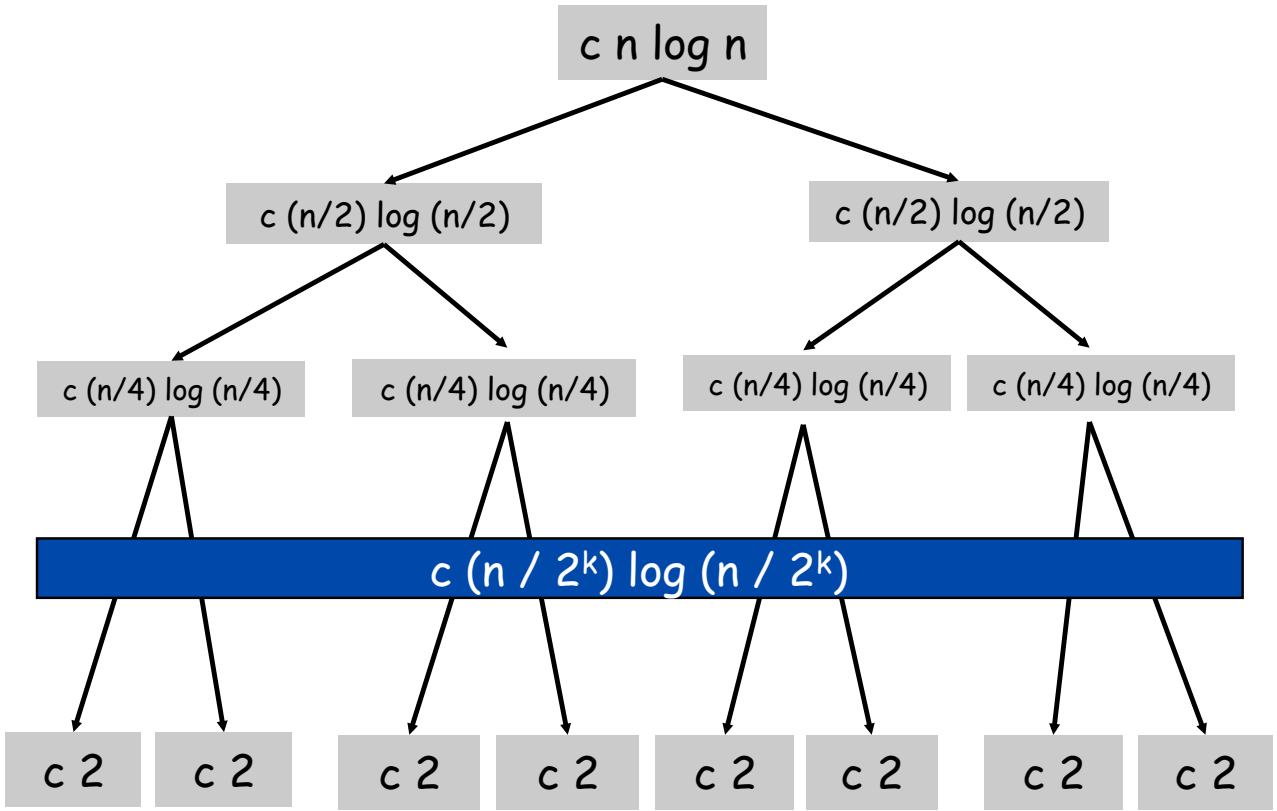
n/2 c (2) log (2)

Altezza albero h?

$$c\sum_{i=1}^{h}\frac{n}{2^i}\log\frac{n}{2^i} \leq hcn\log n$$

Prova con Albero di Ricorsione

$$\mathrm{T}(n) \leq 2T(n/2) + O(n \log n) \Rightarrow \mathrm{T}(n) = O(n \log^2 n)$$



h+1

c n log n

2 c (n/2) log (n/2)

4 c (n/4) log (n/4)

2^k c (n/2^k) log (n/2^k)

n/2 c (2) log (2)

Altezza albero h? $n / 2^h = 2$ cioè $h = \log_2 n - 1$

$$c\sum_{i=1}^{h} \frac{n}{2^i} \log \frac{n}{2^i} \leq h\, c\, n \log n$$

Coppia di punti più vicini: complessità algoritmo

Complessità algoritmo.

$$\mathrm{T}(n) \leq 2T(n/2) + O(n \log n) \Rightarrow \mathrm{T}(n) = O(n \log^2 n)$$

Domanda. Possiamo ottenere O(n log n)?

Coppia di punti più vicini: complessità algoritmo

Indichiamo con T(n) la complessità dell' algoritmo per n punti

```
Closest-Pair(p1, …, pn) {

   Calcolare linea di separazione L tale che divide i
   punti in due parti uguali.                                  O(n log n)

   δ1 = Closest-Pair(metà sinistra)
   δ2 = Closest-Pair(metà destra)                              2T(n / 2)
   δ  = min(δ1, δ2)

   Cancellare tutti punti con distanza >δ da L                 O(n)

   Ordinare i punti rimanenti per coordinata y.                O(n log n)

   Scansione punti ordinati rispetto ad y e confrontare
   distanza tra ogni punto ed i successivi 11 punti.           O(n)
   Se una distanza è <δ, aggiornare δ.

   return δ
}
```

Coppia di punti più vicini: complessità algoritmo

Complessità algoritmo.

$$\mathrm{T}(n) \leq 2T(n/2) + O(n \log n) \Rightarrow \mathrm{T}(n) = O(n \log^2 n)$$

Domanda. Possiamo ottenere O(n log n)?

Risposta. Si! Non ordinare punti ogni volta.

- Due liste ordinate all'inizio: rispetto ad x e rispetto ad y
- Ogni chiamata ricorsiva ritorna due liste: tutti i punti ordinati per coordinata y e tutti i punti ordinati per coordinata x.
- Estrazione delle due liste ordinate dalla singola lista già ordinata.

$$T(n) \leq 2T(n/2) + O(n) \Rightarrow \mathrm{T}(n) = O(n \log n)$$

Coppia di punti più vicini: algoritmo O(n log n)

```
Closest-Pair(P)
  Construct P_x and P_y  (O(n log n) time)
  (p_0*, p_1*) = Closest-Pair-Rec(P_x, P_y)

Closest-Pair-Rec(P_x, P_y)
  If |P| ≤ 3 then
    find closest pair by measuring all pairwise distances
  Endif

  Construct Q_x, Q_y, R_x, R_y (O(n) time)
  (q_0*, q_1*) = Closest-Pair-Rec(Q_x, Q_y)
  (r_0*, r_1*) = Closest-Pair-Rec(R_x, R_y)

  δ = min(d(q_0*, q_1*), d(r_0*, r_1*))
  x* = maximum x-coordinate of a point in set Q
  L = {(x, y) : x = x*}
  S = points in P within distance δ of L.

  Construct S_y (O(n) time)
  For each point s ∈ S_y, compute distance from s
     to each of next 15 points in S_y
     Let s, s' be pair achieving minimum of these distances
     (O(n) time)

  If d(s, s') < δ then
     Return (s, s')
  Else if d(q_0*, q_1*) < d(r_0*, r_1*) then
     Return (q_0*, q_1*)
  Else
     Return (r_0*, r_1*)
  Endif
```

P_x = lista P ordinata rispetto a coordinata x

P_y = lista P ordinata rispetto a coordinata y

Q = punti nelle prime $\lceil n/2 \rceil$ posizioni in P_x

R = punti nelle ultime $\lfloor n/2 \rfloor$ posizioni in P_x

Q_x = punti in Q ordinati rispetto ad x

Q_y = punti in Q ordinati rispetto ad y

R_x = punti in R ordinati rispetto ad x

R_y = punti in R ordinati rispetto ad y

5.5 Integer Multiplication

Aritmetica su interi: addizione

Addizione. Dati due interi di n-digit a e b, computare a + b.

- O(n) operazioni su bit.

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

Addizione

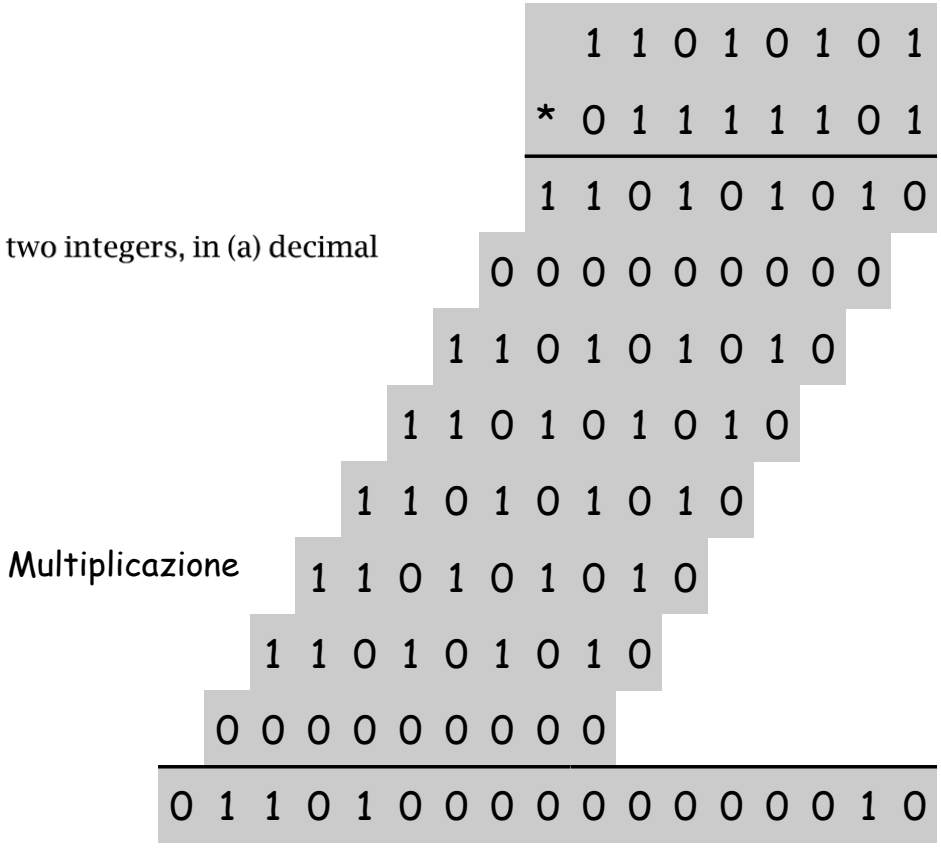
Aritmetica su interi: moltiplicazione

Multiplicazione. Dati due interi di n-digit a e b, computare a × b.

- Soluzione *brute force*: $\Theta(n^2)$ operazioni su bit.

12	1100
× 13	× 1101
36	1100
12	0000
156	1100
	1100
	10011100
(a)	(b)

Figure 5.8 The elementary-school algorithm for multiplying two integers, in (a) decimal and (b) binary representation.



Moltiplicazione Divide-and-Conquer: Warmup

Per moltiplicare due interi di n-digit:

- Multiplicare quattro interi di $\frac{1}{2}$n cifre.
- Addizionare due interi di $\frac{1}{2}$n cifre, e poi "shift" per ottenere risultato.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0
\end{aligned}
$$

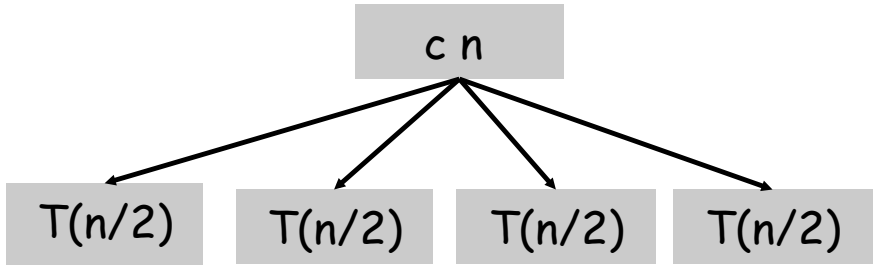
$$
T(n) = \underbrace{4T\left(n/2\right)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n)}_{\text{addizioni, shift}} \Rightarrow T(n) = \Theta(n^2)
$$

assumiamo n potenza di 2

Prova con Albero di Ricorsione

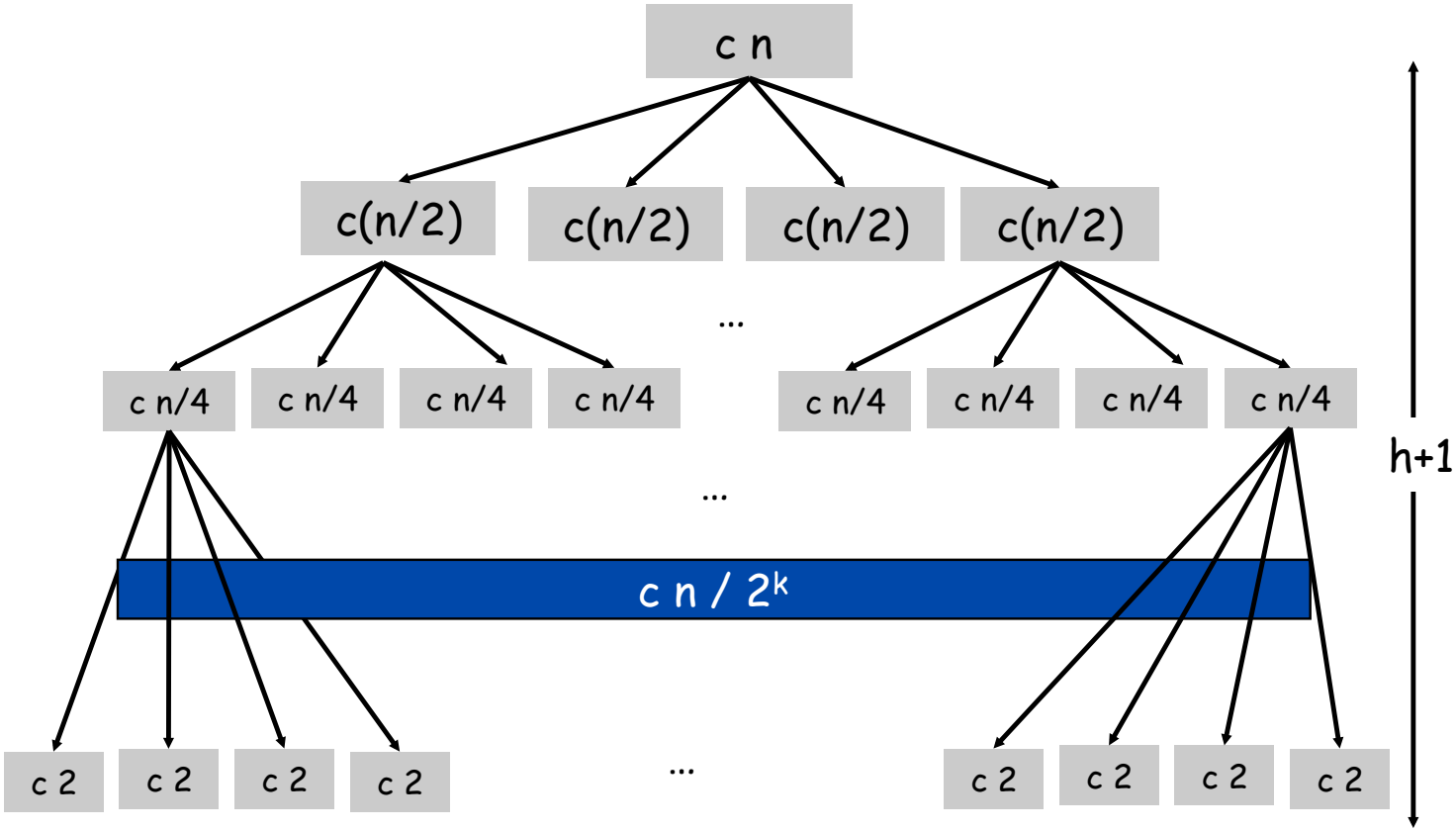
$$T(n) = \underbrace{4T(n/2)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n)}_{\text{addizioni, shift}} \Rightarrow T(n) = \Theta(n^2)$$

T(n)



Prova con Albero di Ricorsione

$$T(n) = \underbrace{4T(n/2)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n)}_{\text{addizioni, shift}} \Rightarrow T(n) = \Theta(n^2)$$

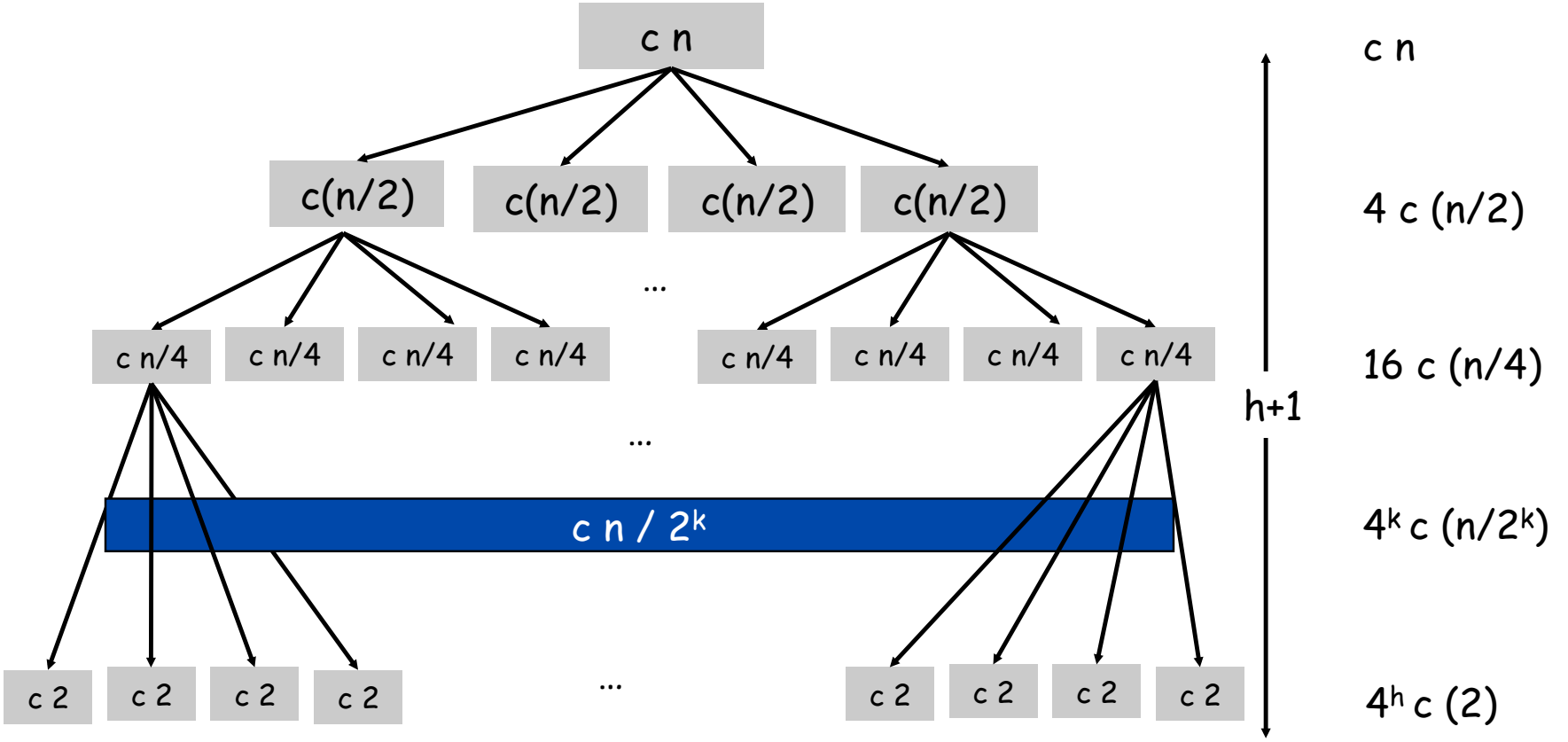


Altezza albero h?

$$c\sum_{i=0}^{h-1} 4^i \frac{n}{2^i} \le cn\sum_{i=0}^{h-1} 2^i = cn(2^h - 1)$$

Prova con Albero di Ricorsione

$$T(n) = \underbrace{4T(n/2)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n)}_{\text{addizioni, shift}} \Rightarrow T(n) = \Theta(n^2)$$



Altezza albero h? $n / 2^h = 2$ cioè $h = \log_2 n - 1$

$$c\sum_{i=0}^{h-1} 4^i \frac{n}{2^i} \le cn\sum_{i=0}^{h-1} 2^i = cn(2^h - 1)$$

Moltiplicazione: Algoritmo di Karatsuba

Moltiplicazione di due interi con n cifre:

- Addizionare due interi con $\frac{1}{2}$n cifre.
- Multiplicare tre interi con $\frac{1}{2}$n cifre.
- Addizione, sottrazione, e shift di interi con $\frac{1}{2}$n cifre per ottenere risultato.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
&= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \big((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0 \big) + x_0 y_0 \\
&\qquad\quad A \qquad\qquad\quad B \qquad\qquad A \quad C \qquad C
\end{aligned}
$$

Moltiplicazione: Algoritmo di Karatsuba

Moltiplicazione di due interi con n cifre:

- Addizionare due interi con $\frac{1}{2}$n cifre.
- Multiplicare tre interi con $\frac{1}{2}$n cifre.
- Addizione, sottrazione, e shift di interi con $\frac{1}{2}$n cifre per ottenere risultato.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
&= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \big((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0 \big) + x_0 y_0 \\
&\qquad\quad A \qquad\qquad\quad B \qquad\quad A \quad C \qquad C
\end{aligned}
$$

```
Recursive-Multiply(x,y):
  Write x = x_1 · 2^{n/2} + x_0
        y = y_1 · 2^{n/2} + y_0
  Compute x_1 + x_0 and y_1 + y_0
  p = Recursive-Multiply(x_1 + x_0, y_1 + y_0)
  x_1 y_1 = Recursive-Multiply(x_1, y_1)
  x_0 y_0 = Recursive-Multiply(x_0, y_0)
  Return x_1 y_1 · 2^n + (p - x_1 y_1 - x_0 y_0) · 2^{n/2} + x_0 y_0
```

Moltiplicazione: Algoritmo di Karatsuba

Moltiplicazione di due interi con n cifre:

- Addizionare due interi con $\frac{1}{2}$n cifre.
- Multiplicare tre interi con $\frac{1}{2}$n cifre.
- Addizione, sottrazione, e shift di interi con $\frac{1}{2}$n cifre per ottenere risultato.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
&= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \big((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0 \big) + x_0 y_0 \\
&\qquad\quad A \qquad\qquad\quad B \qquad\qquad A \quad C \qquad\quad C
\end{aligned}
$$

Teorema. [Karatsuba-Ofman, 1962] La complessità dell' algoritmo di moltiplicazione di due interi con n cifre richiede $O(n^{1.585})$ operazioni sui bit.

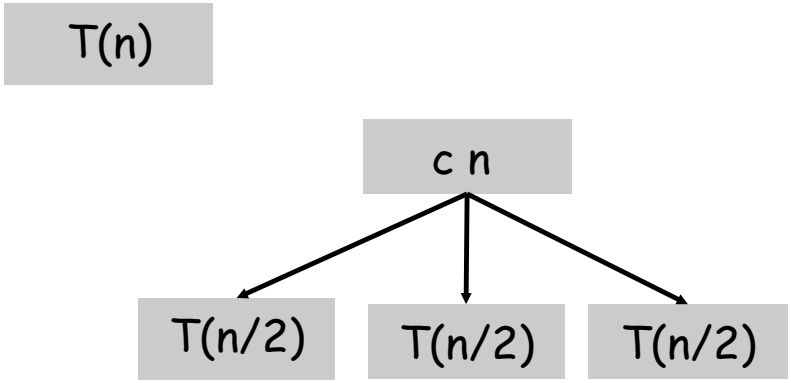
Prova.

$$
\begin{aligned}
T(n) &\le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1+\lceil n/2 \rceil)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n)}_{\text{addizioni, suttrazioni, shift}} \\
&\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585})
\end{aligned}
$$

Prova con Albero di Ricorsione

$$T(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1+\lceil n/2 \rceil)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n)}_{\text{addizioni, suttrazioni, shift}}$$

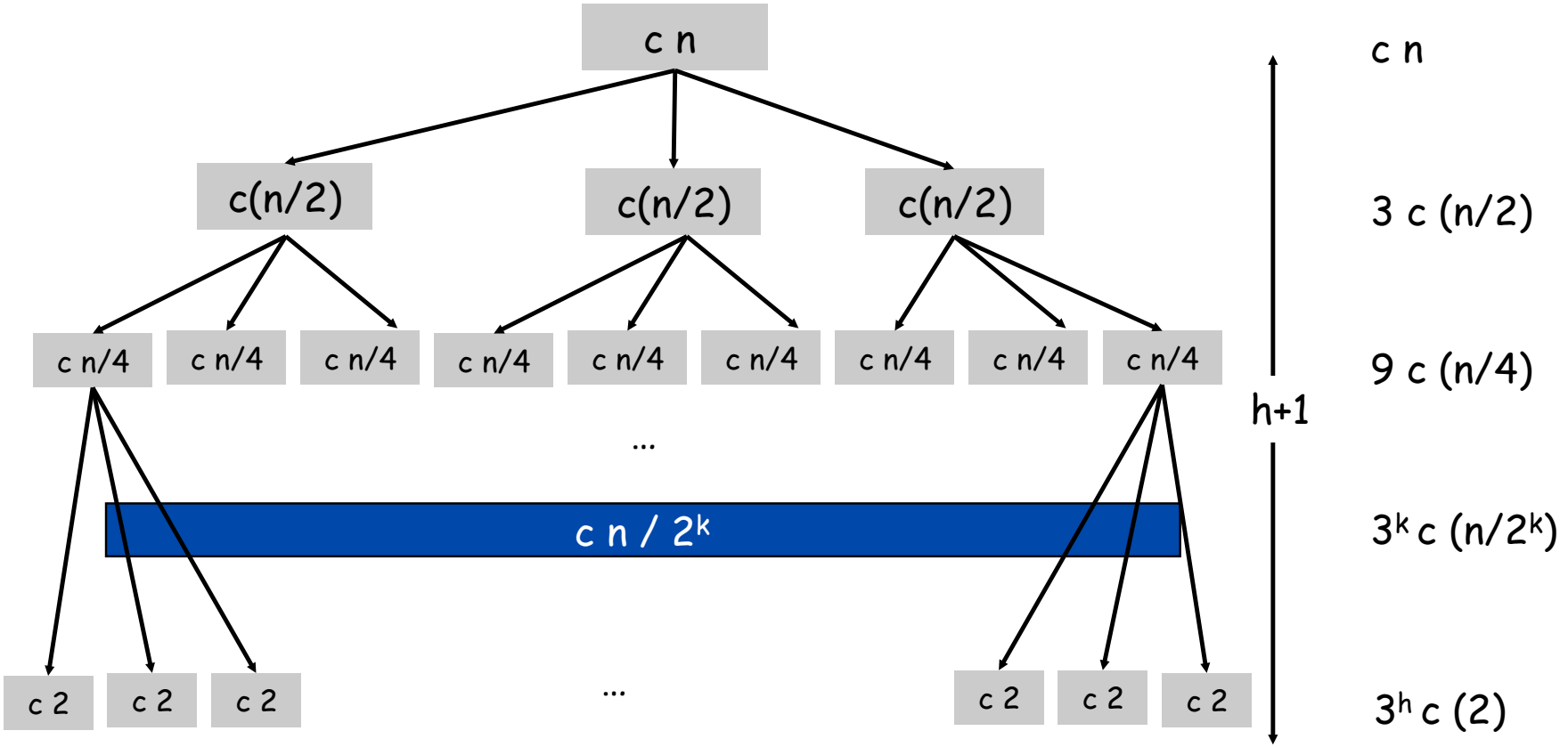
$$\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585})$$



Prova con Albero di Ricorsione

$$T(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1+\lceil n/2 \rceil)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n)}_{\text{addizioni, suttrazioni, shift}}$$

$$\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585})$$



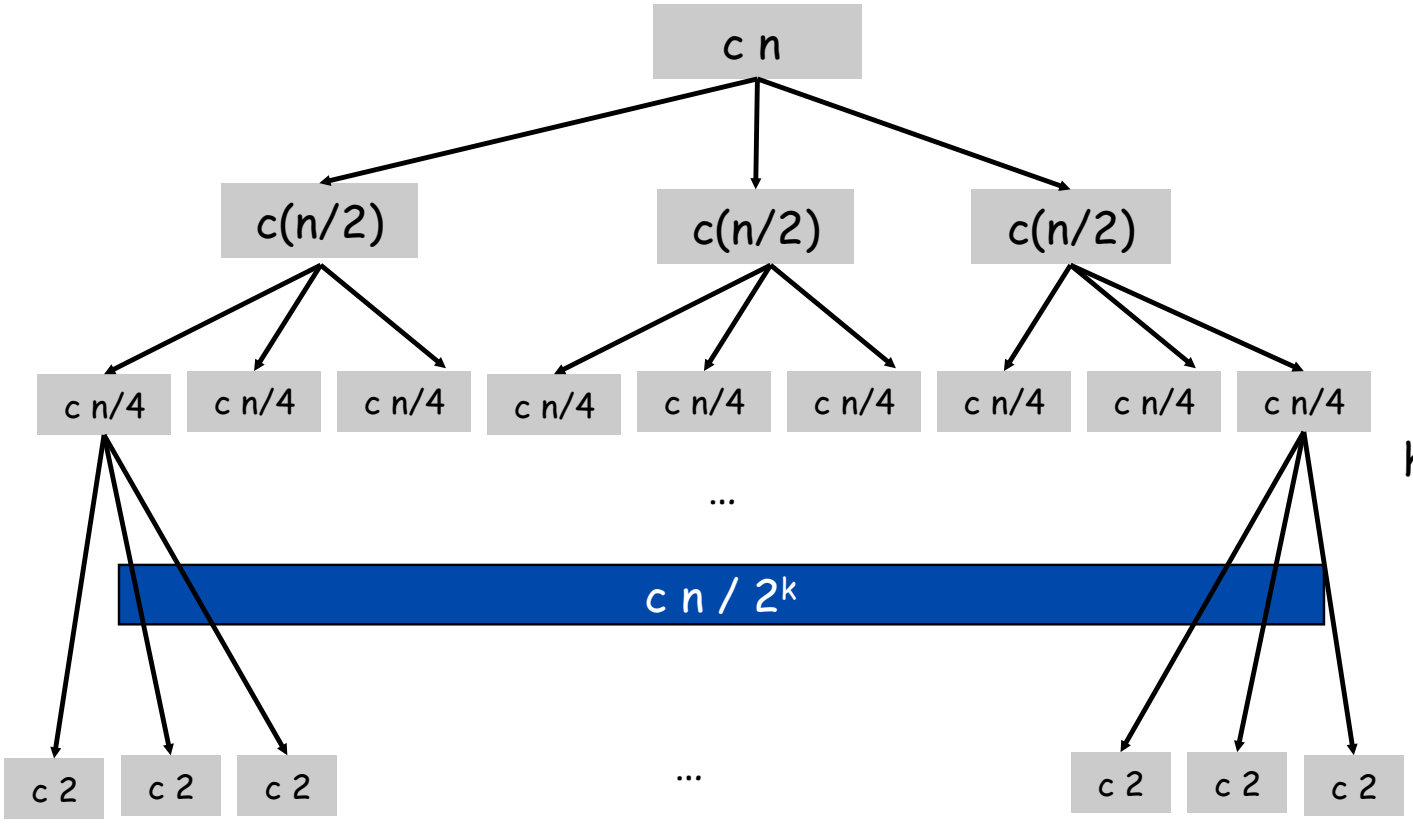
Altezza albero h? $n / 2^h = 2$ cioè $h = \log_2 n - 1$

$$c\sum_{i=0}^{h-1} 3^i \frac{n}{2^i} \le cn\sum_{i=0}^{h-1}\left(\frac{3}{2}\right)^i = O\left(n\left(\frac{3}{2}\right)^h\right)$$

$$n \cdot \left(\frac{3}{2}\right)^{\log_2 n} = n \cdot n^{\log_2(3/2)} = n \cdot n^{\log_2 3 - 1} = n^{\log_2 3}$$

Relazione ricorrenza: caso generale con q>2

$$T(n) \le qT(n/2) + cn$$
$$\Rightarrow T(n) = O(n^{\log_2 q})$$



Altezza albero h? $n / 2^h = 2$ cioè $h = \log_2 n - 1$

$$n \cdot \left(\frac{q}{2}\right)^{\log_2 n} = n \cdot \frac{q^{\log_2 n}}{2^{\log_2 n}} = q^{\log_2 n}$$

$$c\sum_{i=0}^{h-1} q^i \frac{n}{2^i} \le cn\sum_{i=0}^{h-1}\left(\frac{q}{2}\right)^i = O\left(n\left(\frac{q}{2}\right)^h\right)$$

Matrix Multiplication

Moltiplicazione di matrici

Moltiplicazione di matrici. Date due matrici n-per-n A e B, computa $C = AB$.

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

Algoritmo di forza bruta. $\Theta(n^3)$ operazioni aritmetiche.

```
for i ← 1 to n
    for j ← 1 to n
        c_ij ← 0
        for k ← 1 to n
            c_ij ← c_ij + a_ik · b_kj
```

Domanda. E' possibile fare meglio dell'algoritmo di forza bruta?

Moltiplicazione di matrici: prima idea

Divide-and-conquer.

- Divide: partizionare A e B in blocchi $\frac{1}{2}$n-per-$\frac{1}{2}$n.
- Conquer: moltiplicare 8 blocchi $\frac{1}{2}$n-by-$\frac{1}{2}$n ricorsivamente.
- Combine: addizionare prodotti usando 4 addizioni tra matrici.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

Complessità?

Moltiplicazione di matrici: prima idea

Divide-and-conquer.

- Divide: partizionare A e B in blocchi $\frac{1}{2}n$-per-$\frac{1}{2}n$.
- Conquer: moltiplicare 8 blocchi $\frac{1}{2}n$-by-$\frac{1}{2}n$ ricorsivamente.
- Combine: addizionare prodotti usando 4 addizioni tra matrici.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

Complessità?

$$T(n) = \underbrace{8T(n/2)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n^2)}_{\text{addizioni, costruzione sottomatrici}} \Rightarrow T(n) = \Theta(n^3)$$

Moltiplicazione di matrici: algoritmo efficiente, idea di base

Idea di base. Moltiplicare blocchi 2-per-2 di matrici con 7 moltiplicazioni.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

- 7 moltiplicazioni.
- 18 = 10 + 8 addizioni (o sottrazioni).

Moltiplicazione di matrici: algoritmo efficiente

Moltiplicazione efficiente di matrici. (Strassen, 1969)

- Divide: partizionare A e B in blocchi $\frac{1}{2}$n-per-$\frac{1}{2}$n.
- Compute: 14 matrici $\frac{1}{2}$n-per-$\frac{1}{2}$n mediante 10 addizioni di matrici.
- Conquer: moltiplicare 7 matrici $\frac{1}{2}$n-per-$\frac{1}{2}$n ricorsivamente.
- Combine: 7 prodotti in 4 termini usando 8 addizioni di matrici.

Analisi.

- Assumiamo n potenza di 2.
- Sia T(n) il numero di operazioni aritmetiche.

$$\mathrm{T}(n) = \underbrace{7T\left(n/2\right)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n^2)}_{\text{addizioni, sottrazioni}} \Rightarrow \mathrm{T}(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

Moltiplicazione di matrici: algoritmi efficienti in teoria

D. Moltiplicazione due matrici 2-per-2 con solo 7 moltiplicazioni scalari?
R. Si! [Strassen, 1969] $\Theta(n^{\log_2 7}) = O(n^{2.81})$

D. Moltiplicazione due matrici 2-per-2 con solo 6 moltiplicazioni scalari?
R. Impossibile. [Hopcroft and Kerr, 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$

D. Due matrici 3-per-3 con solo 21 moltiplicazioni scalari?
R. Impossibile. $\Theta(n^{\log_3 21}) = O(n^{2.77})$

D. Due matrici 70-by-70 con solo 143.640 moltiplicazioni scalari?
R. Si! [Pan, 1980] $\Theta(n^{\log_{70} 143640}) = O(n^{2.80})$

Miglior risultato finora. $O(n^{2.376})$ [Coppersmith-Winograd, 1987.]

Congettura. $O(n^{2+\varepsilon})$ per ogni $\varepsilon > 0$.

I miglioramenti teorici a Strassen sono sempre meno pratici.

Quicksort

Divide-and-Conquer

Divide-and-conquer.

- Dividere il problema in diverse parti.
- Risolvere ogni parte ricorsivamente.
- Combinare le soluzioni dei sottoproblemi in soluzioni globali.

Uso più comune.

- Dividere il problema di taglia n in due parti.
- Risolvere ogni parte ricorsivamente.
- Combinare le soluzioni dei sottoproblemi in soluzioni globali in tempo lineare.

Ordinamento

Ordinamento. Dati n elementi, disporli in ordine crescente.

Applicazioni ovvie:
- List files in a directory.
- Organize an MP3 library.
- List names in a phone book.
- Display Google PageRank results.

Problemi più semplici dopo ordinamento.
- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

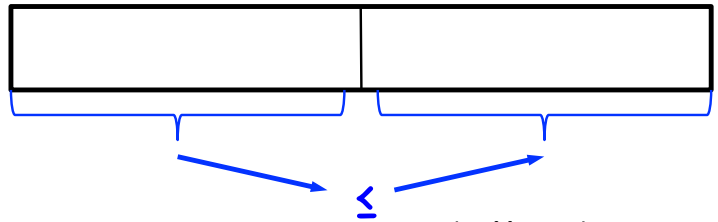
Applicazioni non-ovvie:
- Data compression.
- Computer graphics.
- Interval scheduling.
- Computational biology.
- Minimum spanning tree.
- Supply chain management.
- Simulate a system of particles.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.
- . . .

Abbiamo già visto il Mergesort come algoritmo di ordinamento (complessità $O(n \log n)$).

Quicksort

Divide-and-conquer.

- Divide: dividi lista in due parti, in modo che ogni elemento della prima parte sia ≤ di ogni elemento della seconda parte.



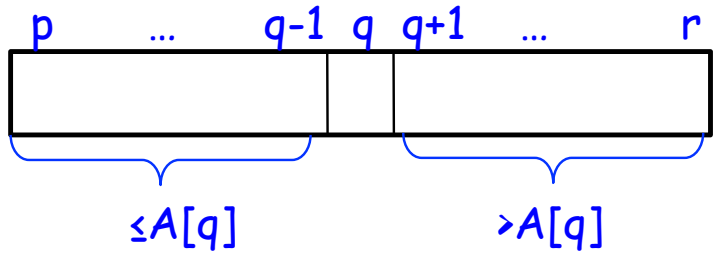
- Conquer: ordina ricorsivamente ognuna delle due parti.
- Combine: sono già ordinati.

Quicksort

Divide-and-conquer.

- Divide: partiziona $A[p...r]$ in due parti $A[p...q-1]$ e $A[q+1...r]$ in modo che
  - ogni elemento di $A[p...q-1]$ sia $\leq A[q]$ ed
  - ogni elemento di $A[q+1...r]$ sia $> A[q]$.

  (Il valore q è scelto dalla procedura di partizione.)



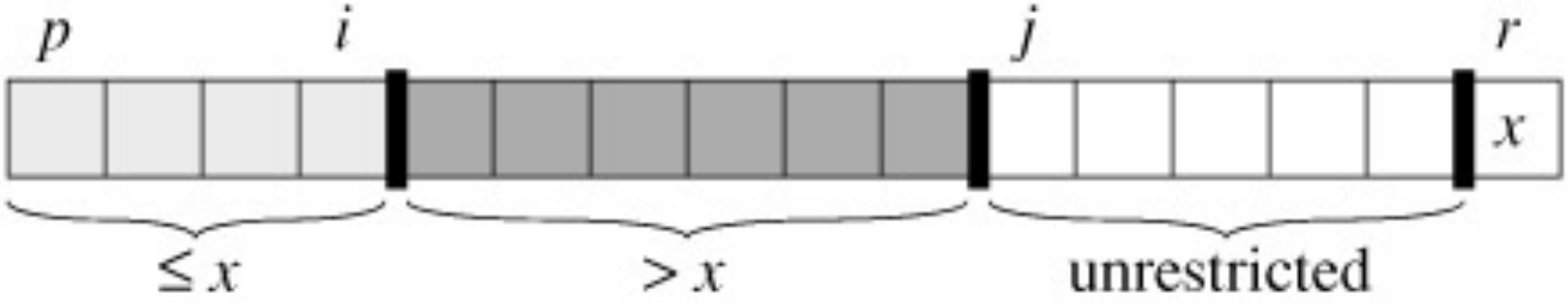
- Conquer: ordina ricorsivamente ognuna delle due parti.
- Combine: sono già ordinati.

Quicksort: algoritmo

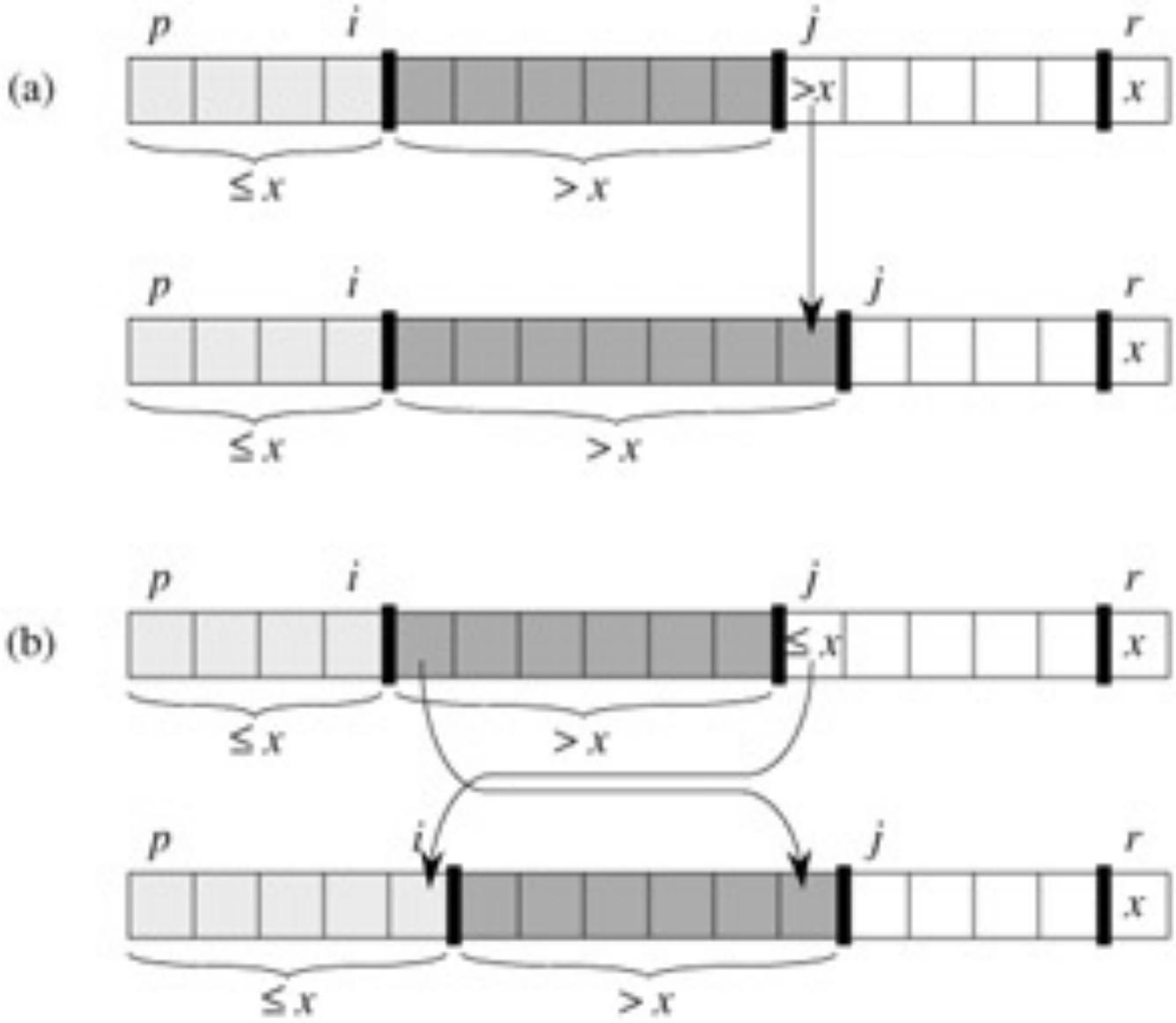
```
Quicksort (A, p, r)
   if p < r then q ← Partiziona (A, p, r)
                    Quicksort (A, p, q - 1)
                    Quicksort (A, q + 1, r)
```

Per ordinare l'array A, chiamata iniziale Quicksort (A, 1, length[A]).

Partiziona: idea



Partiziona: idea



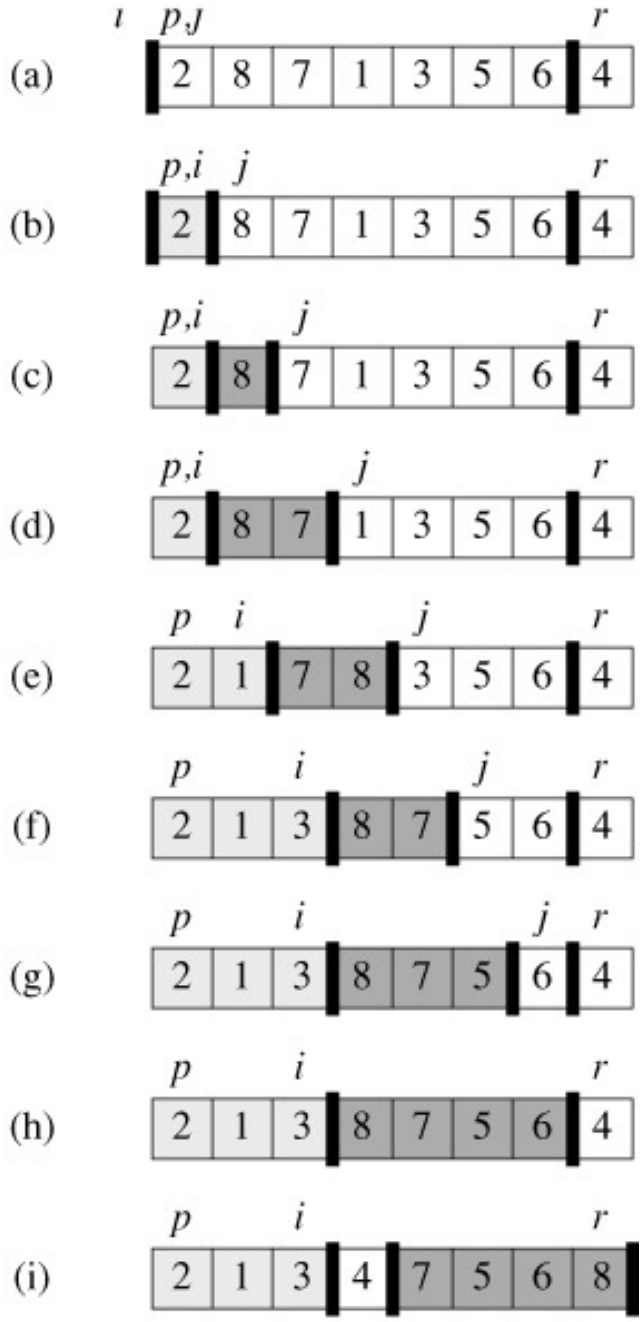
Partiziona: algoritmo

```
Partiziona (A, p, r)
     x ← A[r]
     i ← p - 1
     for j ← p to r - 1 do
          if A[j] ≤ x then i ← i + 1
                           scambia A[i] ↔ A[j]
scambia A[i + 1] ↔ A[r]
return  i + 1
```

Partiziona: esempio

Esempio.

```
Partiziona (A, p, r)
    x ← A[r]
    i ← p - 1
    for j ← p to r - 1 do
        if A[j] ≤ x then i ← i + 1
                         scambia A[i] ↔ A[j]
scambia A[i + 1] ↔ A[r]
return  i + 1
```



Partiziona: complessità

```
Partiziona (A, p, r)
     x ← A[r]
     i ← p - 1
     for j ← p to r - 1 do
          if A[j] ≤ x then i ← i + 1
                           scambia A[i] ↔ A[j]
scambia A[i + 1] ↔ A[r]
return  i + 1
```

Complessità: $\Theta(n)$, dove $n = r - p + 1$

Partizionamento

Partizionamento: Analizziamo tre casi

- Caso peggiore
- Caso migliore
- Caso medio

Partizionamento: caso peggiore

Caso peggiore. Partizione sbilanciata con sottoproblemi di n-1 e 0 elementi

$$T(n) = T(n - 1) + T(0) + \Theta(n)$$
$$= T(n - 1) + \Theta(n).$$

Soluzione $T(n) = \Theta(n^2)$

Partizionamento: caso migliore

Caso migliore. Partizione bilanciata con sottoproblemi di $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil - 1$

$$T(n) = 2\ T(n/2) + \Theta(n)$$

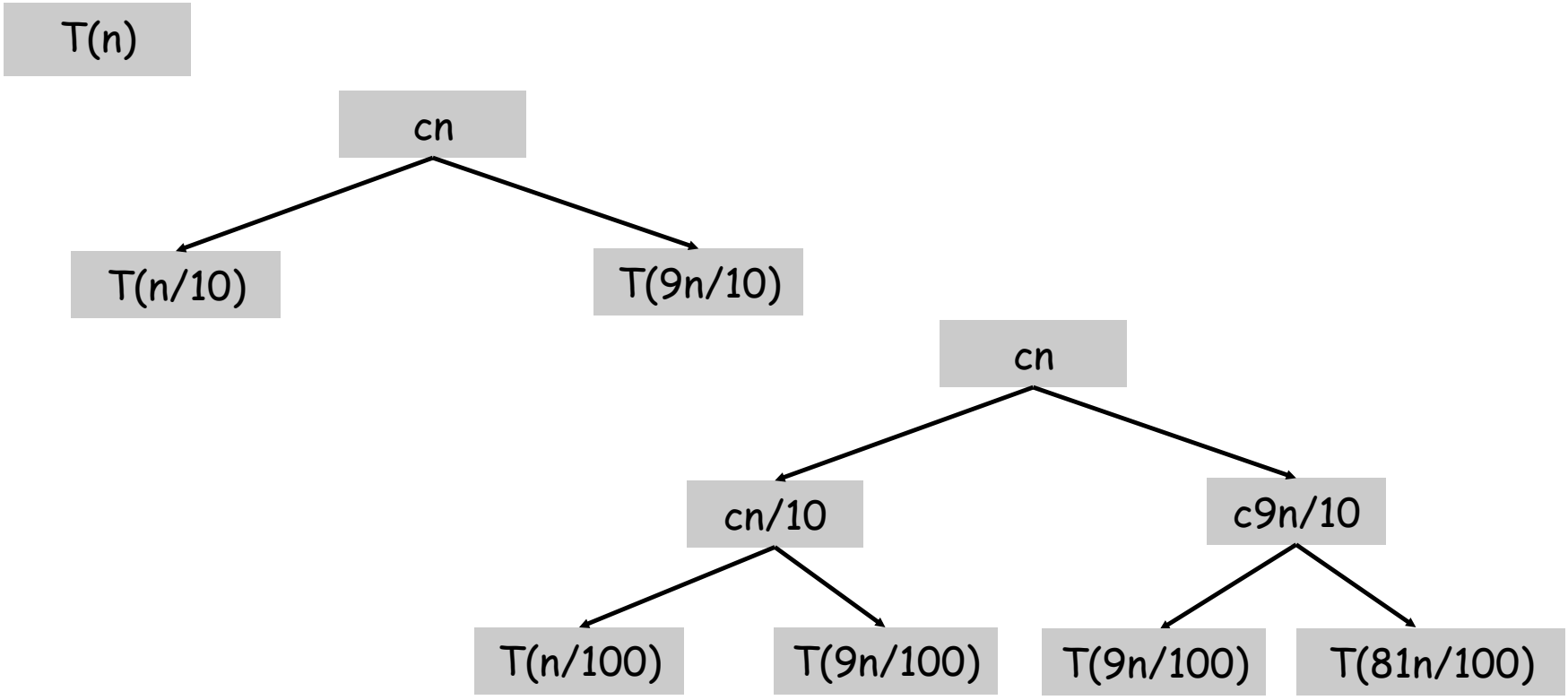
Soluzione $T(n) = \Theta(n \log n)$

(è la stessa relazione del Mergesort)

Partizione bilanciata

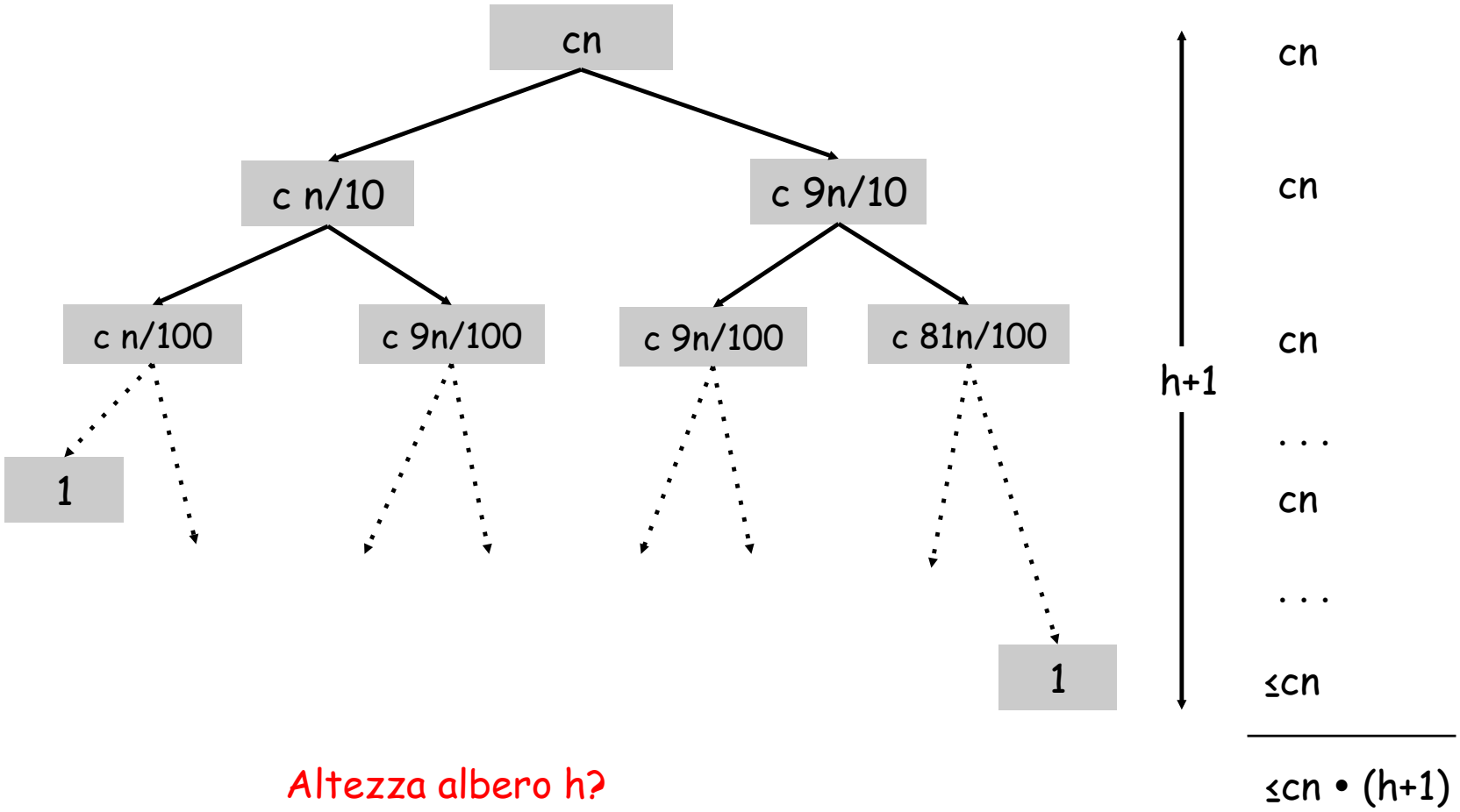
Partizione bilanciata con sottoproblemi di 9n/10 e n/10 elementi

$$T(n) \leq T(9n/10) + T(n/10) + cn$$



Prova con Albero di Ricorsione

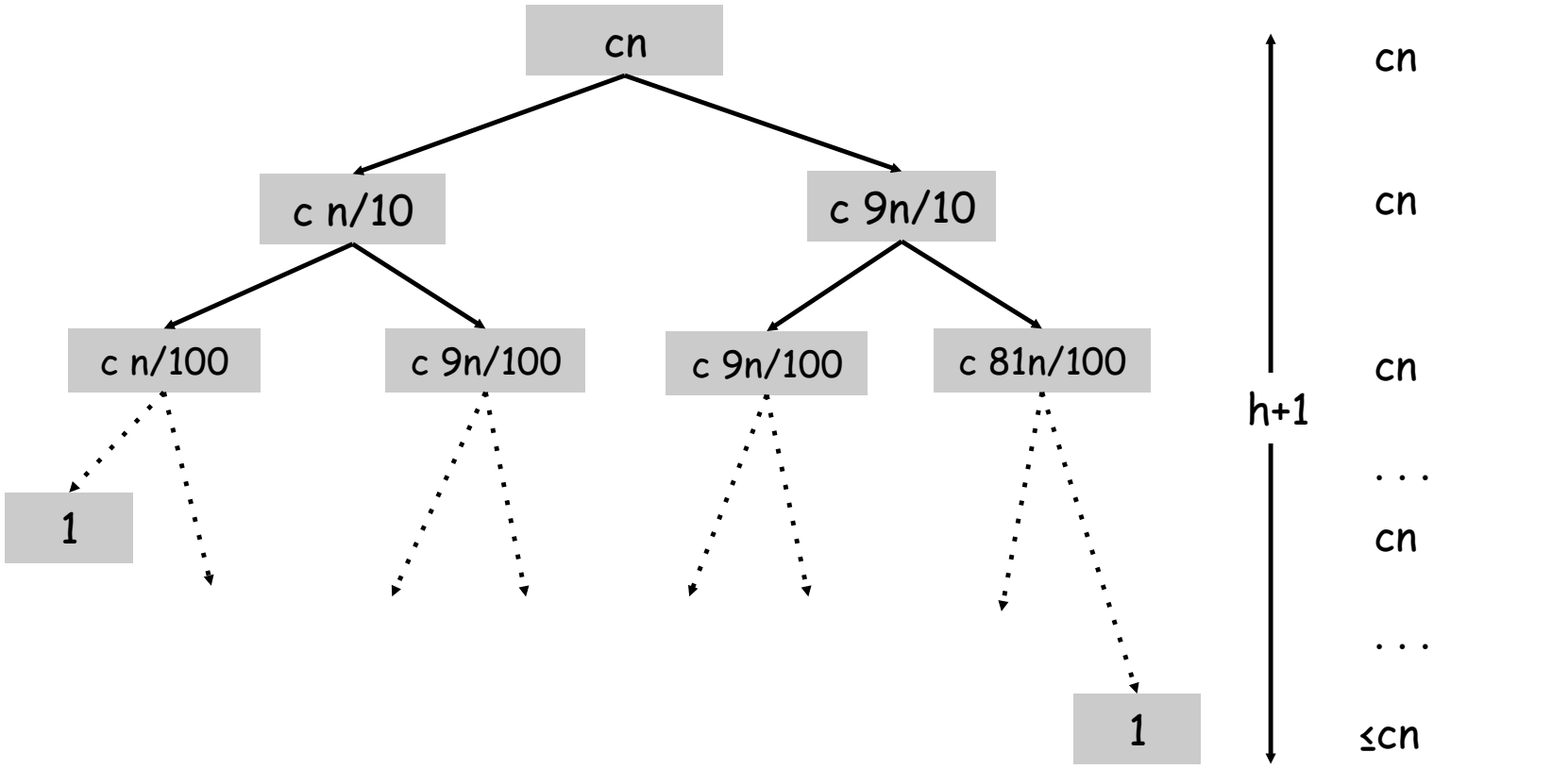
Risoluzione $T(n) = T(9n/10) + T(n/10) + cn$



Altezza albero h?

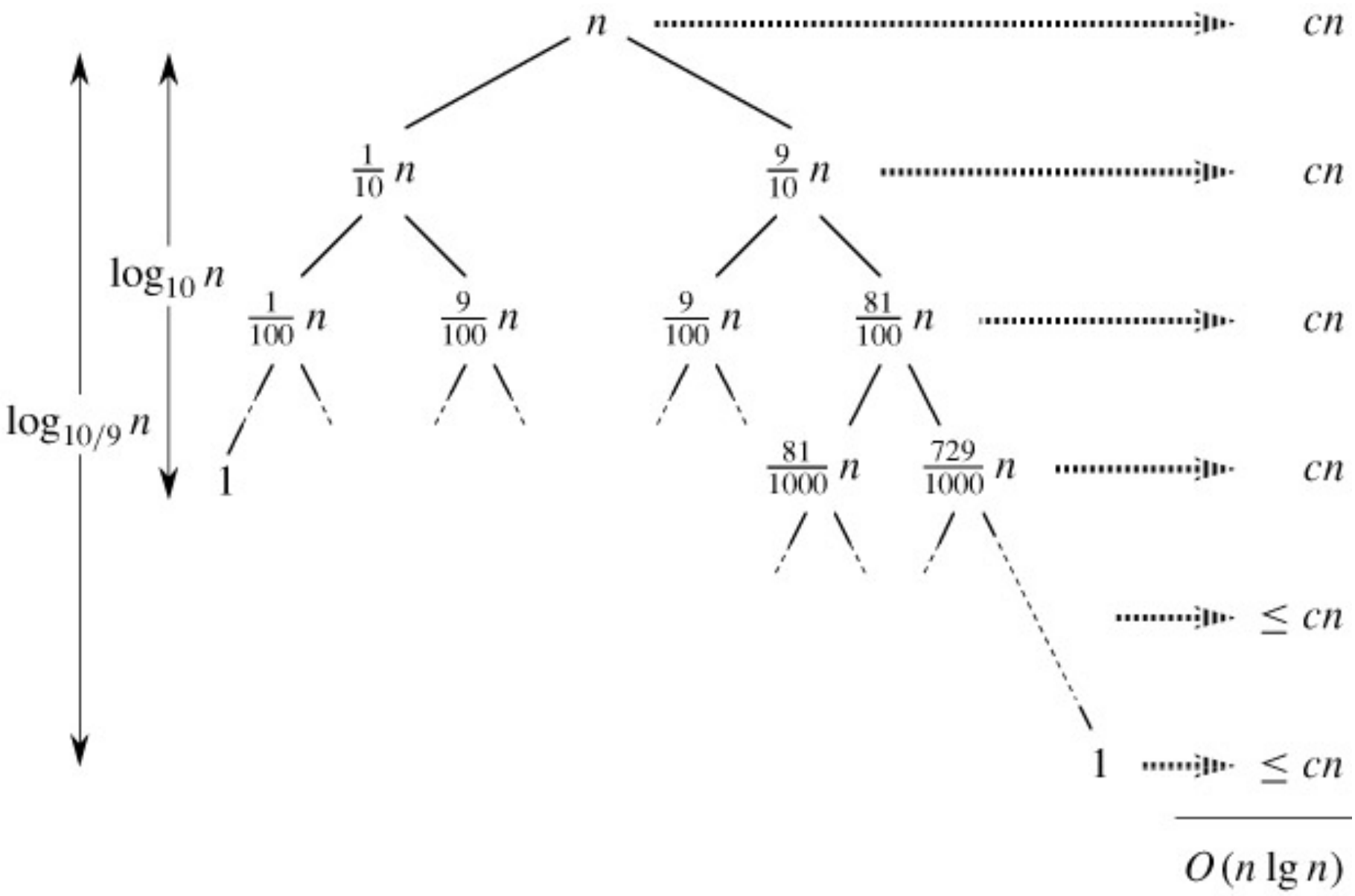
Prova con Albero di Ricorsione

Risoluzione $T(n) = T(9n/10) + T(n/10) + cn$



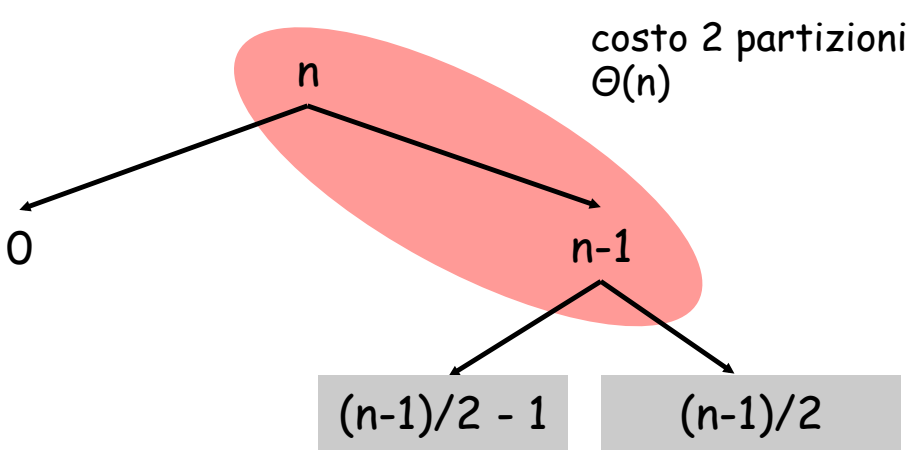
Altezza albero h? $(9/10)^h n = 1$ cioè $h = \log_{10/9} n$

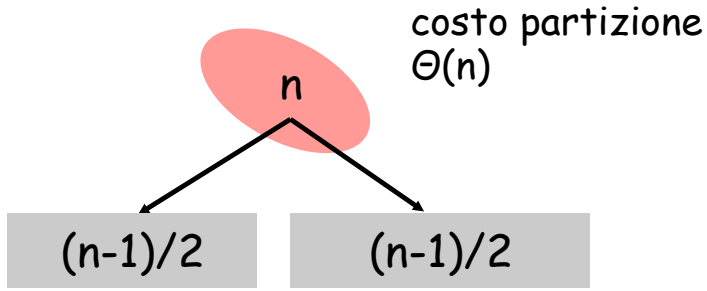
Partizione bilanciata



Partizioni

Partizione sbilanciata seguita da partizione bilanciata





Caso medio:

Intuitivamente è più vicino al caso migliore che al caso peggiore.

Complessità O(n log n)

Quicksort: analisi caso peggiore

$$T(n) = \max_{0 \le q \le n-1}(T(q) + T(n-q-1)) + \Theta(n)$$

Proviamo per induzione che $T(n) \le cn^2$ per qualche costante c.

Sostituendo:

$$T(n) \le \max_{0 \le q \le n-1}(cq^2 + c(n-q-1)^2) + \Theta(n)$$
$$\le c \cdot \max_{0 \le q \le n-1}(q^2 + (n-q-1)^2) + \Theta(n)$$

La funzione $f(q) = q^2 + (n-q-1)^2$ ha un massimo nell'intervallo $0 \le q \le n-1$ in un punto estremo, dato che la derivata seconda $f''(q) > 0$

$$\max_{0 \le q \le n-1}(q^2 + (n-q-1)^2) \le (n-1)^2 = n^2 - 2n + 1$$

Quindi,

$T(n) \le cn^2 - c(2n-1) + \Theta(n)$

$\le cn^2$

Dato che si può scegliere la costante c grande abbastanza affinchè il termine $c(2n-1)$ domina il termine $\Theta(n)$.

Quindi, $T(n) = O(n^2)$.

Esercizio 7.2-3

Se gli elementi sono distinti e sono già ordinati in senso decrescente, la complessità del Quicksort è $\Theta(n^2)$.

Esercizio 7.2-5

Supponiamo che la partizione ad ogni livello è proporziale ad 1-a ed a, dove $0 < a \leq 1/2$ è una costante. Mostrare che la profondità minima di una foglia nell'albero della ricorsione è approssimativamente $- \lg n / \lg a$ e la massima profondità è approssimativamente $- \lg n / \lg (1-a)$. (Trascurare l'arrotondamento agli interi.)

Quicksort: versione randomizzata

```
Partiziona_random (A, p, r)
      i ← random(p,r)
      scambia A[i] ↔ A[r]
      return Partiziona(A, p, r)
```

```
Quicksort_random (A, p, r)
   if p < r then q ← Partiziona_random (A, p, r)
                     Quicksort_random (A, p, q - 1)
                     Quicksort_random (A, q + 1, r)
```

Quicksort: Cenno storico

La procedura del quicksort è dovuta ad Hoare:
C. R. Hoare. Quicksort. Computer Journal, 5(1): 10–15, 1962.
La versione di Hoare appare nel Problema 7-1.

La procedura Partizione che abbiamo visto è dovuta a N. Lomuto.

Esercizi

- Potenze di un numero
- Numeri di Fibonacci
- Massima Sottosequenza
- Massimi in 2-D
- Defective Chessboards

Esercizi da risolvere in forma completa a casa

Esercizio: Potenze di un numero

Input: $a, n \in \mathbb{N}$

Calcolare: a^n

Algoritmo naïve: $\Theta(n)$

Algoritmo Divide and Conquer $\Theta(\log n)$

Esercizio: Potenze di un numero

Input: a, n ∈ N
Calcolare: a^n

Algoritmo naïve: Θ(n)
Algoritmo Divide and Conquer Θ(log n)

$$a^n = \begin{cases} (a^{n/2})^2 & \text{se } n \text{ pari} \\ (a^{(n-1)/2})^2 a & \text{se } n \text{ dispari} \end{cases}$$

Complessità algoritmo Divide and Conquer:
Relazione ricorrenza running time T(n) = T(n/2) + Θ(1)
Soluzione T(n) = Θ(log n)

Esercizio: Potenze di un numero

Soluzione relazione ricorrenza con metodo di sostituzione

$T(n) = T(n/2) + c$ con $n = 2^k$

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

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

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

$= T(n/8) + 3c$

$= T(n/16) + c + 3c$

$= T(n/16) + 4c$

$= \ldots$

$= T(n/2^k) + kc$

$= \Theta(k)$

$= \Theta(\log n)$

da $\frac{n}{2^k} = 1$

segue che $k = \log n$

Esercizio: Numeri di Fibonacci

Input: $n \in \mathbb{N}^+$

Calcolare: F_n

$$F_n = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ F_{n-1} + F_{n-2} & \text{se } n \geq 2 \end{cases}$$

0 1 1 2 3 5 8 13 21 34 ...

Algoritmo naïve: $\Theta(n)$

Algoritmo Divide and Conquer $\Theta(\log n)$

Esercizio: Numeri di Fibonacci

Input: $n \in N$

Calcolare: F_n

$$F_n = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ F_{n-1} + F_{n-2} & \text{se } n \geq 2 \end{cases}$$

Suggerimento per algoritmo Divide and Conquer $\Theta(\log n)$

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

- Provare la relazione (per induzione su n)
- Mostrare come realizzare un algoritmo Divide and Conquer $\Theta(\log n)$

Esercizio: Massima Sottosequenza

Input: sequenza A[1], A[2], …, A[n]

Calcolare: sottosequenza con somma massima

elementi consecutivi

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	*-3*	*-25*	*20*	*-3*	*-16*	*-23*	*18*	*20*	*-7*	*12*	*-5*	*-22*	*15*	*-4*	*7*

20 18+20-7+12 = 43 15-4+7 = 18

Algoritmo naïve: $\Theta(n^2)$

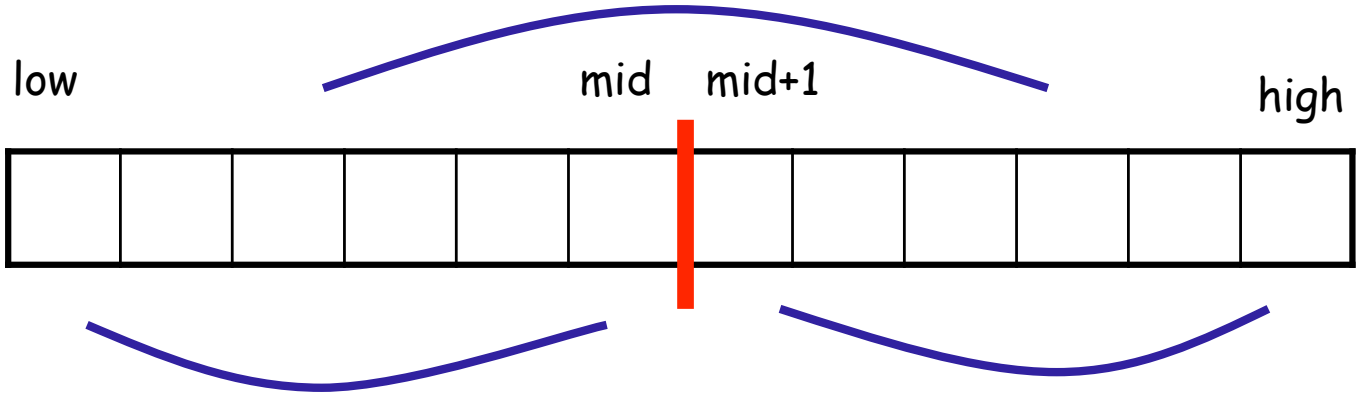
Algoritmo Divide and Conquer $\Theta(n \log n)$

Esercizio: Massima Sottosequenza

Input: sequenza A[1], A[2], …, A[n]
Calcolare: sottosequenza con somma massima

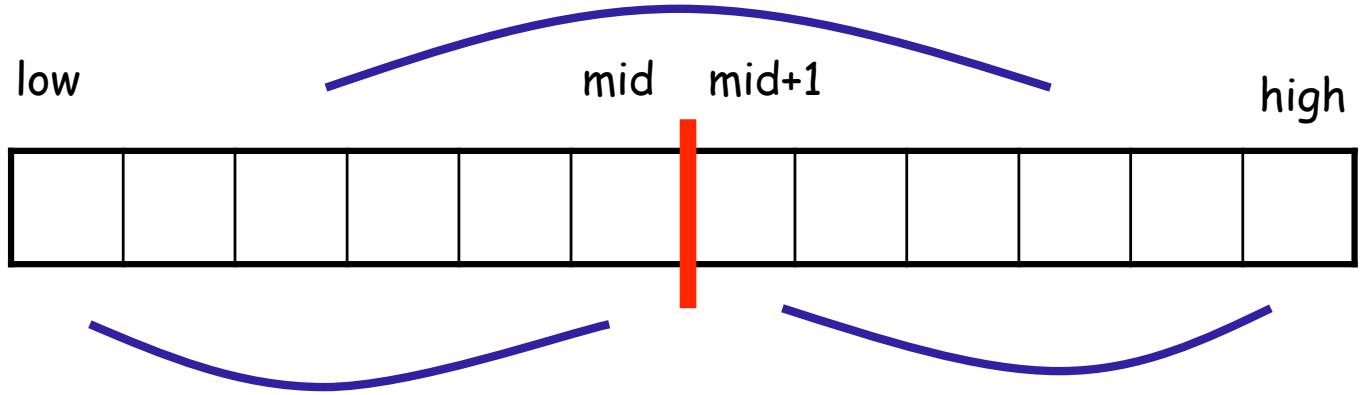
Divide-and-conquer.

- Divide: dividere sequenza in due parti della stessa grandezza.
- Conquer: risolvere ricorsivamente il problema in ogni parte.
- Combine: calcolare la sottosequenza con somma massima che contiene il punto di mezzo (cioè con inizio e fine in parti diverse), e restituire il massimo delle 3 somme.



Esercizio: Massima Sottosequenza

- Combine: calcolare la sottosequenza con somma massima che contiene il punto di mezzo (cioè con inizio e fine in parti diverse), e restituire il massimo delle 3 somme.




```
leftsum ← A[mid]
sum ← A[mid]
maxleft ← mid
for i = mid-1 downto low
    sum ← sum + A[i] ;
    if sum > leftsum
          leftsum ← sum
          maxleft ← i
```

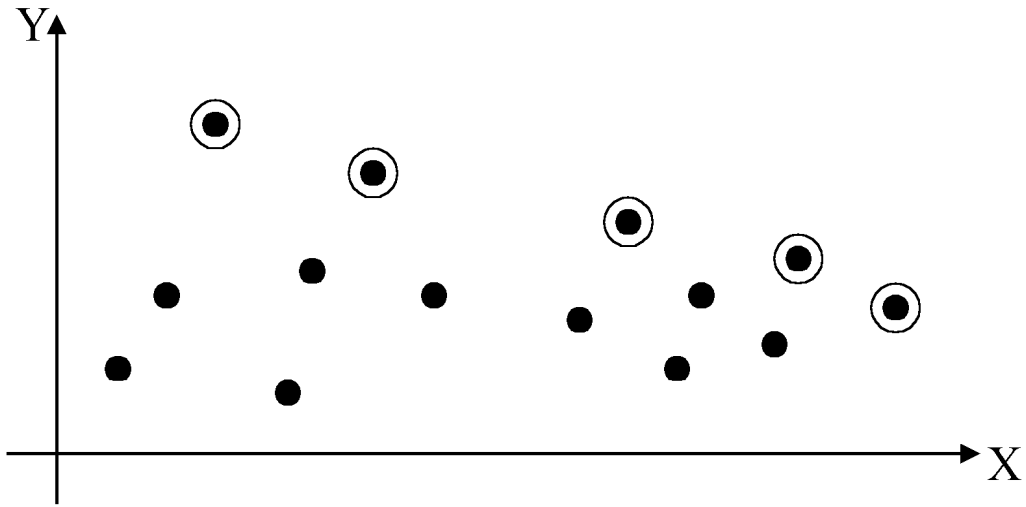
```
rightsum ← A[mid+1]
sum ← A[mid+1]
maxright ← mid+1
for j = mid+1 to high
   sum ← sum + A[j]
     if sum > rightsum
           rightsum ← sum
           maxright ← j
```

Massima somma per sottosequenza che contiene A[mid] e A[mid+1] : leftsum + rightsum

Esercizio: Massimi in 2-D

Input: insieme di punti in un piano
Calcolare: punti di massimo

Un punto (x_1,y_1) domina (x_2,y_2) se $x_1 > x_2$ e $y_1 > y_2$.
Un punto è un massimo se non è dominato da altri punti.



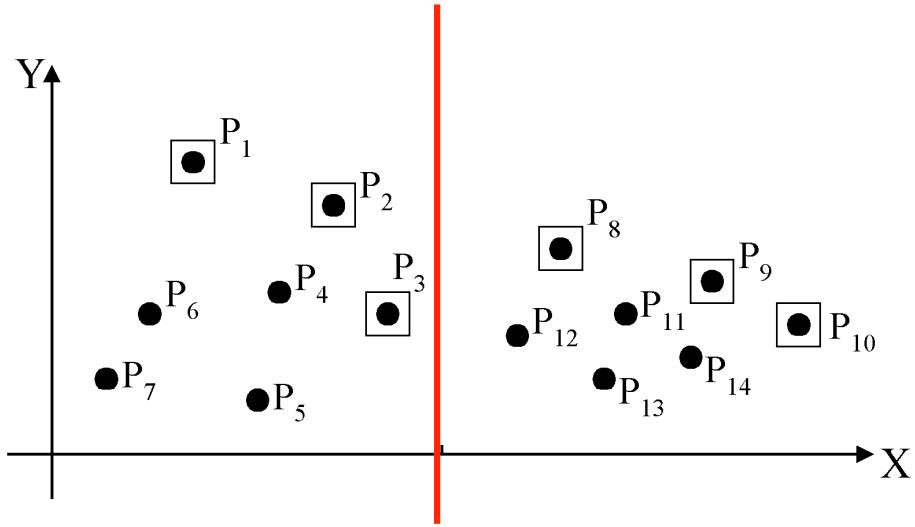
Algoritmo naïve: $\Theta(n^2)$
Algoritmo Divide and Conquer $\Theta(n \log n)$

Esercizio: Massimi in 2-D

Input: insieme di punti in un piano
Calcolare: punti di massimo

Divide-and-conquer.

- Divide: dividere punti in due parti della stessa grandezza mediante una linea verticale.
- Conquer: risolvere ricorsivamente il problema in ognuna delle 2 parti.
- Combine:

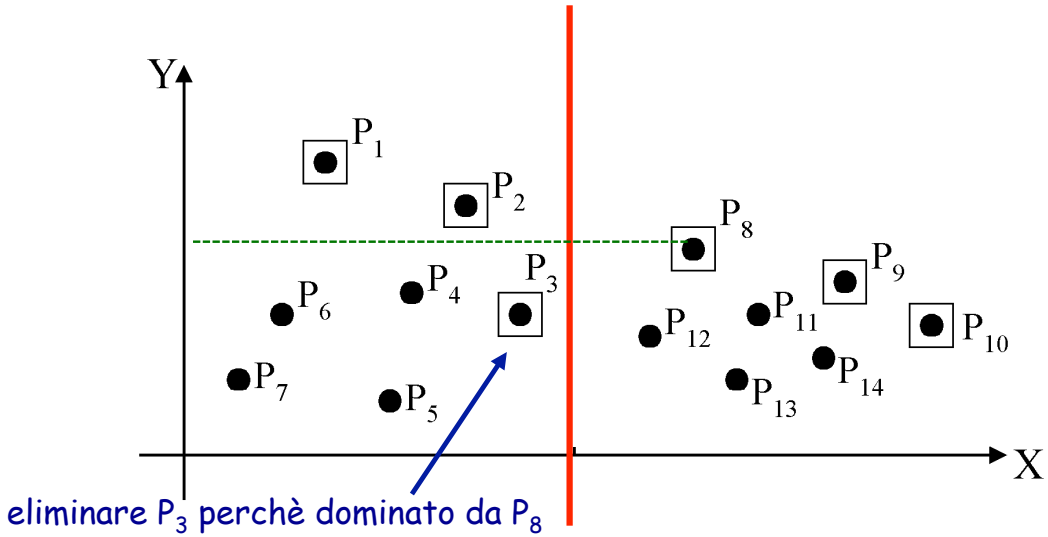


Esercizio: Massimi in 2-D

Input: insieme di punti in un piano
Calcolare: punti di massimo

Divide-and-conquer.

- Divide: dividere punti in due parti della stessa grandezza mediante una linea verticale.
- Conquer: risolvere ricorsivamente il problema in ognuna delle 2 parti.
- Combine: eliminare i punti di massimo della parte sinistra la cui ordinata è minore della massima ordinata di un punto di massimo nella parte destra. Restituire i punti di massimo delle due parti.

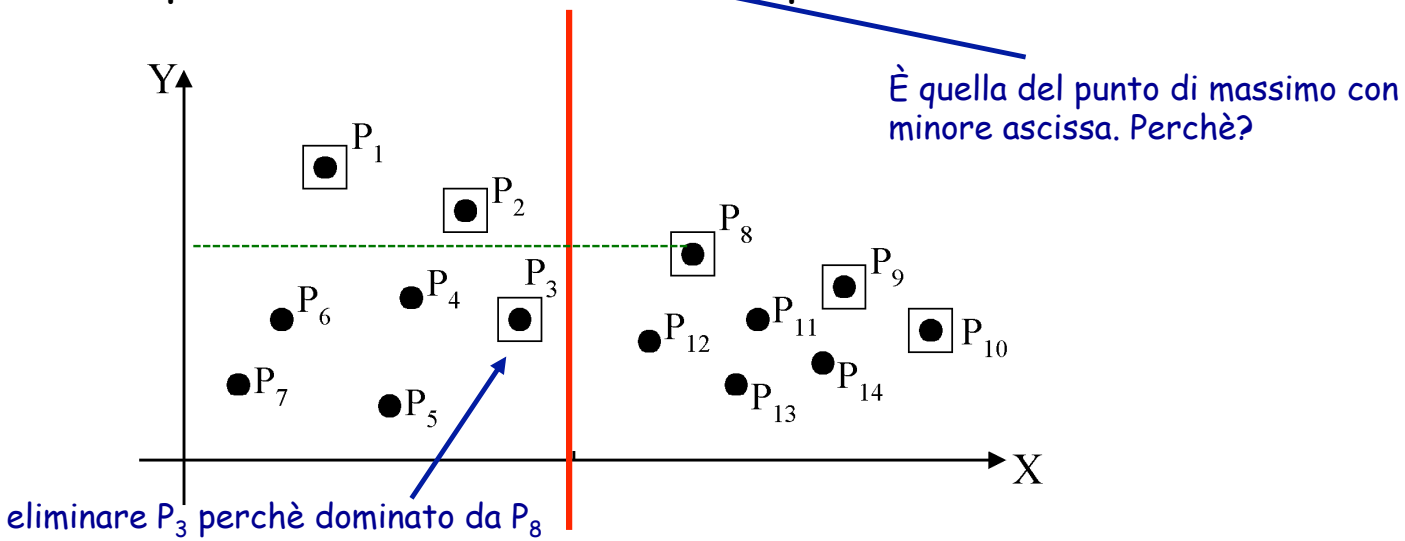


Esercizio: Massimi in 2-D

Input: insieme di punti in un piano
Calcolare: punti di massimo

Divide-and-conquer.

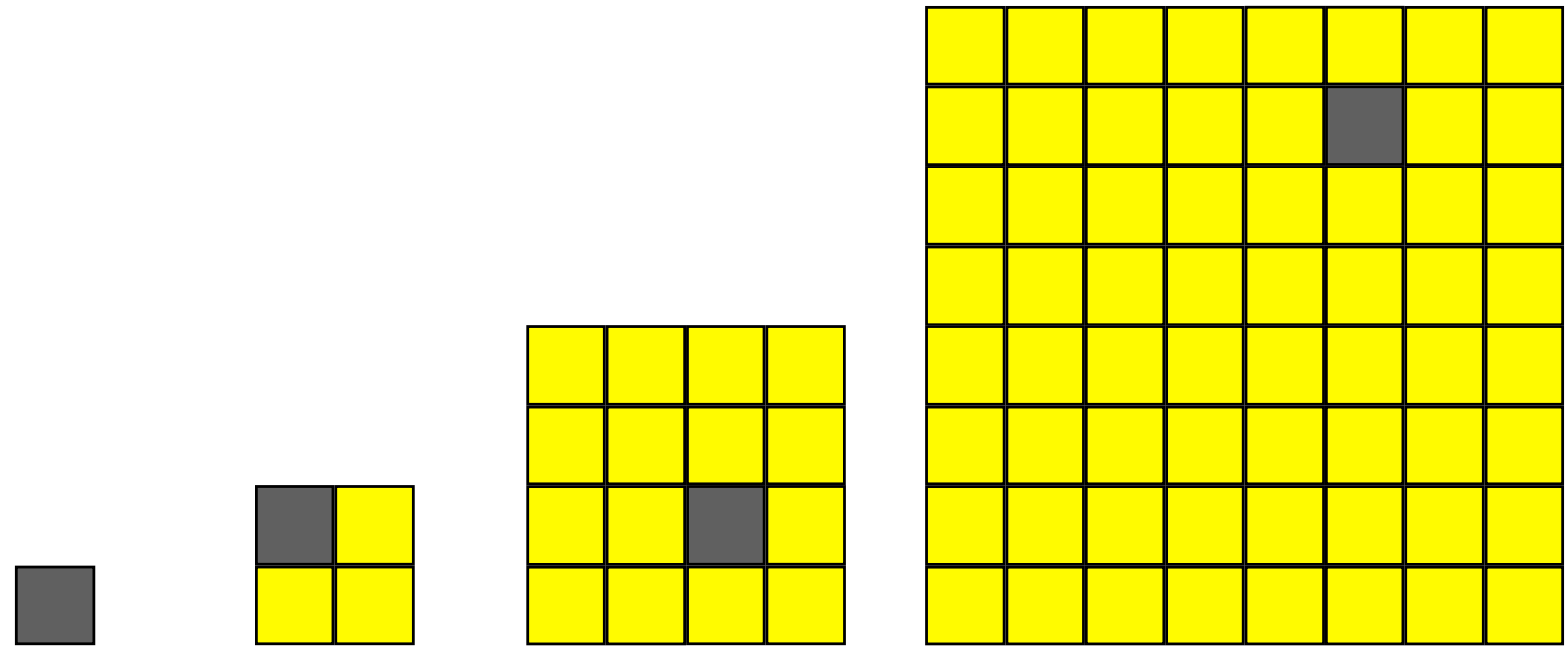
- Divide: dividere punti in due parti della stessa grandezza mediante una linea verticale.
- Conquer: risolvere ricorsivamente il problema in ogni parte.
- Combine: eliminare i punti di massimo della parte sinistra la cui ordinata è minore della massima ordinata di un punto di massimo nella parte destra. Restituire i punti di massimo delle due parti.



Esercizio: Defective Chessboards

Input: scacchiera n×n e casella non disponibile (assumere n potenza di 2)
Calcolare: ricoprimento con triomini

Una Defective Chessboard è una scacchiera con una casella non disponibile



Esercizio: Defective Chessboards

Input: scacchiera n×n e casella non disponibile
Calcolare: ricoprimento con triomini

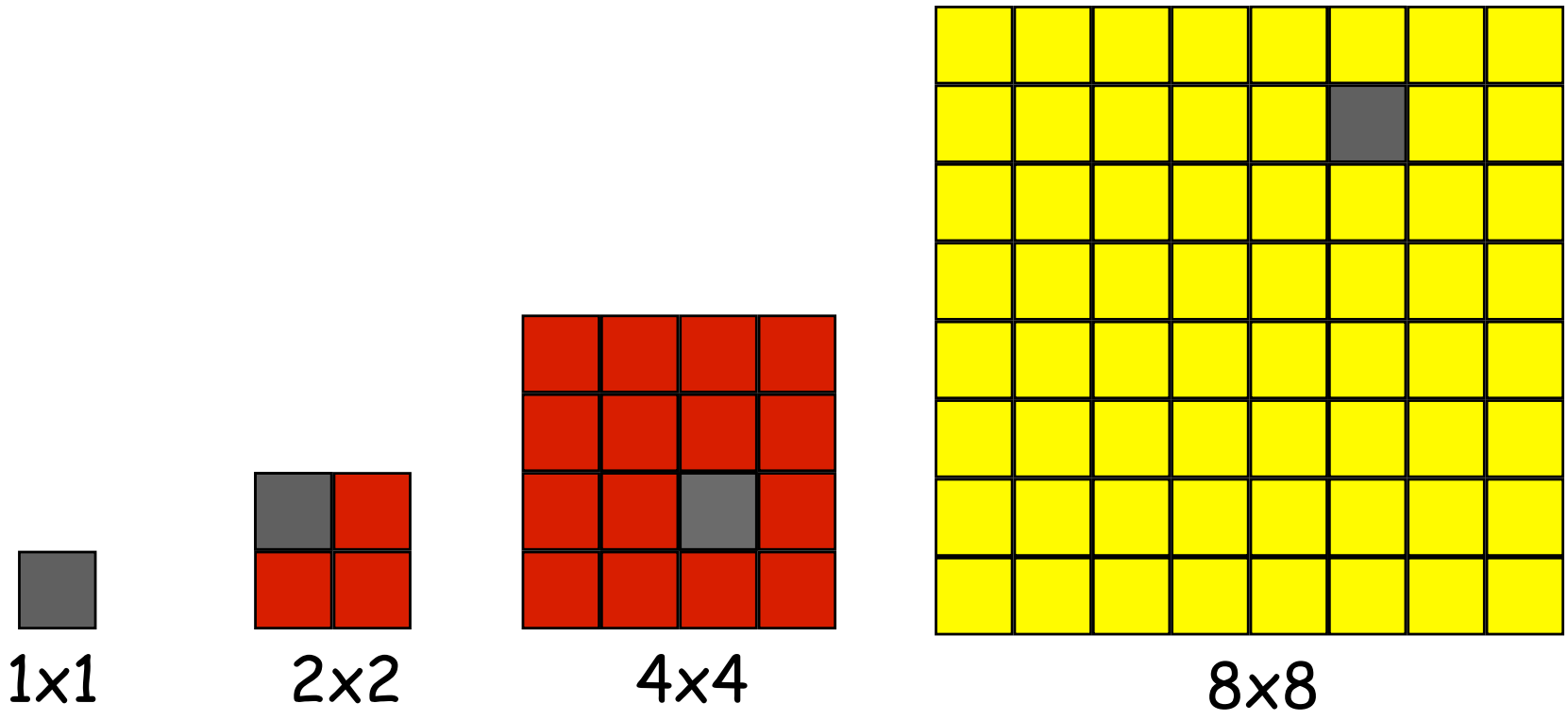
Un triomino è un oggetto a forma di L che copre 3 caselle di una scacchiera.

Ha 4 orientazioni:

Esercizio: Defective Chessboards

Input: scacchiera n×n e casella non disponibile

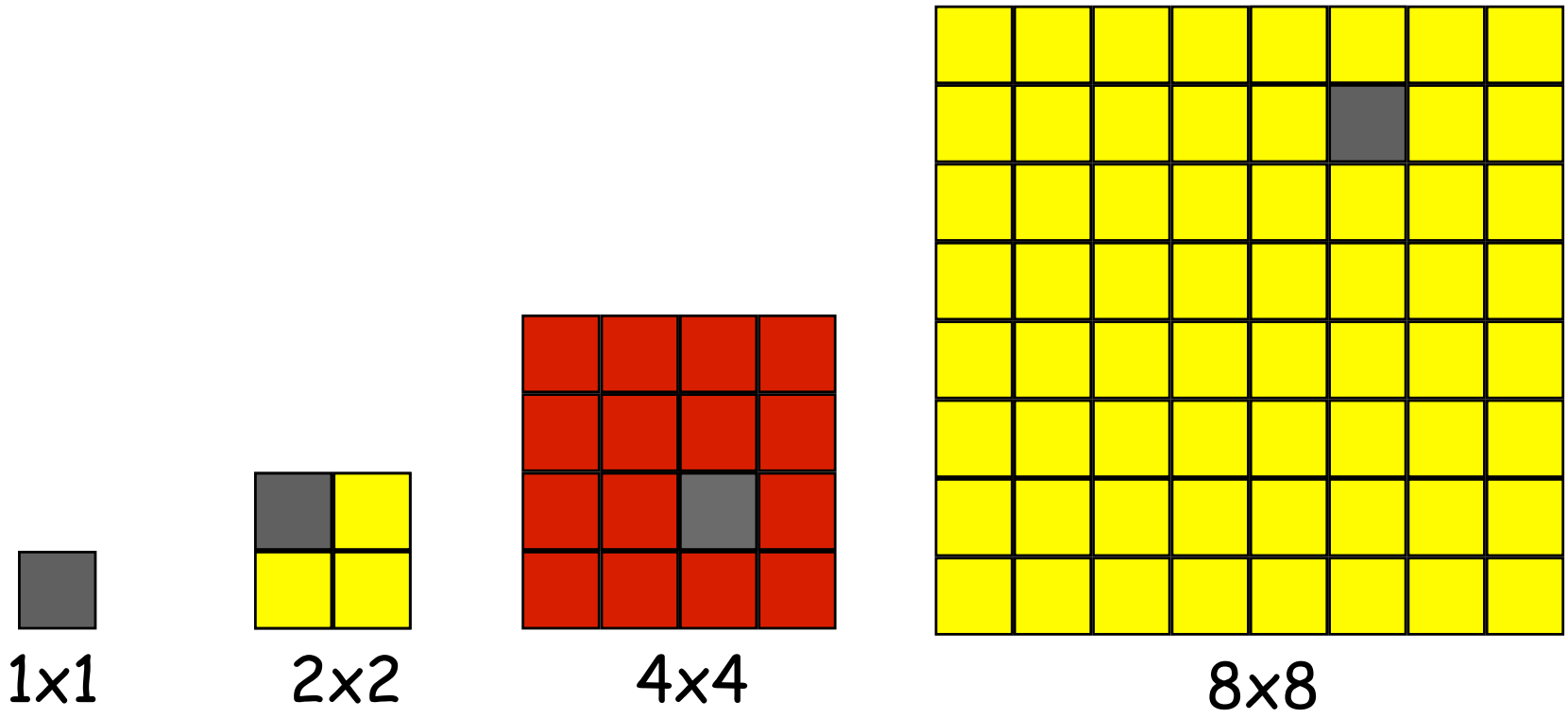
Calcolare: ricoprimento con triomini, cioè posizionamento di $(n^2-1)/3$ triomini sulla scacchiera in modo da ricoprire tutte le caselle disponibili.



Esercizio: Defective Chessboards

Input: scacchiera n×n e casella non disponibile

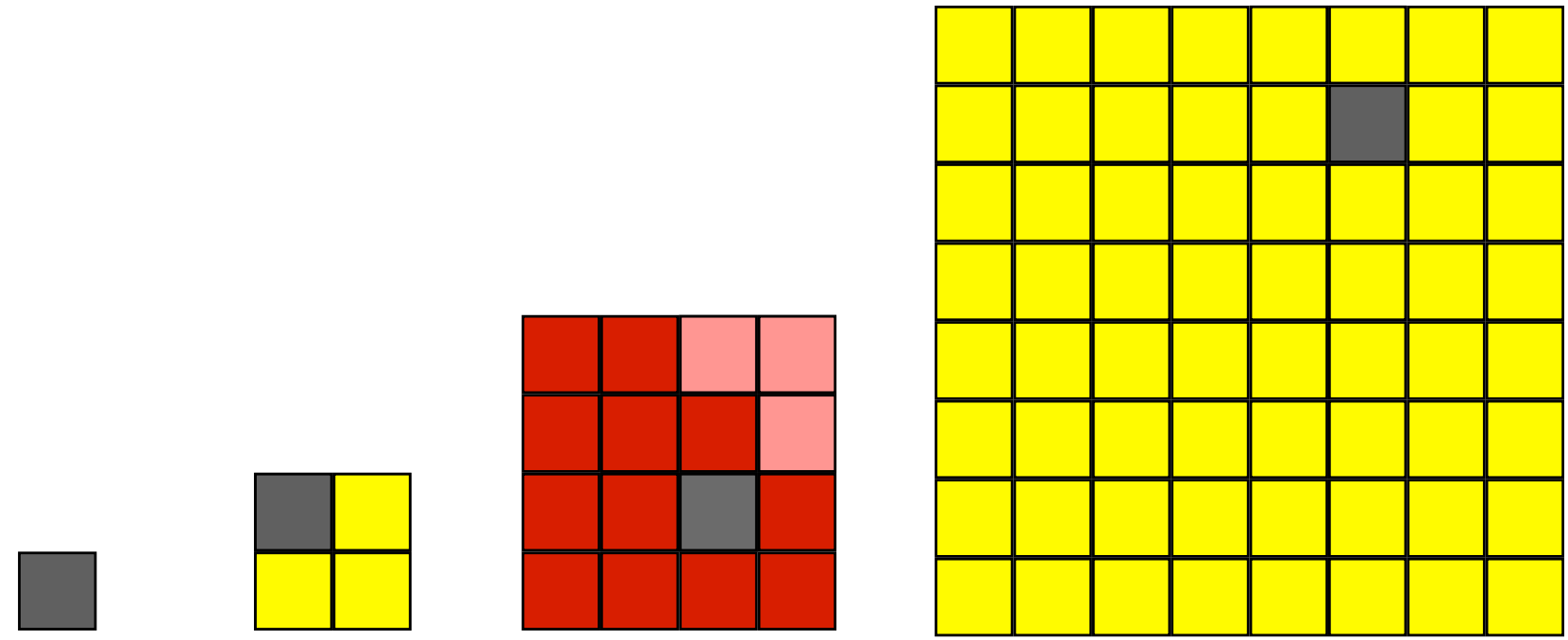
Calcolare: ricoprimento con triomini, cioè posizionamento di $(n^2-1)/3$ triomini sulla scacchiera in modo da ricoprire tutte le caselle disponibili.



Esercizio: Defective Chessboards

Input: scacchiera n×n e casella non disponibile

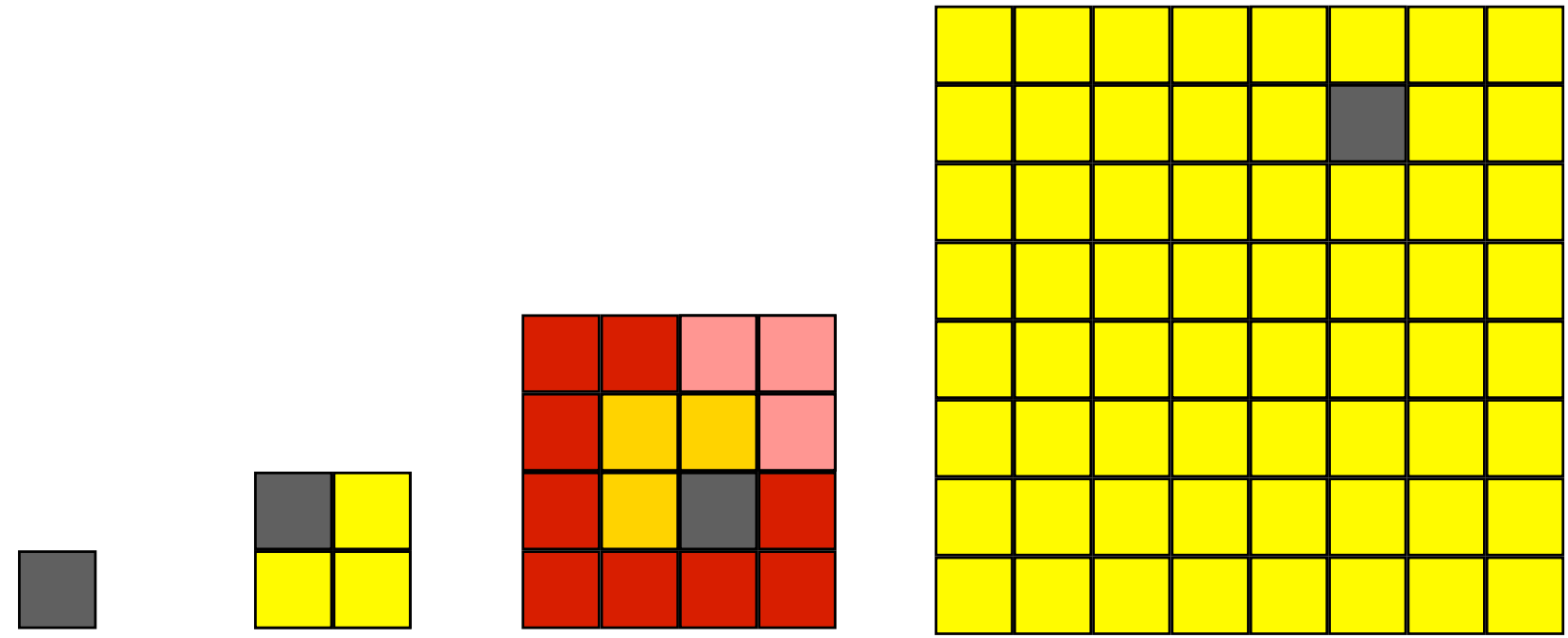
Calcolare: ricoprimento con triomini, cioè posizionamento di $(n^2-1)/3$ triomini sulla scacchiera in modo da ricoprire tutte le caselle disponibili.



Esercizio: Defective Chessboards

Input: scacchiera n×n e casella non disponibile

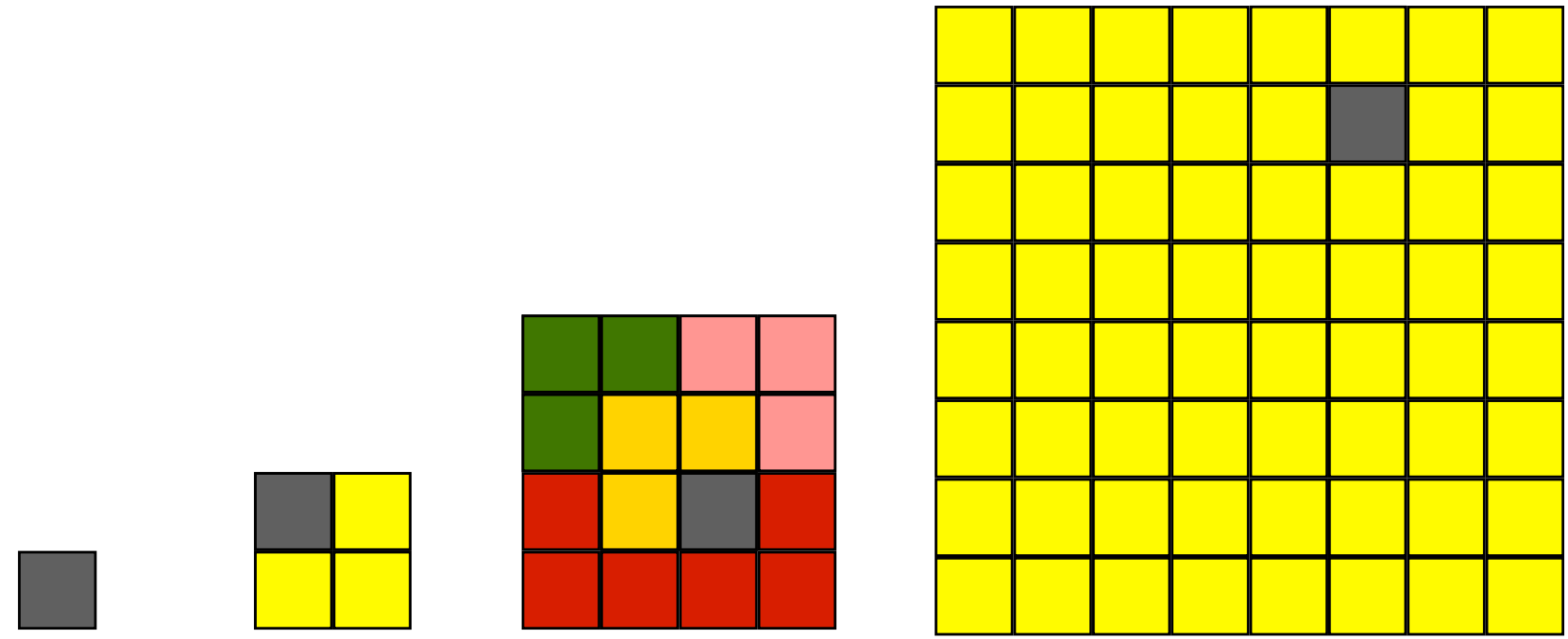
Calcolare: ricoprimento con triomini, cioè posizionamento di $(n^2-1)/3$ triomini sulla scacchiera in modo da ricoprire tutte le caselle disponibili.



Esercizio: Defective Chessboards

Input: scacchiera n×n e casella non disponibile

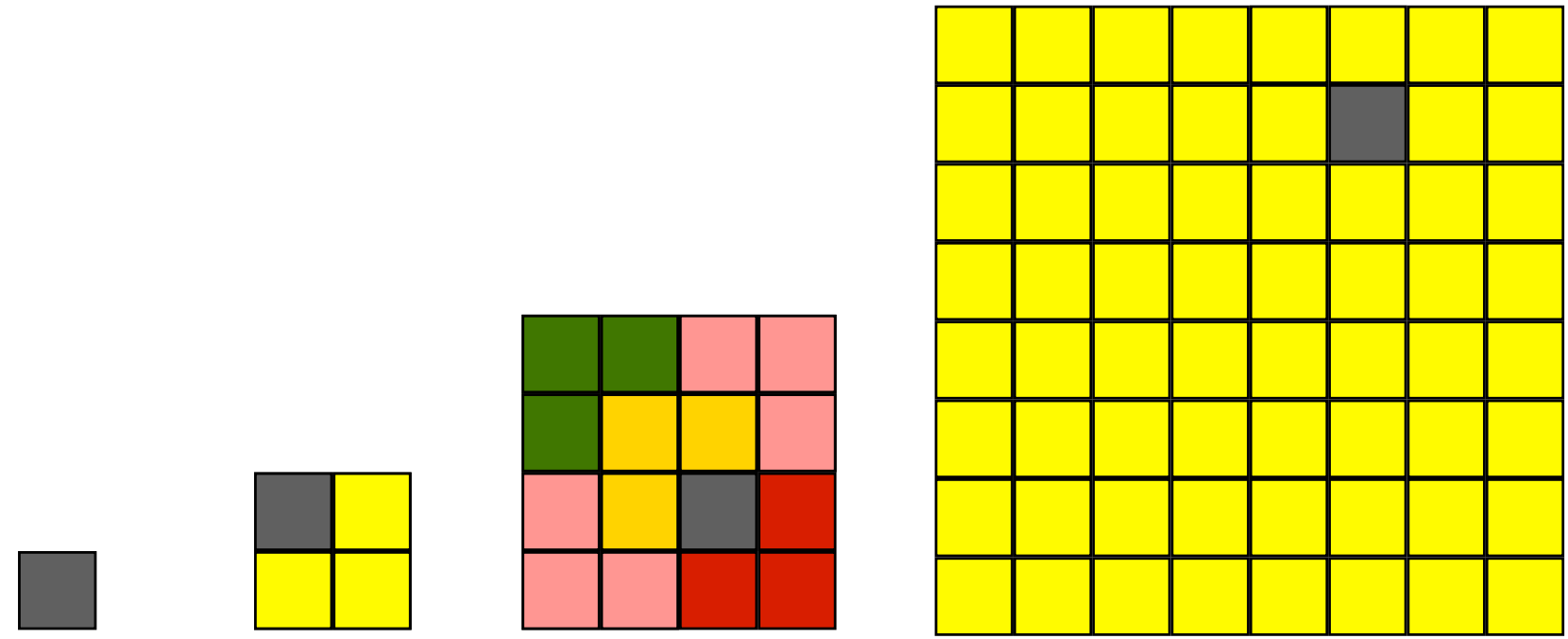
Calcolare: ricoprimento con triomini, cioè posizionamento di $(n^2-1)/3$ triomini sulla scacchiera in modo da ricoprire tutte le caselle disponibili.



Esercizio: Defective Chessboards

Input: scacchiera n×n e casella non disponibile

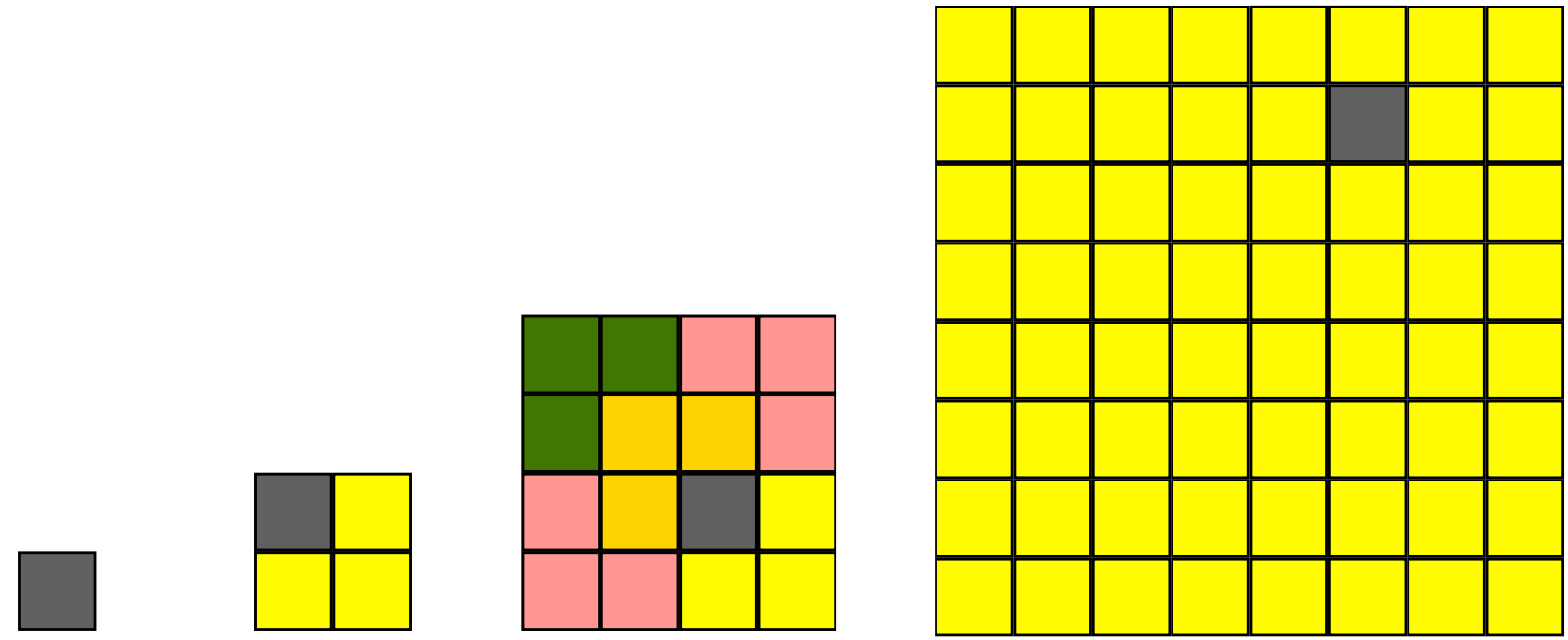
Calcolare: ricoprimento con triomini, cioè posizionamento di $(n^2-1)/3$ triomini sulla scacchiera in modo da ricoprire tutte le caselle disponibili.



Esercizio: Defective Chessboards

Input: scacchiera n×n e casella non disponibile

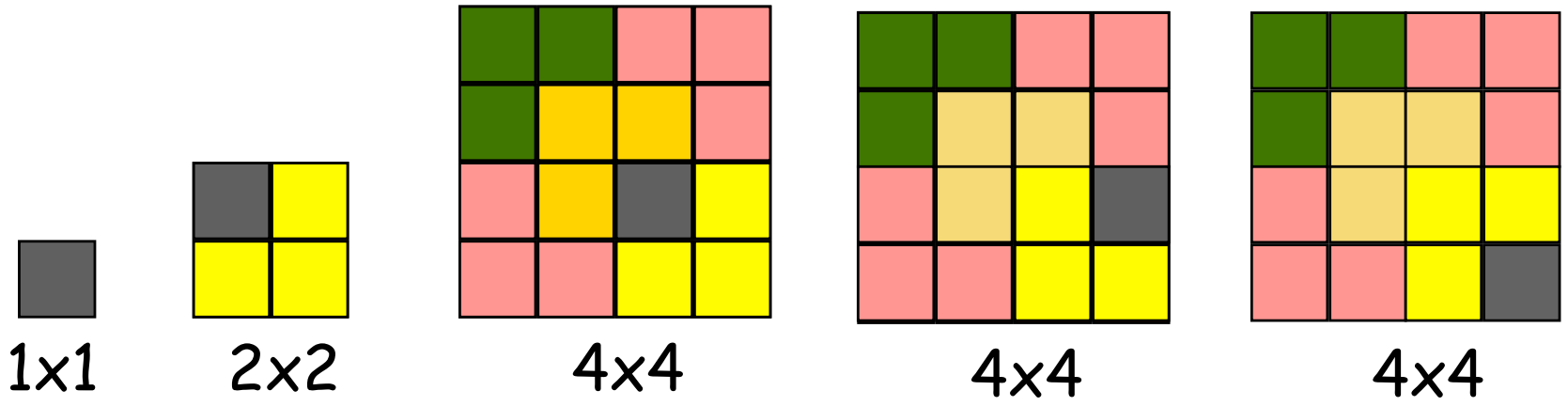
Calcolare: ricoprimento con triomini, cioè posizionamento di $(n^2-1)/3$ triomini sulla scacchiera in modo da ricoprire tutte le caselle disponibili.



Esercizio: Defective Chessboards

Input: scacchiera n×n e casella non disponibile

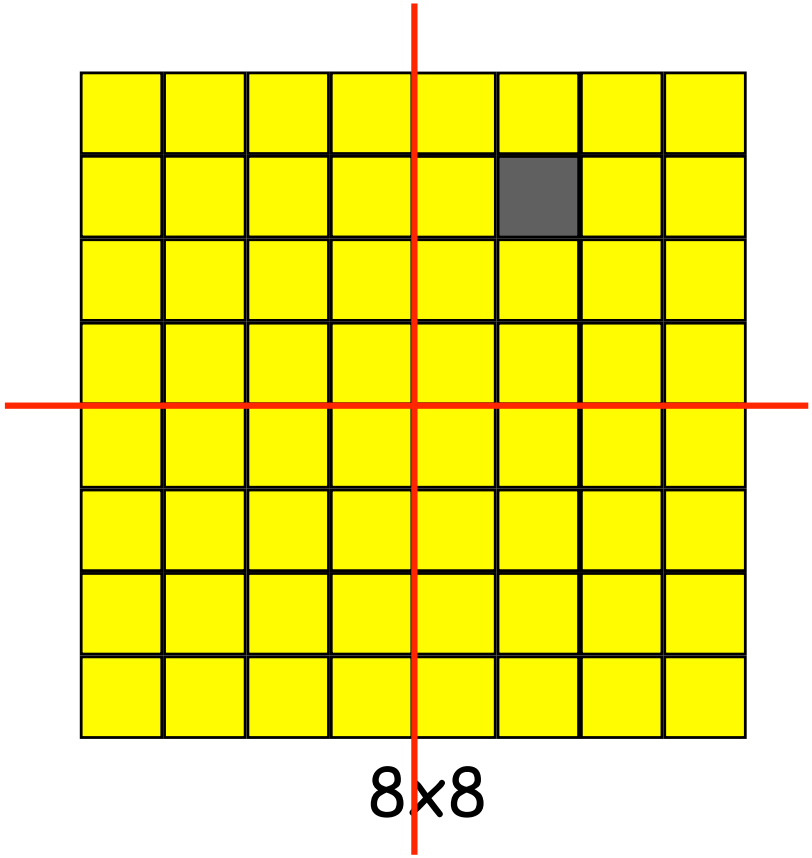
Calcolare: ricoprimento con triomini, cioè posizionamento di $(n^2-1)/3$ triomini sulla scacchiera in modo da ricoprire tutte le caselle disponibili.



Esercizio: Defective Chessboards

Input: scacchiera n×n e casella non disponibile

Calcolare: ricoprimento con triomini, cioè posizionamento di $(n^2-1)/3$ triomini sulla scacchiera in modo da ricoprire tutte le caselle disponibili.



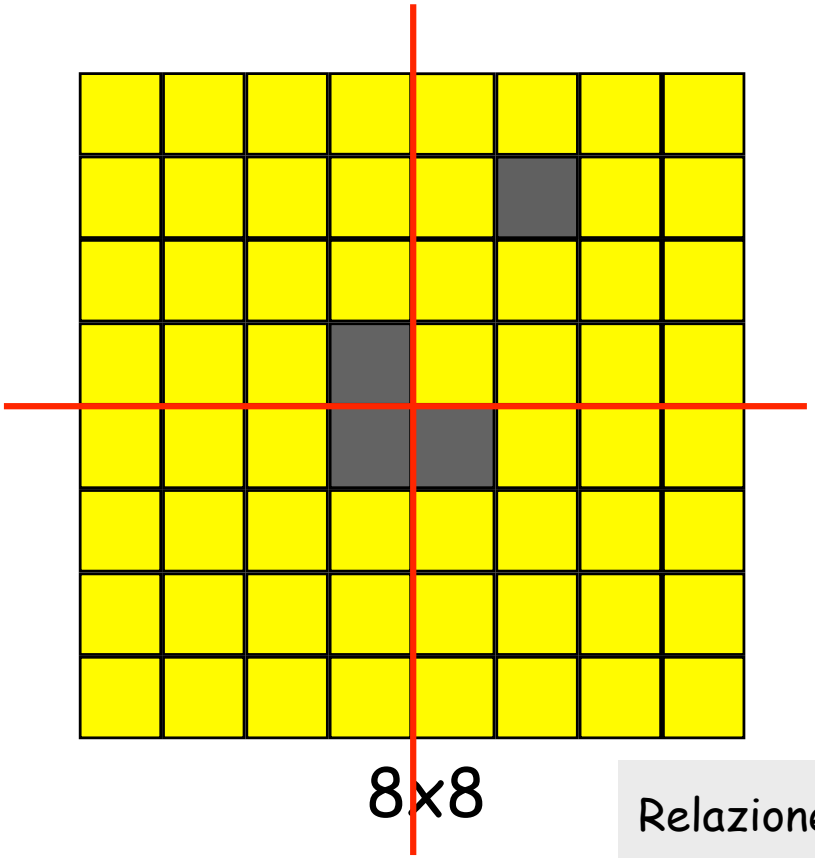
Divide-and-conquer.

- Divide: dividere scacchiera in quattro scacchiere.

Esercizio: Defective Chessboards

Input: scacchiera n×n e casella non disponibile

Calcolare: ricoprimento con triomini, cioè posizionamento di $(n^2-1)/3$ triomini sulla scacchiera in modo da ricoprire tutte le caselle disponibili.



Divide-and-conquer.

- Divide: dividere scacchiera in quattro scacchiere.
- Conquer: rendere defective le altre tre scacchiere e risolvere ricorsivamente il problema per ogni scacchiera.
- Combine: Restituire i ricoprimenti delle quattro parti con in aggiunta il triomino centrale.

Relazione ricorrenza running time $T(n) = 4\ T(n/2) + \Theta(1)$

Soluzione $T(n) = \Theta(n^2)$

Esercizio: Defective Chessboards

Soluzione relazione ricorrenza con metodo di sostituzione

$T(n) = 4\,T(n/2) + c$ con $n=2^k$

$= 4\,(4\,T(n/4) + c) + c$

$= 4^2\,T(n/4) + 4c + c$

$= 4^2\,(4\,T(n/8) + c) + 4c + c$

$= 4^3\,T(n/8) + 4^2c + 4c + c$

$= 4^3\,(4\,T(n/16) + c) + 4^2c + 4c + c$

$= 4^4\,T(n/16) + 4^3c + 4^2c + 4c + c$

$= \ldots$

$= 4^k\,T(n/2^k) + 4^{k-1}c + 4^{k-2}c + \ldots 4^3c + 4^2c + 4c + c$

$= \Theta(4^k)$

$= \Theta(n^2)$

da $\sum_{i=0}^{k-1} \vartheta^i = \frac{\vartheta^k - 1}{\vartheta - 1}$

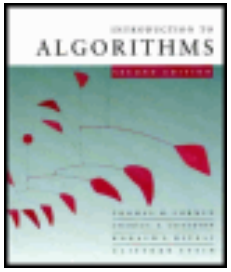
segue che $\sum_{i=0}^{k-1} 4^i = \frac{4^k - 1}{3}$

Riepilogo Capitolo 5, Divide and Conquer

5.1 Mergesort
5.2 Recurrence Relations
5.3 Counting Inversions
5.4 Closest Pair of Points
5.5 Integer Multiplication
5.6 Fast Fourier Transform (no)
Matrix Multiplication
Esercizi

Quicksort

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein
Introduction to Algorithms, Second Edition
The MIT Press © 2001 (Cap. 7 Quicksort)



In italiano:
Introduzione agli algoritmi e strutture dati 2/ed
McGraw-Hill, Maggio 2005