

Chapter 5

Divide and Conquer

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*

2

Divide and Conquer

Algoritmi che vedremo:

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

3

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.

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.
- ...

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.

5

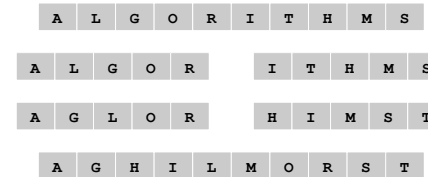
Mergesort

Mergesort.

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

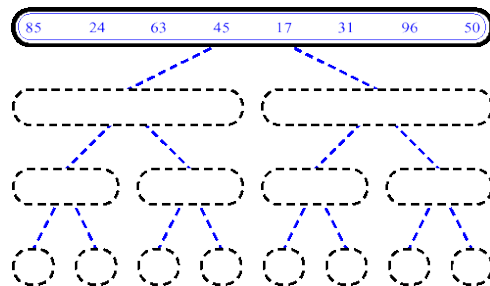


Jon von Neumann (1945)



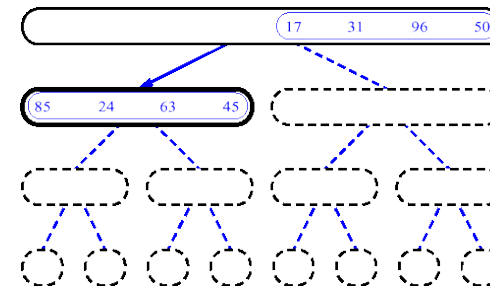
6

MergeSort (esempio) - 1



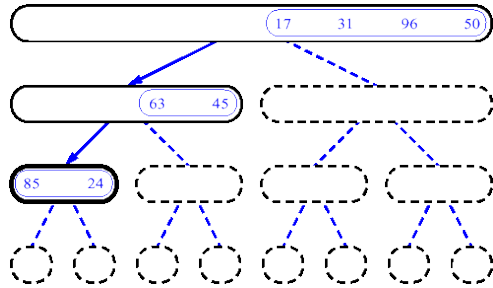
7

MergeSort (esempio) - 2



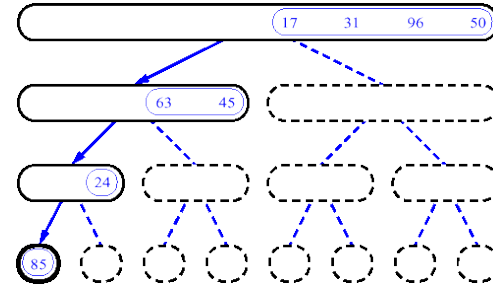
8

MergeSort (esempio) - 3



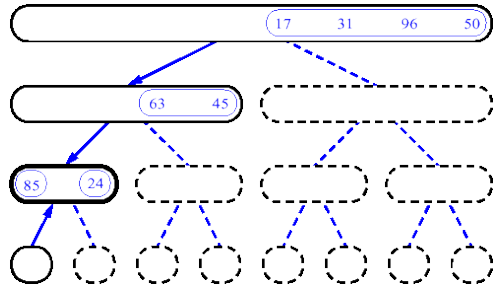
9

MergeSort (esempio) - 4



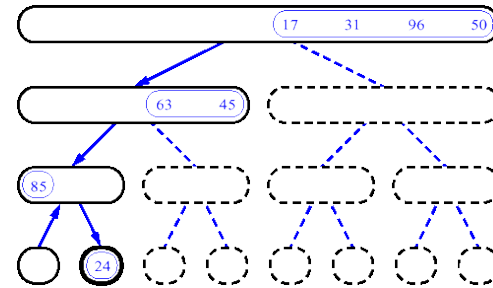
10

MergeSort (esempio) - 5



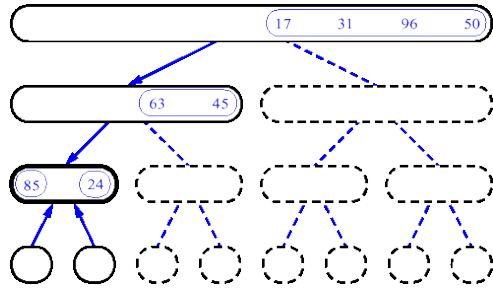
11

MergeSort (esempio) - 6



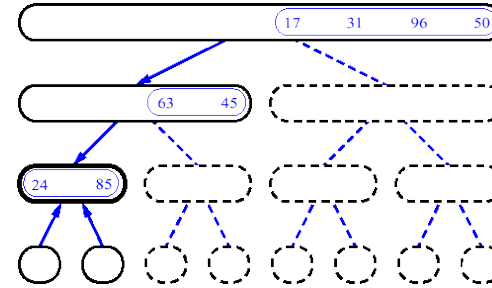
12

MergeSort (esempio) - 7



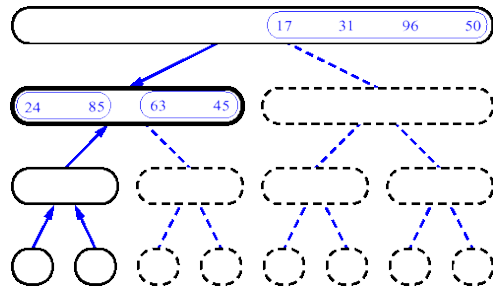
13

MergeSort (esempio) - 8



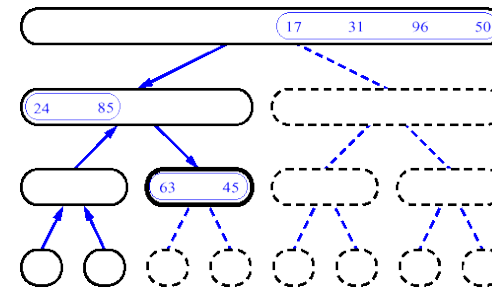
14

MergeSort (esempio) - 9



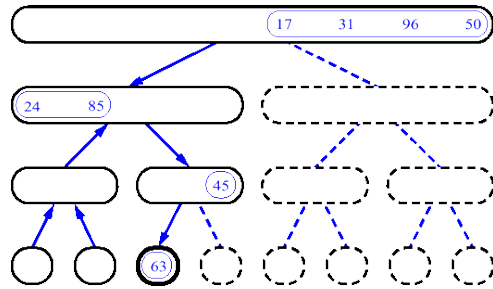
15

MergeSort (esempio) - 10



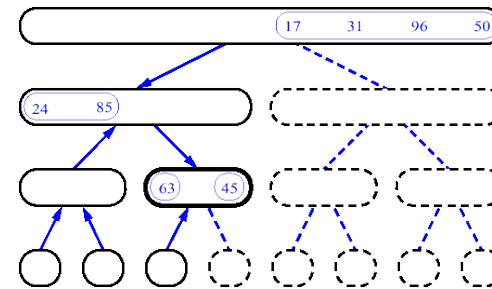
16

MergeSort (esempio) - 11



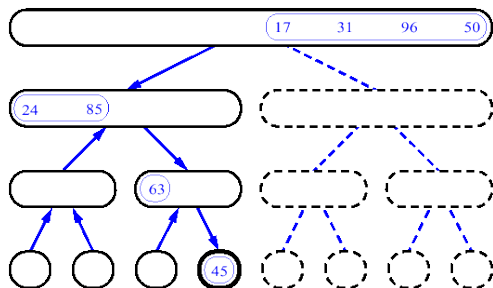
17

MergeSort (esempio) - 12



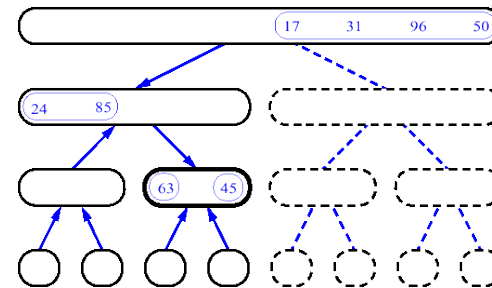
18

MergeSort (esempio) - 13



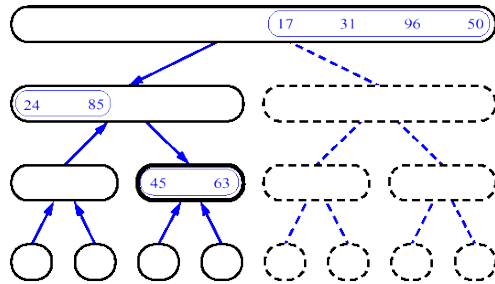
19

MergeSort (esempio) - 14



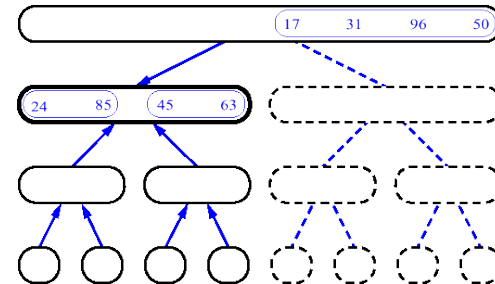
20

MergeSort (esempio) - 15



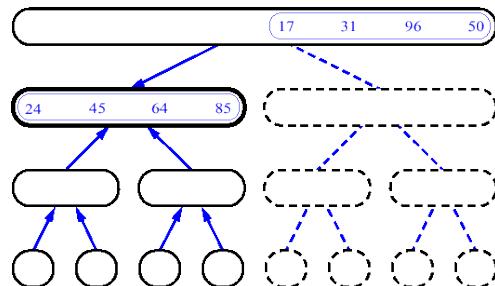
21

MergeSort (esempio) - 16



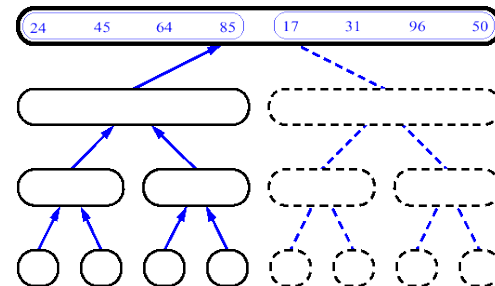
22

MergeSort (esempio) - 17



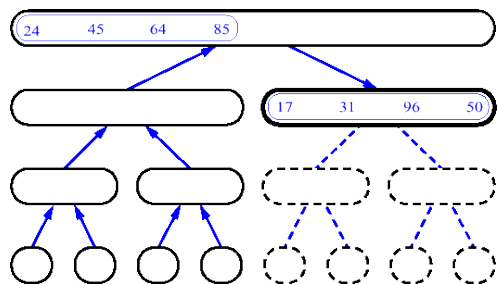
23

MergeSort (esempio) - 18



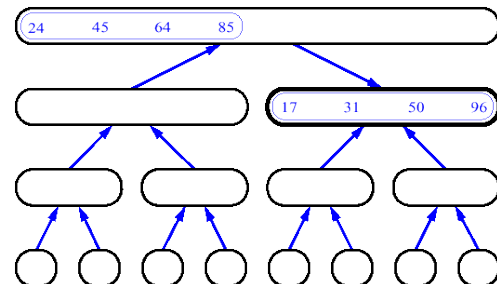
24

MergeSort (esempio) - 19



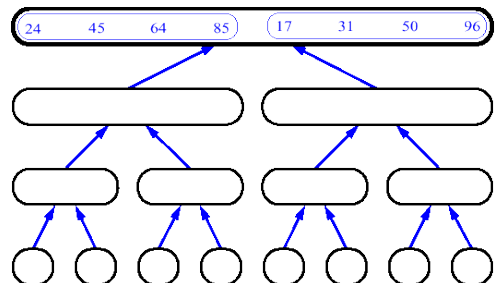
25

MergeSort (esempio) - 20



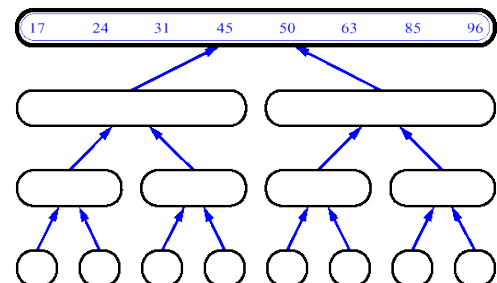
26

MergeSort (esempio) - 21



27

MergeSort (esempio) - 22



28

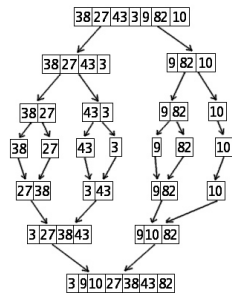
Mergesort

Mergesort.

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



Jon von Neumann (1945)



29

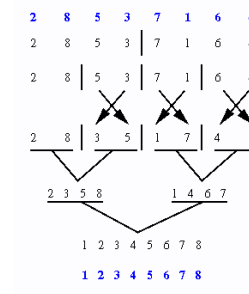
Mergesort

Mergesort.

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



Jon von Neumann (1945)



30

Merge

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

Come effettuare il merge efficientemente?

- Uso di un array temporaneo.

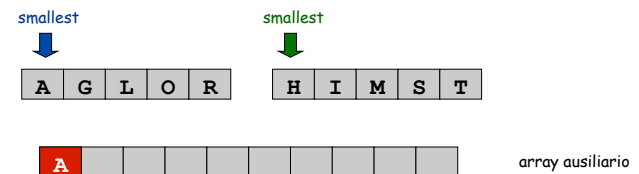


31

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.



32

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.

array ausiliario

33

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.

array ausiliario

34

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.

array ausiliario

35

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.

array ausiliario

36

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.

array ausiliario

37

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.

array ausiliario

38

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.

array ausiliario

39

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.

array ausiliario

40

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.

array ausiliario

41

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.

array ausiliario

42

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.

43

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.

dividi $O(1)$

ordina $2T(n/2)$

merge $O(n)$

44

Una utile relazione di ricorrenza

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

Ricorrenza del Mergesort.

$$T(n) \leq \begin{cases} 0 & \text{se } n=1 \\ T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + cn & \text{altrimenti} \end{cases}$$

risolvi parte sinistra
risolvi parte destra
merge

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

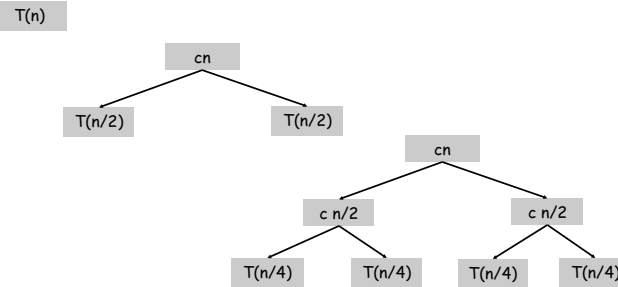
Varie prove. Descriviamo vari modi per risolvere questa ricorrenza. Inizialmente assumiamo che n è una potenza di 2 e sostituiamo \leq con $=$.

45

Prova con Albero di Ricorrenza

$$T(n) \leq \begin{cases} 0 & \text{se } n=1 \\ T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + cn & \text{altrimenti} \end{cases}$$

risolvi parte sinistra
risolvi parte destra
merge

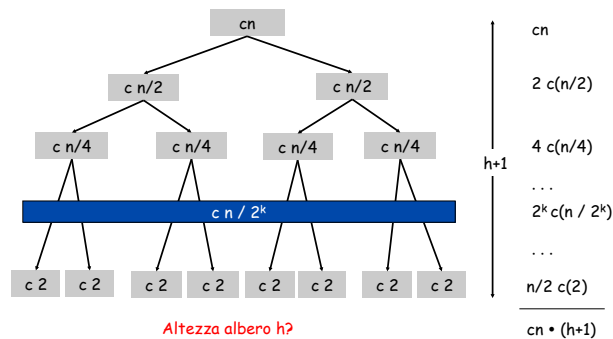


46

Prova con Albero di Ricorsonza

$$T(n) \leq \begin{cases} 0 & \text{se } n=1 \\ T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + cn & \text{altrimenti} \end{cases}$$

risolvi parte sinistra
risolvi parte destra
merge

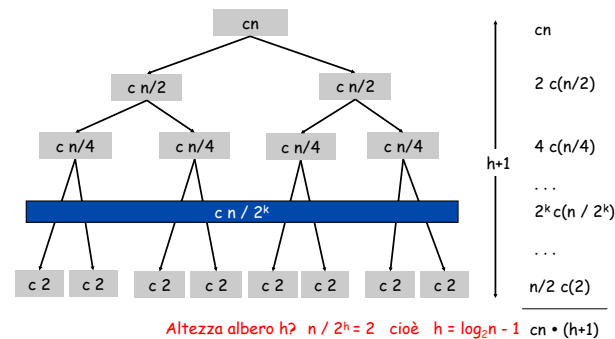


47

Prova con Albero di Ricorsonza

$$T(n) \leq \begin{cases} 0 & \text{se } n=1 \\ T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + cn & \text{altrimenti} \end{cases}$$

risolvi parte sinistra
risolvi parte destra
merge



48

Prova con sequenza telescopica

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

$$T(n) \leq \begin{cases} 0 & \text{se } n=1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + cn & \text{altrimenti} \end{cases}$$

risolvi parte sinistra risolvi parte destra merge

Prova. Per $n > 1$:

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

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

49

Prova per induzione

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

$$T(n) \leq \begin{cases} 0 & \text{se } n=1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + cn & \text{altrimenti} \end{cases}$$

risolvi parte sinistra risolvi parte destra merge

Prova. (per induzione su n)

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

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

50

Analisi della ricorrenza del Mergesort

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

$$T(n) \leq \begin{cases} 0 & \text{se } n=1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + cn & \text{altrimenti} \end{cases}$$

risolvi parte sinistra risolvi parte destra merge

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 $1, 2, \dots, n-1$.

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

$n_2 = \lceil n/2 \rceil$
 $\leq \lceil 2^{\lceil \log n \rceil} / 2 \rceil$
 $= 2^{\lceil \log n \rceil} / 2$
 $\Rightarrow \log n_2 \leq \lceil \log n \rceil - 1$

51

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, \dots, 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.

53

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).

54

Conteggio Inversioni

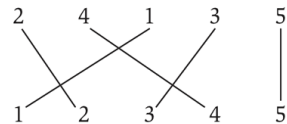


Figure 5.4 Counting the number of inversions in the sequence 2, 4, 1, 3, 5. Each crossing pair of line segments corresponds to one pair that is in the opposite order in the input list and the ascending list—in other words, an inversion.

Inversioni
2-1, 4-1, 4-3

55

Conteggio Inversioni: Divide-and-Conquer

Divide-and-conquer.

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

56

Conteggio Inversioni: Divide-and-Conquer

Divide-and-conquer.

- **Divide:** dividi lista in due parti.

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

Divide: $O(1)$.

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

57

Conteggio Inversioni: Divide-and-Conquer

Divide-and-conquer.

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

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

Divide: $O(1)$.

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

Conquer: $2 T(n / 2)$

5 inversioni blu-blu
5-4, 5-2, 4-2, 8-2, 10-2

8 inversioni verde-verde
6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

58

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 delle tre quantità.

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

Divide: $O(1)$.

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

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.

59

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 |
|---|----|----|----|----|----|

60

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.

$i = 6$

↓

| | | | | | | | | | | | |
|---|---|----|----|----|----|---|----|----|----|----|----|
| 3 | 7 | 10 | 14 | 18 | 19 | 2 | 11 | 16 | 17 | 23 | 25 |
|---|---|----|----|----|----|---|----|----|----|----|----|

due metà ordinate

↓

| | | | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|--|--|

array ausiliario

Totale:

61

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.

$i = 6$

↓

| | | | | | | | | | | | |
|---|---|----|----|----|----|---|----|----|----|----|----|
| 3 | 7 | 10 | 14 | 18 | 19 | 2 | 11 | 16 | 17 | 23 | 25 |
|---|---|----|----|----|----|---|----|----|----|----|----|

due metà ordinate

↓

| | | | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|--|--|

array ausiliario

Totale: 6

62

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.

$i = 6$

↓

| | | | | | | | | | | | |
|---|---|----|----|----|----|---|----|----|----|----|----|
| 3 | 7 | 10 | 14 | 18 | 19 | 2 | 11 | 16 | 17 | 23 | 25 |
|---|---|----|----|----|----|---|----|----|----|----|----|

due metà ordinate

↓

| | | | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|--|--|

array ausiliario

Totale: 6

63

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.

$i = 6$

↓

| | | | | | | | | | | | |
|---|---|----|----|----|----|---|----|----|----|----|----|
| 3 | 7 | 10 | 14 | 18 | 19 | 2 | 11 | 16 | 17 | 23 | 25 |
|---|---|----|----|----|----|---|----|----|----|----|----|

due metà ordinate

↓

| | | | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|--|--|

array ausiliario

Totale: 6

64

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.

$i = 5$

↓

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|---|---|----|----|----|----|----|-------------------|
| 3 | 7 | 10 | 14 | 18 | 19 | ↓ | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate |
| | | | | | | 6 | | | | | | | |

| | | | | | | | | | | | | | |
|---|---|--|--|--|--|--|--|--|--|--|--|--|------------------|
| 2 | 3 | | | | | | | | | | | | array ausiliario |
|---|---|--|--|--|--|--|--|--|--|--|--|--|------------------|

Totale: 6

65

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.

$i = 5$

↓

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|---|---|----|----|----|----|----|-------------------|
| 3 | 7 | 10 | 14 | 18 | 19 | ↓ | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate |
| | | | | | | 6 | | | | | | | |

| | | | | | | | | | | | | | |
|---|---|---|--|--|--|--|--|--|--|--|--|--|------------------|
| 2 | 3 | 7 | | | | | | | | | | | array ausiliario |
|---|---|---|--|--|--|--|--|--|--|--|--|--|------------------|

Totale: 6

66

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.

$i = 4$

↓

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|---|---|----|----|----|----|----|-------------------|
| 3 | 7 | 10 | 14 | 18 | 19 | ↓ | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate |
| | | | | | | 6 | | | | | | | |

| | | | | | | | | | | | | | |
|---|---|---|--|--|--|--|--|--|--|--|--|--|------------------|
| 2 | 3 | 7 | | | | | | | | | | | array ausiliario |
|---|---|---|--|--|--|--|--|--|--|--|--|--|------------------|

Totale: 6

67

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.

$i = 4$

↓

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|---|---|----|----|----|----|----|-------------------|
| 3 | 7 | 10 | 14 | 18 | 19 | ↓ | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate |
| | | | | | | 6 | | | | | | | |

| | | | | | | | | | | | | | |
|---|---|---|----|--|--|--|--|--|--|--|--|--|------------------|
| 2 | 3 | 7 | 10 | | | | | | | | | | array ausiliario |
|---|---|---|----|--|--|--|--|--|--|--|--|--|------------------|

Totale: 6

68

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.

$i = 3$

↓

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|---|---|----|----|----|----|----|-------------------|
| 3 | 7 | 10 | 14 | 18 | 19 | ↓ | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate |
| | | | | | | 6 | | | | | | | |

| | | | | | | | | | | | | | |
|---|---|---|----|--|--|--|--|--|--|--|--|--|------------------|
| 2 | 3 | 7 | 10 | | | | | | | | | | array ausiliario |
|---|---|---|----|--|--|--|--|--|--|--|--|--|------------------|

Totale: 6

69

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.

$i = 3$

↓

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|---|---|----|----|----|----|----|-------------------|
| 3 | 7 | 10 | 14 | 18 | 19 | ↓ | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate |
| | | | | | | 6 | 3 | | | | | | |

| | | | | | | | | | | | | | |
|---|---|---|----|----|--|--|--|--|--|--|--|--|------------------|
| 2 | 3 | 7 | 10 | 11 | | | | | | | | | array ausiliario |
|---|---|---|----|----|--|--|--|--|--|--|--|--|------------------|

Totale: 6 + 3

70

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.

$i = 3$

↓

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|---|---|----|----|----|----|----|-------------------|
| 3 | 7 | 10 | 14 | 18 | 19 | ↓ | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate |
| | | | | | | 6 | 3 | | | | | | |

| | | | | | | | | | | | | | |
|---|---|---|----|----|--|--|--|--|--|--|--|--|------------------|
| 2 | 3 | 7 | 10 | 11 | | | | | | | | | array ausiliario |
|---|---|---|----|----|--|--|--|--|--|--|--|--|------------------|

Totale: 6 + 3

71

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.

$i = 3$

↓

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|---|---|----|----|----|----|----|-------------------|
| 3 | 7 | 10 | 14 | 18 | 19 | ↓ | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate |
| | | | | | | 6 | 3 | | | | | | |

| | | | | | | | | | | | | | |
|---|---|---|----|----|----|--|--|--|--|--|--|--|------------------|
| 2 | 3 | 7 | 10 | 11 | 14 | | | | | | | | array ausiliario |
|---|---|---|----|----|----|--|--|--|--|--|--|--|------------------|

Totale: 6 + 3

72

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.

$i = 2$

↓

| | | | | | | | | | | | | | | |
|---|---|----|----|----|----|-----|---|----|----|----|----|----|-------------------|--|
| 3 | 7 | 10 | 14 | 18 | 19 | | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate | |
| | | | | | | 6 | | | | | | | 3 | |

| | | | | | | | | | | | | | |
|---|---|---|----|----|----|--|--|--|--|--|--|--|------------------|
| 2 | 3 | 7 | 10 | 11 | 14 | | | | | | | | array ausiliario |
|---|---|---|----|----|----|--|--|--|--|--|--|--|------------------|

Totale: $6 + 3$

73

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.

$i = 2$

↓

| | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|-----|---|----|----|----|----|----|-------------------|-----|--|
| 3 | 7 | 10 | 14 | 18 | 19 | | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate | | |
| | | | | | | 6 | | | | | | | 3 | 2 | |

| | | | | | | | | | | | | | |
|---|---|---|----|----|----|----|--|--|--|--|--|--|------------------|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | | | | | | | array ausiliario |
|---|---|---|----|----|----|----|--|--|--|--|--|--|------------------|

Totale: $6 + 3 + 2$

74

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.

$i = 2$

↓

| | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|-----|---|----|----|----|----|----|-------------------|-----|--|
| 3 | 7 | 10 | 14 | 18 | 19 | | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate | | |
| | | | | | | 6 | | | | | | | 3 | 2 | |

| | | | | | | | | | | | | | |
|---|---|---|----|----|----|----|--|--|--|--|--|--|------------------|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | | | | | | | array ausiliario |
|---|---|---|----|----|----|----|--|--|--|--|--|--|------------------|

Totale: $6 + 3 + 2$

75

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.

$i = 2$

↓

| | | | | | | | | | | | | | | | | |
|---|---|----|----|----|----|-----|---|----|----|----|----|----|-------------------|-----|-----|--|
| 3 | 7 | 10 | 14 | 18 | 19 | | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate | | | |
| | | | | | | 6 | | | | | | | 3 | 2 | 2 | |

| | | | | | | | | | | | | | |
|---|---|---|----|----|----|----|----|--|--|--|--|--|------------------|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 17 | | | | | | array ausiliario |
|---|---|---|----|----|----|----|----|--|--|--|--|--|------------------|

Totale: $6 + 3 + 2 + 2$

76

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.

$i = 2$

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|---|---|----|----|----|----|----|-------------------|
| 3 | 7 | 10 | 14 | 18 | 19 | ↓ | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate |
| | | | | | | | 6 | 3 | 2 | 2 | | | |

| | | | | | | | | | | | | | |
|---|---|---|----|----|----|----|----|--|--|--|--|--|------------------|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 17 | | | | | | array ausiliario |
|---|---|---|----|----|----|----|----|--|--|--|--|--|------------------|

Totale: $6 + 3 + 2 + 2$

77

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.

$i = 2$

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|---|---|----|----|----|----|----|-------------------|
| 3 | 7 | 10 | 14 | 18 | 19 | ↓ | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate |
| | | | | | | | 6 | 3 | 2 | 2 | | | |

| | | | | | | | | | | | | | |
|---|---|---|----|----|----|----|----|----|--|--|--|--|------------------|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 17 | 18 | | | | | array ausiliario |
|---|---|---|----|----|----|----|----|----|--|--|--|--|------------------|

Totale: $6 + 3 + 2 + 2$

78

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.

$i = 1$

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|---|---|----|----|----|----|----|-------------------|
| 3 | 7 | 10 | 14 | 18 | 19 | ↓ | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate |
| | | | | | | | 6 | 3 | 2 | 2 | | | |

| | | | | | | | | | | | | | |
|---|---|---|----|----|----|----|----|----|--|--|--|--|------------------|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 17 | 18 | | | | | array ausiliario |
|---|---|---|----|----|----|----|----|----|--|--|--|--|------------------|

Totale: $6 + 3 + 2 + 2$

79

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.

$i = 1$

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|---|---|----|----|----|----|----|-------------------|
| 3 | 7 | 10 | 14 | 18 | 19 | ↓ | 2 | 11 | 16 | 17 | 23 | 25 | due metà ordinate |
| | | | | | | | 6 | 3 | 2 | 2 | | | |

| | | | | | | | | | | | | | |
|---|---|---|----|----|----|----|----|----|----|--|--|--|------------------|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 17 | 18 | 19 | | | | array ausiliario |
|---|---|---|----|----|----|----|----|----|----|--|--|--|------------------|

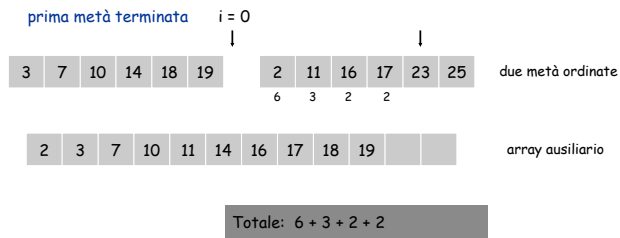
Totale: $6 + 3 + 2 + 2$

80

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.

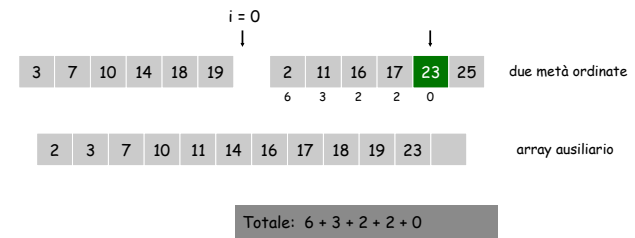


81

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.

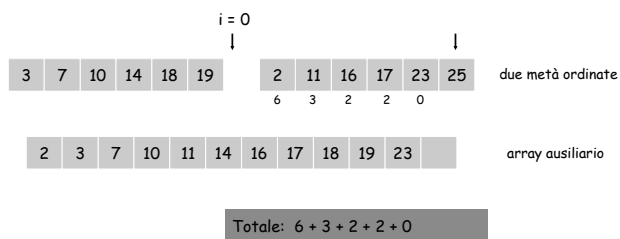


82

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.

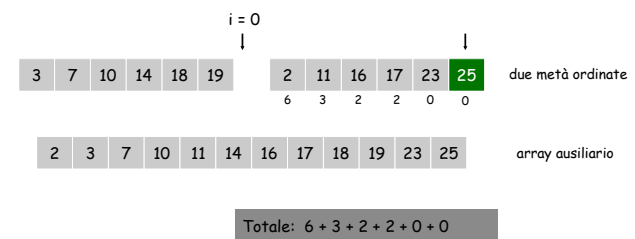


83

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.



84

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.

$i = 0$

| | | | | | | | | | | | |
|---|---|----|----|----|----|---|----|----|----|----|----|
| 3 | 7 | 10 | 14 | 18 | 19 | 2 | 11 | 16 | 17 | 23 | 25 |
| | | | | | | 6 | 3 | 2 | 2 | 0 | 0 |

due metà ordinate

| | | | | | | | | | | | |
|---|---|---|----|----|----|----|----|----|----|----|----|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 17 | 18 | 19 | 23 | 25 |
|---|---|---|----|----|----|----|----|----|----|----|----|

array ausiliario

Totale: $6 + 3 + 2 + 2 + 0 + 0 = 13$

85

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

86

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: $6 + 3 + 2 + 2 + 0 + 0$ Conteggio: $O(n)$

| | | | | | | | | | | | |
|---|---|---|----|----|----|----|----|----|----|----|----|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 17 | 18 | 19 | 23 | 25 |
|---|---|---|----|----|----|----|----|----|----|----|----|

Merge: $O(n)$

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

87

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
    
```

88

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
  (rA, A) ← Sort-and-Count(A)
  (rB, B) ← Sort-and-Count(B)
  (r, L) ← Merge-and-Count(A, B)

  return r = rA + rB + r e la lista ordinata L
}
```

89

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 $\Theta(n^2)$.

91

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?

92

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: $\min\{3, 2, 3, 3, 7\} = 2$

93

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 .

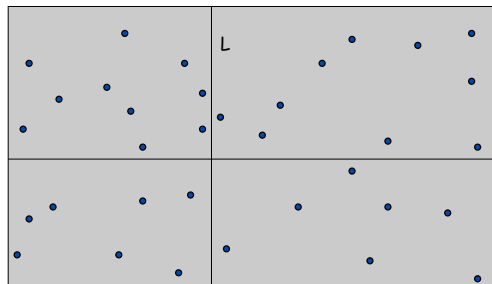
per semplificare la presentazione. Altrimenti rotazione oppure estendere algoritmo

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.

94

Coppia di punti più vicini: Primo tentativo

Divide. Dividiamo la regione in 4 quadranti.

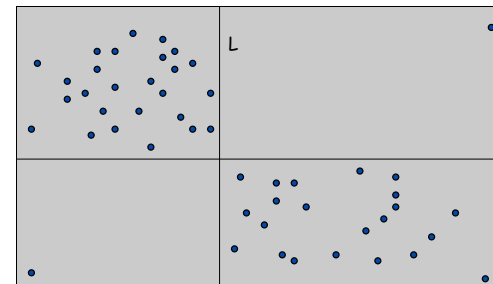


95

Coppia di punti più vicini: Primo tentativo

Divide. Dividiamo la regione in 4 quadranti

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

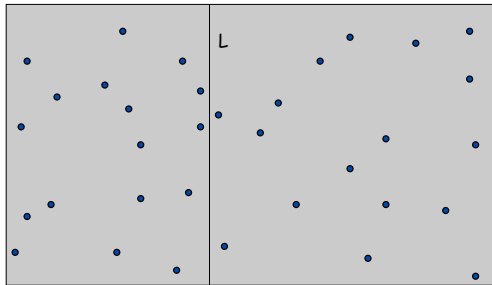


96

Coppia di punti più vicini

Algoritmo.

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

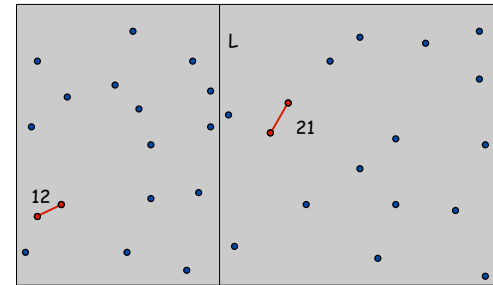


97

Coppia di punti più vicini

Algoritmo.

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

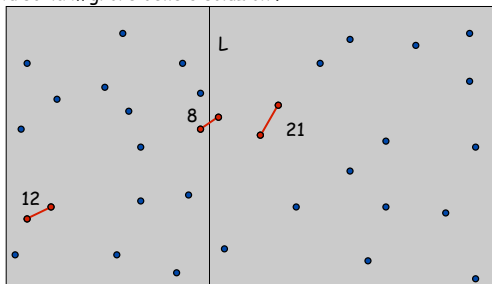


98

Coppia di punti più vicini

Algoritmo.

- **Divide:** disegnare linea verticale L così 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.



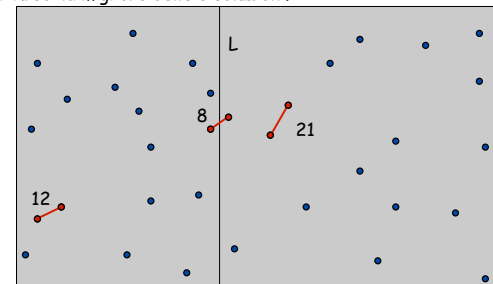
99

Coppia di punti più vicini

Algoritmo.

- **Divide:** disegnare linea verticale L così 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 $\Theta(n^2)$



100

Coppia di punti più vicini

Trovare la coppia più vicina con un punto in ogni parte, **assumendo che la distanza sia $< \delta$** .

$\delta = \min(12, 21)$

101

Coppia di punti più vicini

Trovare la coppia più vicina con un punto in ogni parte, **assumendo che la distanza sia $< \delta$** .

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

$\delta = \min(12, 21)$

102

Coppia di punti più vicini

Trovare la coppia più vicina con un punto in ogni parte, **assumendo che la distanza sia $< \delta$** .

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

$\delta = \min(12, 21)$

103

Coppia di punti più vicini

Trovare la coppia più vicina con un punto in ogni parte, **assumendo che la distanza sia $< \delta$** .

- Osservazione: 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!

$\delta = \min(12, 21)$

104

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 .

105

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 $\max \frac{1}{2}\delta \cdot \sqrt{2} < \delta$).

106

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 $|i - j| \leq 11$.

107

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.

108

Coppia di punti più vicini: algoritmo

```

Closest-Pair( $p_1, \dots, p_n$ ) {
    Calcolare linea di separazione L tale che divide i
    punti in due parti uguali.

     $\delta_1$  = Closest-Pair(metà sinistra)
     $\delta_2$  = Closest-Pair(metà destra)
     $\delta$  = min( $\delta_1, \delta_2$ )

    Cancellare tutti i punti con distanza  $>\delta$  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  $<\delta$ , aggiornare  $\delta$ .

    return  $\delta$ .
}
    
```

109

Coppia di punti più vicini: complessità algoritmo

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

```

Closest-Pair( $p_1, \dots, p_n$ ) {
    Calcolare linea di separazione L tale che divide i
    punti in due parti uguali.
     $\delta_1$  = Closest-Pair(metà sinistra)
     $\delta_2$  = Closest-Pair(metà destra)
     $\delta$  = min( $\delta_1, \delta_2$ )

    Cancellare tutti i punti con distanza  $>\delta$  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  $<\delta$ , aggiornare  $\delta$ .

    return  $\delta$ .
}
    
```

$O(n \log n)$

$2T(n/2)$

$O(n)$

$O(n \log n)$

$O(n)$

110

Coppia di punti più vicini: complessità algoritmo

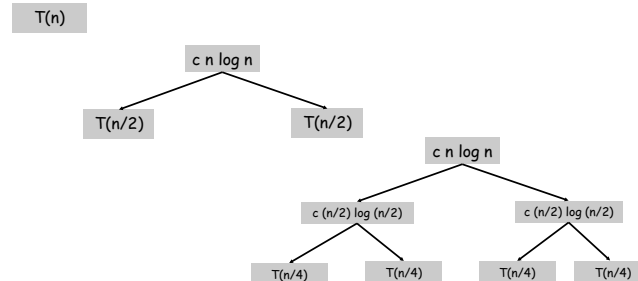
Complessità algoritmo.

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

111

Prova con Albero di Ricorsione

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



112

Prova con Albero di Ricorsione

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

Altezza albero h?

113

Prova con Albero di Ricorsione

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

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

114

Coppia di punti più vicini: complessità algoritmo

Complessità algoritmo.

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

Domanda. Possiamo ottenere $O(n \log n)$?

115

Coppia di punti più vicini: complessità algoritmo

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

```

Closest-Pair( $P_1, \dots, P_n$ ) {
  Calcolare linea di separazione  $L$  tale che divide i punti in due parti uguali.
   $\delta_1 = \text{Closest-Pair}(\text{metà sinistra})$ 
   $\delta_2 = \text{Closest-Pair}(\text{metà destra})$ 
   $\delta = \min(\delta_1, \delta_2)$ 
  Cancellare tutti i punti con distanza  $> \delta$  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  $< \delta$ , aggiornare  $\delta$ .
  return  $\delta$ 
}
    
```

116

Coppia di punti più vicini: complessità algoritmo

Complessità algoritmo.

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

Domanda. Possiamo ottenere $O(n \log n)$?

Risposta. Sì! 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.

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

117

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

```

Closest-Pair(P)
  Construct  $P_x$  and  $P_y$  ( $O(n \log n)$  time)
   $(q_0^x, r_0^x) = \text{Closest-Pair-Rec}(P_x, P_x)$ 
   $(q_0^y, r_0^y) = \text{Closest-Pair-Rec}(P_y, P_y)$ 
   $P_x$  = lista P ordinata rispetto a coordinata x
   $P_y$  = lista P ordinata rispetto a coordinata y

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

Construct  $Q_x, Q_y, R_x, R_y$  ( $O(n)$  time)
 $(q_0^x, q_1^x) = \text{Closest-Pair-Rec}(Q_x, Q_x)$ 
 $(r_0^x, r_1^x) = \text{Closest-Pair-Rec}(R_x, R_x)$ 
 $(q_0^y, q_1^y) = \text{Closest-Pair-Rec}(Q_y, Q_y)$ 
 $(r_0^y, r_1^y) = \text{Closest-Pair-Rec}(R_y, R_y)$ 
   $Q$  = 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  $y$ 

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

Construct  $S_y$  ( $O(n)$  time)
For each point  $s \in S_y$ , 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  $d(s, s') < \delta$  then
  Return  $(s, s')$ 
Else if  $d(q_0^x, q_1^x) < d(r_0^x, r_1^x)$  then
  Return  $(q_0^x, q_1^x)$ 
Else
  Return  $(r_0^x, r_1^x)$ 
Endif
    
```

118

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 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |

Addizione

120

Aritmetica su interi: moltiplicazione

Moltiplicazione. Dati due interi di n-digit a e b, computare a x b.

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

```

      1100
    × 1101
    -----
      1100
     0000
    1100
   1100
  -----
 12    1100
 ×13   1100
  ---
 36    1100
 12    1100
 156   1001100
  -----
156   1001100
  
```

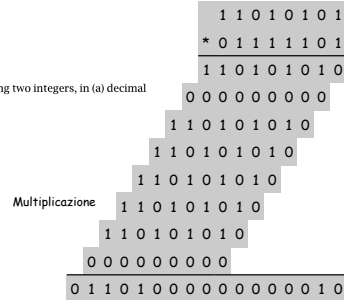


Figure 5.8 The elementary-school algorithm for multiplying two integers, in (a) decimal and (b) binary representation.

121

Moltiplicazione Divide-and-Conquer: Warmup

Per moltiplicare due interi di n-digit:

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

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

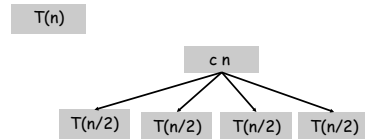
$$T(n) = \underbrace{4T(n/2)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n)}_{\text{addizioni, shift}} \Rightarrow T(n) = \Theta(n^2)$$

↑
assumiamo n potenza di 2

122

Prova con Albero di Ricorsione

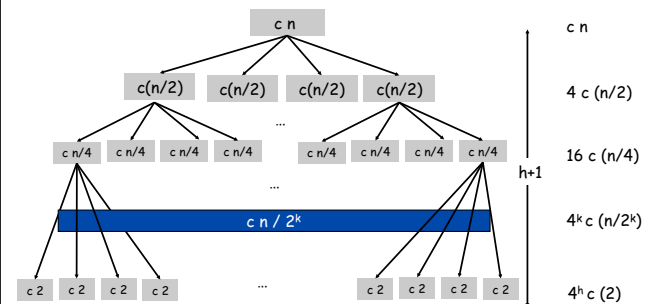
$$T(n) = \underbrace{4T(n/2)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n)}_{\text{addizioni, shift}} \Rightarrow T(n) = \Theta(n^2)$$



123

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?

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

124

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 $h = \log_2 n - 1$ $\sum_{i=0}^{h-1} 4^i \cdot \frac{n}{2^i} = cn \sum_{i=0}^{h-1} 2^i = cn(2^h - 1)$

125

Moltiplicazione: Algoritmo di Karatsuba

Moltiplicazione di due interi con n cifre:

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

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

126

Moltiplicazione: Algoritmo di Karatsuba

Moltiplicazione di due interi con n cifre:

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

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

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

127

Moltiplicazione: Algoritmo di Karatsuba

Moltiplicazione di due interi con n cifre:

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

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

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.

$$T(n) \leq T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + T(\lceil 1 + \lfloor n/2 \rfloor \rceil) + \Theta(n)$$

128

Prova con Albero di Ricorzione

$$T(n) \leq \underbrace{T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + T(\lfloor 1 + \lfloor n/2 \rfloor)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n)}_{\text{addizioni, sottrazioni, shift}}$$

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

$T(n)$

cn

$T(n/2) \quad T(n/2) \quad T(n/2)$

129

Prova con Albero di Ricorzione

$$T(n) \leq \underbrace{T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + T(\lfloor 1 + \lfloor n/2 \rfloor)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n)}_{\text{addizioni, sottrazioni, shift}}$$

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

cn

$3c(n/2)$

$9c(n/4)$

$3^k c(n/2^k)$

$3^h c(2)$

$h+1$

$cn / 2^k$

$c2 \quad c2 \quad c2$

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

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

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

130

Relazione ricorrenza: caso generale con $q > 2$

$$T(n) \leq qT(n/2) + cn$$

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

cn

$qc(n/2)$

$q^2 c(n/4)$

$q^k c(n/2^k)$

$q^h c(2)$

$h+1$

$cn / 2^k$

$c2 \quad c2 \quad c2$

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

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

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

131

Matrix Multiplication

Moltiplicazione di matrici

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

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

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

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

```

for i ← 1 to n
  for j ← 1 to n
    cij ← 0
    for k ← 1 to n
      cij ← cij + aik · bkj
    
```

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

133

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: aggiungere 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?

134

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: aggiungere 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?

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

135

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).

136

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.

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

137

Moltiplicazione di matrici: algoritmi efficienti in teoria

D. Moltiplicazione due matrici 2-per-2 con solo 7 moltiplicazioni scalari?

R. Sì! [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. Sì! [Pan, 1980] $\Theta(n^{\log_{70} 143640}) = O(n^{2.80})$

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

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

I miglioramenti teorici a Strassen sono sempre meno pratici.

138

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**.

140

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.

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.
- ...

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.

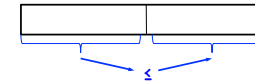
Abbiamo già visto il Mergesort come algoritmo di ordinamento (complessità $O(n \log n)$).

141

Quicksort

Divide-and-conquer.

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



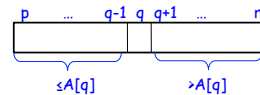
- **Conquer:** ordina ricorsivamente ognuna delle due parti.
- **Combine:** sono già ordinati.

142

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.

143

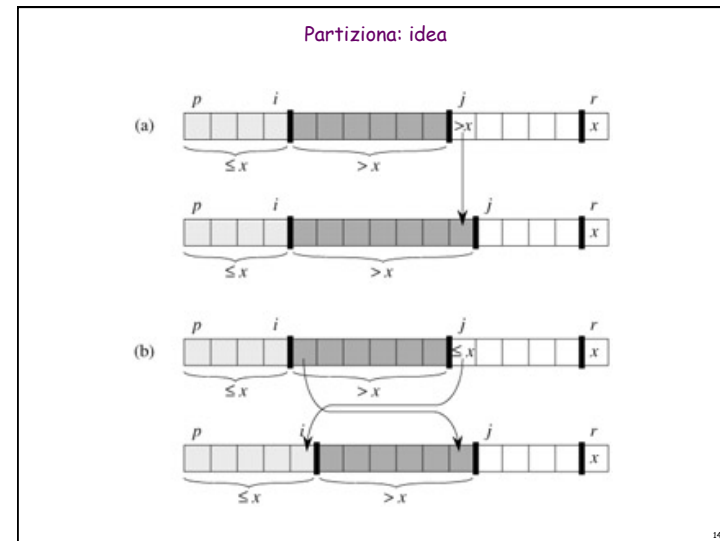
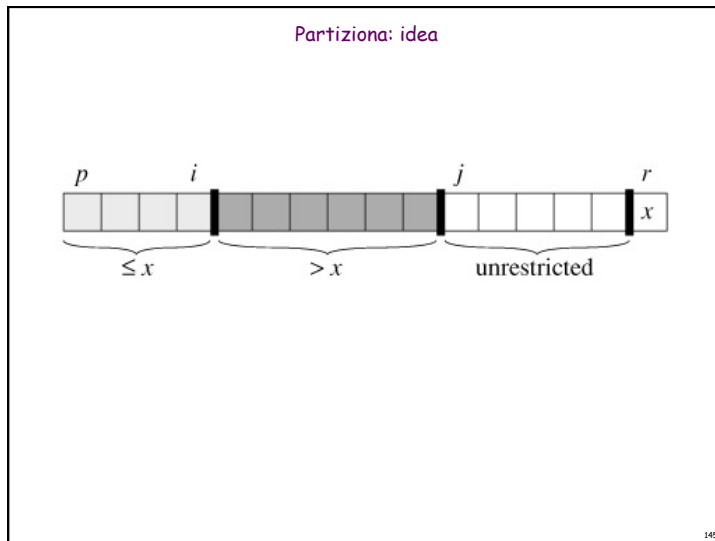
Quicksort: algoritmo

```

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 $\text{Quicksort}(A, 1, \text{length}[A])$.

144

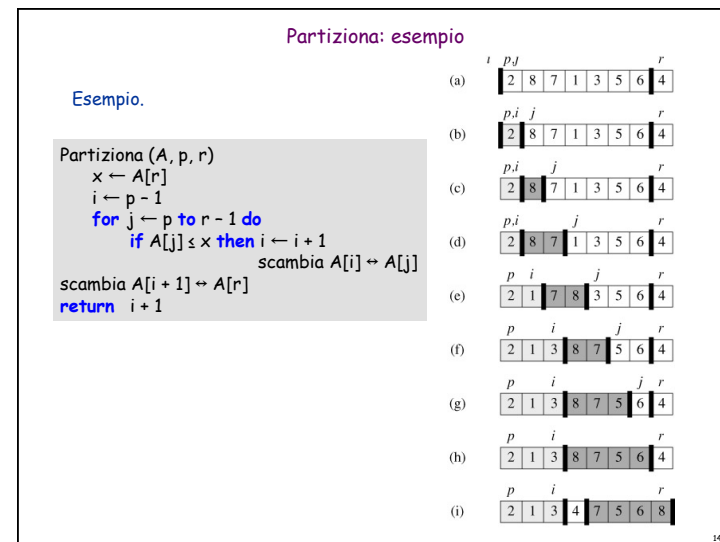


Partiziona: algoritmo

```

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
    
```

147



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 $n = r - p + 1$

149

Partizionamento

Partizionamento: Analizziamo tre casi

- Caso peggiore
- Caso migliore
- Caso medio

150

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

151

Partizionamento: caso migliore

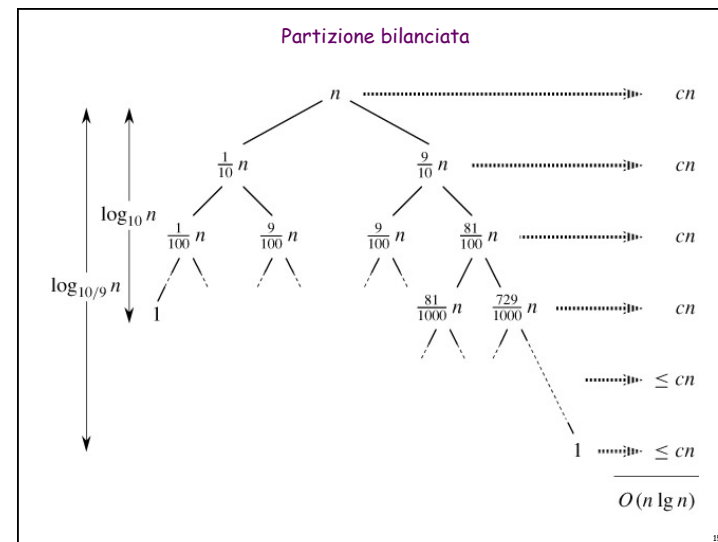
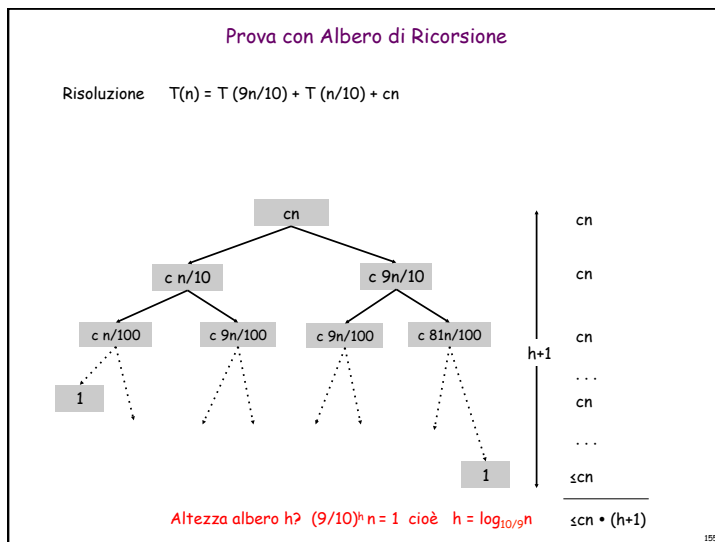
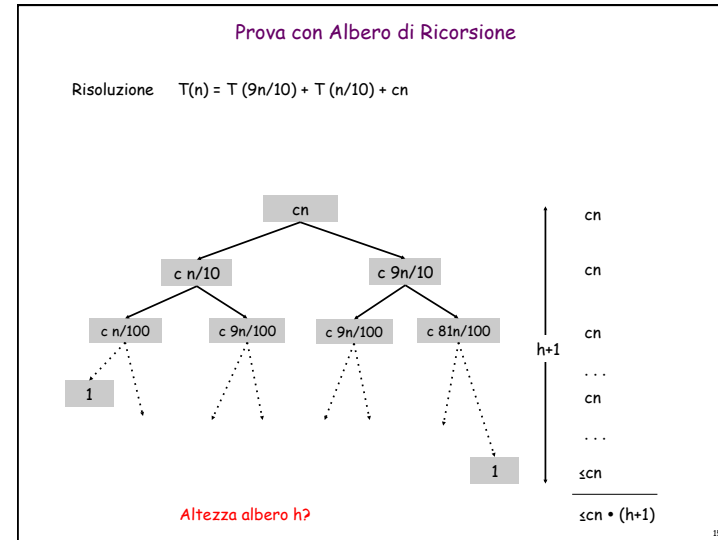
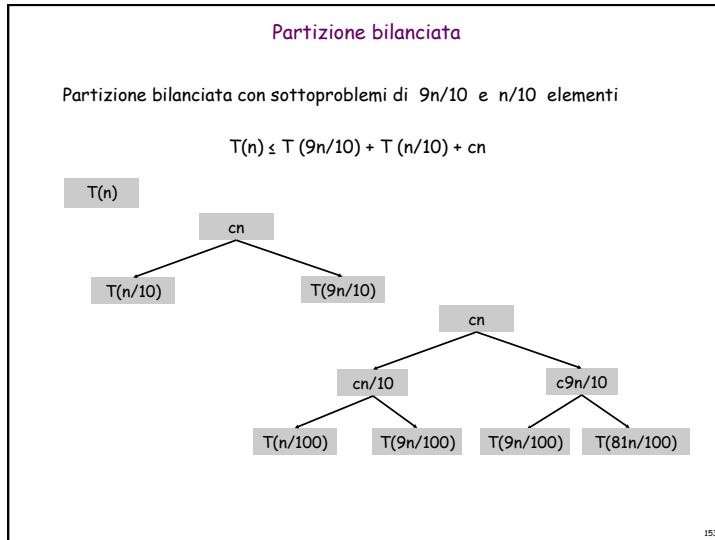
Caso migliore. Partizione bilanciata con sottoproblemi di $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil - 1$

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

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

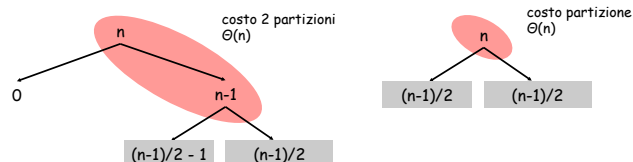
(è la stessa relazione del Mergesort)

152



Partizioni

Partizione sbilanciata seguita da partizione bilanciata



Caso medio:

Intuitivamente è più vicino al caso migliore che al caso peggiore.
Complessità $O(n \log n)$

157

Quicksort: analisi caso peggiore

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

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

$$\begin{aligned} \text{Sostituendo: } T(n) &\leq \max_{0 \leq q \leq n-1} (cq^2 + c(n-q-1)^2) + \Theta(n) \\ &\leq c \cdot \max_{0 \leq q \leq n-1} (q^2 + (n-q-1)^2) + \Theta(n) \end{aligned}$$

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

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

Quindi,

$$\begin{aligned} T(n) &\leq cn^2 - c(2n-1) + \Theta(n) \\ &\leq cn^2 \end{aligned}$$

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

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

158

Esercizio 7.2-3

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

159

Esercizio 7.2-5

Supponiamo che la partizione ad ogni livello è proporzionale 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.)

160

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)

```

161

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.

162

Esercizi

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

Esercizi da risolvere in forma completa a casa

163

Esercizio: Potenze di un numero

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

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

164

Esercizio: Potenze di un numero

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

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

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

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

165

Esercizio: Potenze di un numero

Soluzione relazione ricorrenza con metodo di sostituzione

$$\begin{aligned} T(n) &= T(n/2) + c && \text{con } n = 2^k \\ &= T(n/4) + c + c \\ &= T(n/4) + 2c \\ &= T(n/8) + c + 2c \\ &= T(n/8) + 3c \\ &= T(n/16) + c + 3c \\ &= T(n/16) + 4c \\ &= \dots \\ &= T(n/2^k) + kc \\ &= \Theta(k) \\ &= \Theta(\log n) \end{aligned}$$

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

166

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 $\Theta(\log n)$

167

Esercizio: Numeri di Fibonacci

Input: $n \in \mathbb{N}$
 Calcolare: F_n

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

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 $\Theta(\log n)$

168

Esercizio: Massima Sottosequenza

Input: sequenza $A[1], A[2], \dots, A[n]$

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 $\Theta(n \log n)$

169

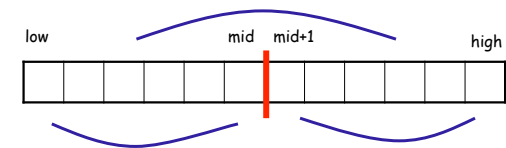
Esercizio: Massima Sottosequenza

Input: sequenza $A[1], A[2], \dots, A[n]$

Calcolare: sottosequenza con somma massima

Divide-and-conquer.

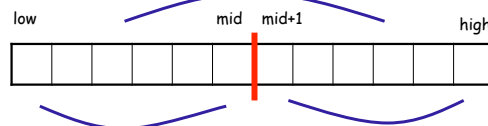
- **Divide:** dividere sequenza in due parti della stessa grandezza.
- **Conquer:** risolvere ricorsivamente il problema in ogni parte.
- **Combine:** calcolare la sottosequenza con somma massima che contiene il punto di mezzo (cioè con inizio e fine in parti diverse), e restituire il massimo delle 3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



170

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

```

171

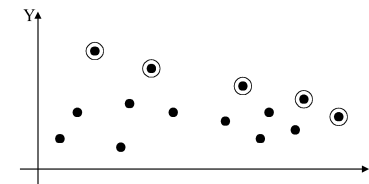
Esercizio: Massimi in 2-D

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 $\Theta(n \log n)$

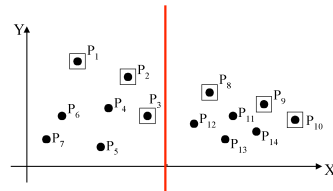
172

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:**



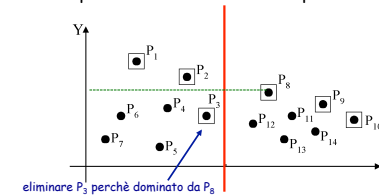
173

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.



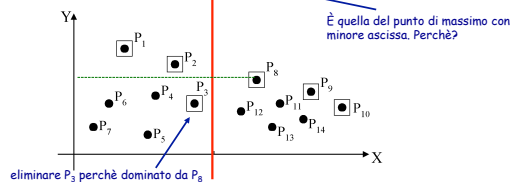
174

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.

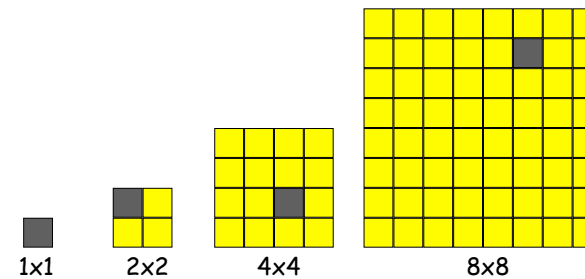


175

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



176

Esercizio: Defective Chessboards

Input: scacchiera $n \times n$ e casella non disponibile
Calcolare: ricoprimento con triomini

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

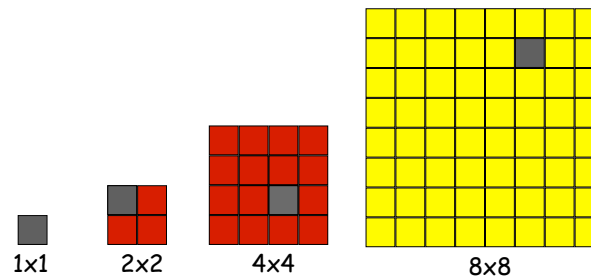
Ha 4 orientazioni:



177

Esercizio: Defective Chessboards

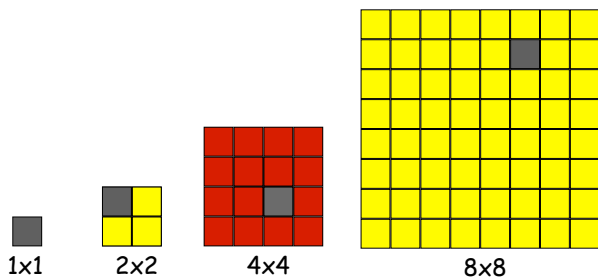
Input: scacchiera $n \times n$ e casella non disponibile
Calcolare: ricoprimento con triomini, cioè posizionamento di $(n^2-1)/3$ triomini sulla scacchiera in modo da ricoprire tutte le caselle disponibili.



178

Esercizio: Defective Chessboards

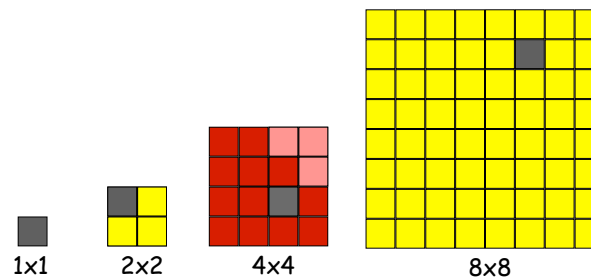
Input: scacchiera $n \times n$ e casella non disponibile
Calcolare: ricoprimento con triomini, cioè posizionamento di $(n^2-1)/3$ triomini sulla scacchiera in modo da ricoprire tutte le caselle disponibili.



179

Esercizio: Defective Chessboards

Input: scacchiera $n \times n$ 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.



180

Esercizio: Defective Chessboards

Input: scacchiera $n \times n$ 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.

1x1 2x2 4x4 8x8

181

Esercizio: Defective Chessboards

Input: scacchiera $n \times n$ 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.

1x1 2x2 4x4 8x8

182

Esercizio: Defective Chessboards

Input: scacchiera $n \times n$ 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.

1x1 2x2 4x4 8x8

183

Esercizio: Defective Chessboards

Input: scacchiera $n \times n$ 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.

1x1 2x2 4x4 8x8

184

Esercizio: Defective Chessboards

Input: scacchiera $n \times n$ 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.

1x1 2x2 4x4 4x4 4x4

185

Esercizio: Defective Chessboards

Input: scacchiera $n \times n$ 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.

8x8

Divide-and-conquer.
 □ **Divide:** dividere scacchiera in quattro scacchiere.

186

Esercizio: Defective Chessboards

Input: scacchiera $n \times n$ 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.

8x8

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

187

Esercizio: Defective Chessboards

Soluzione relazione ricorrenza con metodo di sostituzione

$$\begin{aligned}
 T(n) &= 4 T(n/2) + c && \text{con } n=2^k \\
 &= 4 (4 T(n/4) + c) + c \\
 &= 4^2 T(n/4) + 4c + c \\
 &= 4^2 (4 T(n/8) + c) + 4c + c \\
 &= 4^3 T(n/8) + 4^2c + 4c + c \\
 &= 4^3 (4 T(n/16) + c) + 4^2c + 4c + c \\
 &= 4^4 T(n/16) + 4^3c + 4^2c + 4c + c \\
 &= \dots \\
 &= 4^k T(n/2^k) + 4^{k-1}c + 4^{k-2}c + \dots + 4^3c + 4^2c + 4c + c \\
 &= \Theta(4^k) \\
 &= \Theta(n^2)
 \end{aligned}$$

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

188

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

