# Relazioni di ricorrenza,
# divide et impera, ordinamento in $O(n \log n)$

Algoritmi e strutture dati

Ugo de'Liguoro, Andras Horvath

1

## Relazioni di ricorrenza

- calcolo ricorsivo del fattoriale

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n-1)! & \text{altrimenti} \end{cases}$$

- algoritmo corrispondente ha tempo di calcolo che soddisfa la seguente relazione di ricorrenza:

$$T(n) = \begin{cases} c & \text{se } n = 0 \\ T(n-1) + d & \text{altrimenti} \end{cases}$$

La funzione tempo di un algoritmo ricorsivo è ricorsiva e può essere descritta tramite una **relazione di ricorrenza**.

Qual è l'ordine di grandezza di $T(n)$?

2

## Relazioni lineari a partizione costante

$$
\begin{aligned}
T(n) &= T(n-1) + d \\
&\qquad\qquad \text{(sappiamo che } T(n-1) = T(n-2) + d\text{)} \\
&= T(n-2) + d + d \\
&= T(n-2) + 2d \\
&\;\;\dots \\
&= T(n-k) + kd \qquad\qquad \text{con} \;\; k \le n
\end{aligned}
$$

scegliendo $k = n$

$$= T(0) + nd = c + nd \in \Theta(n)$$

3

## Metodi di soluzione

- metodo dell'**iterazione**: applicare ripetutamente la ricorrenza fino a trovarne la soluzione
- abbiamo applicato il metodo dell'iterazione sul lucido precedente
- metodo della **sostituzione**: ipotizzare una soluzione e applicare il principio di induzione per verificare la soluzione ipotizzata
- lo vediamo sul lucido successivo

4

## Metodo della sostituzione

- consideriamo la relazione di ricorrenza
$$T(n) = \begin{cases} c & \text{se } n = 0 \\ T(n-1) + d & \text{altrimenti} \end{cases}$$
- dimostriamo con induzione che la soluzione sia
$$T(n) = c + nd$$
- caso base: $c + 0 \cdot d = c = T(0)$ ok!
- passo induttivo: dobbiamo dimostrare che
$$T(n) = c + nd \Rightarrow T(n+1) = c + (n+1)d$$
  - secondo la relazione di ricorrenza:
$$T(n+1) = T(n) + d =$$
  - utilizzando l'ipotesi induttiva:
$$= c + nd + d = c + (n+1)d$$
ok!

5

2

## Relazioni lineari a partizione costante

$$T(n) = T(n-1) + d$$

Min-Ric$(A, i)$
 ▷ Pre: $0 < n = length(A)$, $1 \le i \le n$
 ▷ Post: ritorna il minimo in $A[i..n]$
**if** $i = length(A)$ **then** ▷ $A[n..n]$ ha l'unico el. $A[n]$
 **return** $A[i]$
**else**
 **return** $\min(A[i], \text{Min-Ric}(A, i+1))$
**end if**

Algoritmi ricorsivi di scansione di una struttura lineare hanno questa struttura.

6

3

# Analisi dell'algoritmo di Hanoi

```
MoveTower(n, A, B, C)
if n ≥ 1 then
    MoveTower(n − 1, A, C, B)
    move 1 disk from A to C
    MoveTower(n − 1, B, A, C)
```

Tempo di calcolo:

$$T(n) = \begin{cases} b & n = 0 \\ cT(n-1) + d & n \geq 1 \end{cases}$$

dove $c = 2$ nel caso delle torri di Hanoi ma consideriamo un generale intero $c > 1$ (con $c > 1$ la ricorrenza è identica a quella precedente).

7

# Metodo dell'iterazione

$$T(n) = cT(n-1) + d$$

8

# Metodo dell'iterazione

$$
\begin{aligned}
T(n) &= cT(n-1) + d \\
&= c(cT(n-2) + d) + d = c^2T(n-2) + cd + d = \\
&= c^2(cT(n-3) + d) + cd + d = c^3T(n-3) + c^2d + cd + d =
\end{aligned}
$$

dopo $k \leq n$ iterazioni:

$$= c^kT(n-k) + (c^{k-1} + c^{k-2} + \cdots + c + 1)d$$

dopo $n$ iterazioni:

$$= c^nT(0) + (c^{n-1} + c^{n-2} + \cdots + c + 1)d$$

con $c > 1$ :

$$= c^nb + \frac{c^n - 1}{c - 1}d$$

9

# Metodo della sostituzione

- sul lucido precedente abbiamo trovato la soluzione

$$T(n) = c^nb + \frac{c^n - 1}{c - 1}d$$

- verifichiamo la soluzione con induzione
- caso base: $c^0b + \frac{c^0 - 1}{c - 1}d = b = T(0)$ ok!
- passo induttivo: dobbiamo dimostrare l'implicazione

$$T(n) = c^nb + \frac{c^n - 1}{c - 1}d \quad \Rightarrow \quad T(n+1) = c^{n+1}b + \frac{c^{n+1} - 1}{c - 1}d$$

10

# Metodo della sostituzione

secondo la relazione di ricorrenza:

$$T(n+1) = cT(n) + d$$

11

# Metodo della sostituzione

secondo la relazione di ricorrenza:

$$T(n+1) = cT(n) + d$$

utilizzando l'ipotesi induttiva:

$$= c\left(c^n b + \frac{c^n - 1}{c-1}d\right) + d$$

$$= c^{n+1}b + \frac{c^{n+1} - c}{c-1}d + d$$

$$= c^{n+1}b + \frac{c^{n+1} - c + c - 1}{c-1}d$$

$$= c^{n+1}b + \frac{c^{n+1} - 1}{c-1}d$$

12

# Analisi dell'algoritmo di Hanoi

- la soluzione è

$$T(n) = c^n b + \frac{c^n - 1}{c - 1} d$$

- cui ordine di grandezza è $c^n$, cioè $T(n) \in \Theta(c^n)$
- se ci interessa **l'ordine di grandezza del tempo di calcolo** di un algoritmo che stampi le mosse a video allora $b = 1, c = 2$ e $d = 1$ con le quali otteniamo $2^n$
- se ci interessa il **numero di mosse necessarie** allora $b = 0, c = 2$ e $d = 1$ con le quali otteniamo $2^n - 1$

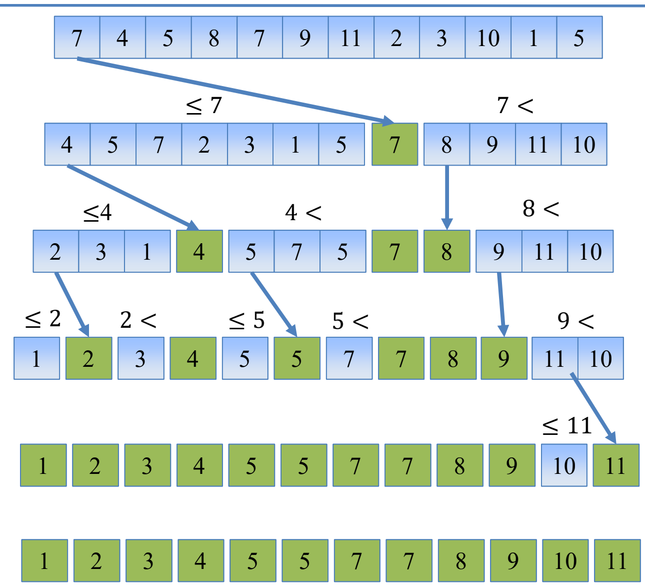
13

# Quick-Sort

- l'idea dell'algoritmo dato $A[1..n]$ :
  - se $n \leq 1$ non fare niente (il vettore è ordinato)
  - scegli un elemento del vettore, chiamato perno, sia il valore di questo elemento $q$
  - riorganizza il vettore in modo tale da avere all'inizio elementi $\leq q$, seguito da $q$ e in fondo gli elementi $> q$
  - questo implica che $q$ è al posto giusto
  - sia $p$ la posizione di $q$, e quindi abbiamo
    $A[1..p-1] \leq A[p] < A[p+1..n]$
  - ripeti tutto su $A[1..p-1]$ e $A[p+1..n]$

14

# Quick-Sort




15

# Partizionamento

- il partizionamento può essere effettuato in tanti modi
- di seguito sviluppiamo un algoritmo che lo effettua
- questo algoritmo non è quello che vedete applicato sul lucido precedente!

16

# Partizionamento

- l'idea del partizionamento di $A[1..n]$ :
  - come perno scegliamo $A[1]$
  - due indici per seguire il partizionamento, $i$ e $j$, tali che
    - elementi in $A[2..i-1]$ sono già esaminati e $A[2..i-1] \leq A[1]$
    - elementi in $A[i..j]$ sono elementi da esaminare
    - elementi in $A[j+1..n]$ sono già esaminati e $A[1] < A[j+1..n]$

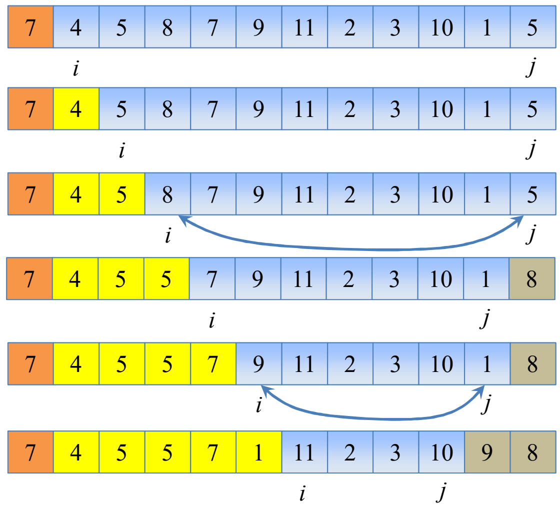
17

# Partizionamento

- l'idea del partizionamento di $A[1..n]$ (cont.):
  - si parte con $i = 2, j = n$
  - $i$ e $j$ si spostano secondo le regole se $i \leq j$:
    - se $A[i] \leq A[1]$ incrementa $i$
    - se $A[i] > A[1] \land A[j] > A[1]$ decrementa $j$
    - se $A[i] > A[1] \land A[j] \leq A[1]$ scambia $A[i]$ e $A[j]$, incrementa $i$, decrementa $j$
  - quando per la prima volta $i > j$ scambia $A[1]$ e $A[j]$

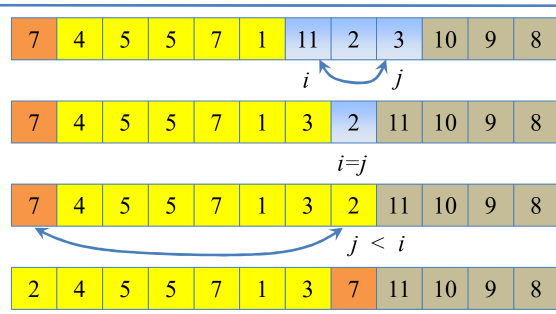
18

# Partizionamento




19

# Partizionamento




20

# Partizionamento

```
PARTITION(A[1..n])
i ← 2, j ← n
while i ≤ j do
    if A[i] ≤ A[1] then
        i ← i + 1
    else
        if A[j] > A[1] then
            j ← j − 1
        else
            scambia A[i] con A[j]
            i ← i + 1, j ← j − 1
        end if
    end if
end while
scambia A[1] con A[j]
return j
```

21

# Partizionamento, correttezza

- correttezza con invariante di ciclo:
$$A[2..i-1] \leq A[1] \wedge A[1] < A[j+1..n]$$
- inizializzazione: $i = 2, j = n$, i due sottovettori sono vuoti ok!
- mantenimento: il ciclo si esegue solo se $i \leq j$ e effettua <u>una e una sola</u> operazione fra i seguenti
  - se $A[i] \leq A[1]$ incrementa $i$, altrimenti
  - se $A[1] < A[j]$ decrementa $j$, altrimenti
  - scambia $A[i]$ e $A[j]$, incrementa $i$, decrementa $j$
- è evidente che con tutte le tre $A[2..i-1] \leq A[1] \wedge A[1] < A[j+1..n]$ viene mantenuto ok!
- all'uscita: $i = j + 1$ ($j = i - 2$ non può succedere) l'invariante implica che
$$A[j] \leq A[1] < A[j+1]$$
e quindi lo scambio finale e l'invariante garantiscono
$$A[1..j-1] \leq A[j] < A[j+1..n]$$
e quindi è corretto restituire $j$ (il perno finisce nella posizione $j$) ok!

22

# Quick-Sort

```
QUICK-SORT(A[1..n])
if n > 1 then
    p ← PARTITION(A[1..n])
    if p > 2 then
        QUICK-SORT(A[1..p − 1])
    end if
    if p < n − 1 then
        QUICK-SORT(A[p + 1..n])
    end if
end if
```

23

# Quick-Sort, correttezza

- correttezza con induzione completa assumendo che il partizionamento funzioni correttamente
- caso base: con $n \leq 1$ il vettore in input è ordinato ok!
- passo induttivo:
  corretto con dimensione $< n \Rightarrow$ corretto con dimensione $= n$
  - dalla correttezza del partizionamento dopo la chiamata a Partition:
    $A[1..p-1] \leq A[p] < A[p+1..n]$
  - dall'ipotesi induttiva:
    - se $A[1..p-1]$ ha più di 1 elemento, sarà ordinato correttamente con la prima chiamata ricorsiva di Quick-Sort perché ha meno di $n$ elementi
    - se $A[p+1..n]$ ha più di 1 elemento, sarà ordinato correttamente con la seconda chiamata ricorsiva di Quick-Sort perché ha meno di $n$ elementi
  - segue che $A[1..n]$ è ordinato ok!

24

# Quick-Sort, complessità

- complessità del partizionamento:
  - Partition scansiona una volta il vettore su cui opera (una parte da sinistra e l'altra da destra)
  - l'ordine di grandezza del tempo di calcolo (ovvero il numero di operazioni) è lineare
  - porteremo avanti i calcoli con
$$T_P(n) = an$$
con $a$ costante

25

# Quick-Sort, complessità

- dopo il partizionamento le due chiamate ricorsive lavorano con dimensione $p-1$ e $n-p$
- le due situazioni "estremi" sono
  - $A[1..p-1]$ e $A[p+1..n]$ hanno circa lo stesso numero di elementi
  - $A[1..p-1]$ ha $n-1$ elementi e $A[p+1..n]$ è vuoto (o vice versa)
- le seconda situazione, con **partizioni sbilanciate**, dà luogo ad una relazione di ricorrenza simile a quelle già studiate:

$$T(n) = \begin{cases} c & n = 1 \\ T(n-1) + T_P(n) + b & n > 1 \end{cases} = \begin{cases} c & n = 1 \\ T(n-1) + an + b & n > 1 \end{cases}$$

26

## Quick-Sort, complessità

- con metodo dell'iterazione:

$$
\begin{aligned}
T(n) &= T(n-1) + an + b \\
&= T(n-2) + a(n-1) + b + an + b = \\
&= T(n-3) + a(n-2) + b + a(n-1) + b + an + b =
\end{aligned}
$$

dopo $k \leq n$ iterazioni:

$$
= T(n-k) + a\sum_{i=0}^{k-1}(n-i) + kb
$$

dopo $n-1$ iterazioni:

$$
\begin{aligned}
&= T(1) + a\sum_{i=0}^{n-2}(n-i) + (n-1)b \\
&= c + a\sum_{i=0}^{n-2}(n-i) + (n-1)b = c + a\sum_{i=2}^{n} i + (n-1)b
\end{aligned}
$$

- quindi $T(n) \in \Theta(n^2)$ (che è il suo caso peggiore come vedremo)

27

## Relazioni lineari a partizione costante

I coefficienti $a_i$ sono costanti

Partizione costante: $i$ è limitato da $h$

$$
T(n) = \begin{cases} d & \text{se } n \leq m \leq h \\ \sum\limits_{1 \leq i \leq h} a_i T(n-i) + cn^{\beta} & \text{se } n > m \end{cases}
$$

Ordine costante h

Lavoro polinomiale di divisione/combinazione

Linare: n ha grado 1

28

# Relazioni lineari a partizione costante

$$T(n) = \begin{cases} d & \text{se } n \le m \le h \\ \sum_{1 \le i \le h} a_i T(n-i) + cn^{\beta} & \text{se } n > m \end{cases}$$

Teorema. Se $c > 0, \beta \ge 0, a = \sum_{1 \le i \le h} a_i$ allora $\begin{cases} T(n) \in O(n^{\beta+1}) & \text{se } a = 1 \\ T(n) \in O(a^n n^{\beta}) & \text{se } a \ge 2 \end{cases}$

Il teorema si chiama teorema master per relazioni lineari a partizione costante.

29

# Relazioni lineari a partizione costante

$$T(n) = \begin{cases} d & \text{se } n \le m \le h \\ \sum_{1 \le i \le h} a_i T(n-i) + cn^{\beta} & \text{se } n > m \end{cases}$$

Teorema. Se $c > 0, \beta \ge 0, a = \sum_{1 \le i \le h} a_i$ allora $\begin{cases} T(n) \in O(n^{\beta+1}) & \text{se } a = 1 \\ T(n) \in O(a^n n^{\beta}) & \text{se } a \ge 2 \end{cases}$

Quick-Sort (caso peggiore):

con $h = 1, a = a_1 = 1, \beta = 1, c = 1$

$$\boxed{T(n) \in O(n^{\beta+1}) = O(n^2)}$$

30

# Relazioni lineari a partizione costante

$$T(n) = \begin{cases} d & \text{se } n \leq m \leq h \\ \sum_{1 \leq i \leq h} a_i T(n-i) + cn^\beta & \text{se } n > m \end{cases}$$

Teorema. Se $c > 0, \beta \geq 0, a = \sum_{1 \leq i \leq h} a_i$ allora $\begin{cases} T(n) \in O(n^{\beta+1}) & \text{se } a = 1 \\ T(n) \in O(a^n n^\beta) & \text{se } a \geq 2 \end{cases}$

Minimo ricorsivo:

con $h = 1, a = a_1 = 1, \beta = 0, c = 1$

$$\boxed{T(n) \in O(n^{\beta+1}) = O(n)}$$

31

# Relazioni lineari a partizione costante

$$T(n) = \begin{cases} d & \text{se } n \leq m \leq h \\ \sum_{1 \leq i \leq h} a_i T(n-i) + cn^\beta & \text{se } n > m \end{cases}$$

Teorema. Se $c > 0, \beta \geq 0, a = \sum_{1 \leq i \leq h} a_i$ allora $\begin{cases} T(n) \in O(n^{\beta+1}) & \text{se } a = 1 \\ T(n) \in O(a^n n^\beta) & \text{se } a \geq 2 \end{cases}$

Torri di Hanoi:

con $h = 1, a = a_1 = 2, \beta = 0, c = 1$

$$\boxed{T(n) \in O(a^n n^\beta) = O(2^n)}$$

32

# Divide et Impera

"Per meglio dominare occorre dividere gli avversari"
Ossia: suddividi il problema in sottoproblemi di dimensione circa uguale; risolvi i sottoproblemi in maniera ricorsiva, infine combina i risultati

DivideEtImpera ($P$, $n$) // Pre: $n$ è la dimensione di $P$

**if** $n \le k$ **then** risolvi direttamente $P$

**else**

dividi $P$ nei sottoproblemi $P_1,\ldots, P_h$ di dimensioni $n_1,\ldots, n_h$

**for** $i \leftarrow 1$ **to** $h$ **do**

$R_i \leftarrow$ DivideEtImpera ($P_i$, $n_i$)

**return** combinazione di $R_1, \ldots, R_h$

33

# Divide et Impera

"Per meglio dominare occorre dividere gli avversari"
Ossia: suddividi il problema in sottoproblemi di dimensione circa uguale; risolvi i sottoproblemi in maniera ricorsiva, infine combina i risultati

$$T(n) = \begin{cases} d & \text{se } n \le k \\ \underbrace{D(n)}_{\text{Tempo per dividere}} + \underbrace{C(n)}_{\text{Tempo per combinare}} + \underbrace{\sum_{i=1}^{h} T(n_i)}_{\text{Somma dei tempi dei sottoproblemi}} & \text{se } n > k \end{cases}$$

34

# Minimo e Massimo

L'algoritmo **DI-Min-Max** calcola il min e max in A[p..q].

```
DI-Min-Max (A, p, q)
if p = q then return (A[p], A[p])
if p = q – 1 then
      if A[p] < A[q] then return (A[p], A[q])
      else return (A[q], A[p])
r ← (p + q)/2
(min1, max1) ← DI-Min-Max (A, p, r)
(min2, max2) ← DI-Min-Max (A, r +1, q)
return (min (min1, min2), max(max1, max2))
```

Quanti sono i confronti?



# Minimo e Massimo

Se $n$ = q – p + 1 allora i confronti $C(n)$ sono:

$$C(n) = \begin{cases} 0 & \text{se } n = 1 \\ 1 & \text{se } n = 2 \\ C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + 2 & \text{se } n > 2 \end{cases}$$

Per $n = 2^k$ con $k \geq 1$ con il metodo dell'iterazione:

$$C(n) = 2C\left(\frac{n}{2}\right) + 2 = 2\left(2C\left(\frac{n}{4}\right) + 2\right) + 2 = 4C\left(\frac{n}{4}\right) + 4 + 2 =$$

$$= 4\left(2C\left(\frac{n}{8}\right) + 2\right) + 4 + 2 = 8C\left(\frac{n}{8}\right) + 8 + 4 + 2 = \cdots$$

$$= 2^{j}C\left(\frac{n}{2^{j}}\right) + 2^{j} + 2^{j-1} + \cdots + 2 =$$

con $\frac{n}{2^{j}} = 2 \Rightarrow j = \log_2 n - 1$

$$C(n) = 2^{\log_2 n - 1}C\left(\frac{n}{2^{\log_2 n-1}}\right) + 2^{\log_2 n - 1} + 2^{\log_2 n - 2} + \cdots + 2 =$$

# Minimo e Massimo

~~Se $n$ = q – p + 1 allora i confronti $C(n)$ sono:~~

$$C(n) = \begin{cases} 0 & \text{se } n = 1 \\ 1 & \text{se } n = 2 \\ C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + 2 & \text{se } n > 2 \end{cases}$$

Per $n = 2^k$ con $k \geq 1$ con il metodo dell'iterazione (cont.):

$$C(n) = 2^{\log_2 n-1} C\left(\frac{n}{2^{\log_2 n-1}}\right) + 2^{\log_2 n-1} + 2^{\log_2 n-2} + \cdots + 2 =$$

$$= \frac{n}{2}C(2) + \frac{n}{2} + \frac{n}{4} + \cdots + 2 = \frac{n}{2} + \left(\frac{n}{2} + \frac{n}{4} + \cdots + 2 + 1\right) - 1 =$$

$$\cancel{= \frac{n}{2} + \left(2^{k-1} + 2^{k-2} + \cdots + 2 + 1\right)} - 1 =$$

$$= \frac{n}{2} + n - 1 - 1 = \frac{3}{2}n - 2$$

37

# Minimo e Massimo

~~Se $n$ = q – p + 1 allora i confronti $C(n)$ sono:~~

$$C(n) = \begin{cases} 0 & \text{se } n = 1 \\ 1 & \text{se } n = 2 \\ C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + 2 & \text{se } n > 2 \end{cases}$$

~~Per $n = 2^k$ con $k \geq 1$ abbiamo ottenuto $C(n) = \frac{3}{2}n - 2$.~~

Controlliamo con induzione se va bene:

Caso base: $C(2) = \frac{3}{2}2 - 2 = 3 - 2 = 1$ ok!

Passo induttivo: $C\left(\frac{n}{2}\right) = \frac{3}{2} \cdot \frac{n}{2} - 2 \;\Rightarrow C(n) = \frac{3}{2}n - 2$

$$C(n) = 2C\left(\frac{n}{2}\right) + 2 = 2\left(\frac{3}{2} \cdot \frac{n}{2} - 2\right) + 2 = \frac{3}{2}n - 4 + 2 = \frac{3}{2}n - 2$$

ok!

38

## Relazioni lineari a partizione bilanciata

```
BinSearch-Ric(x, A, i, j)
      ▷ Pre: A[i..j] ordinato
      ▷ Post: true se x ∈ A[i..j]
if i > j then        ▷ A[i..j] = ∅
    return false
else
    m ← ⌊(i + j)/2⌋
    if x = A[m] then
        return true
    else
        if x < A[m] then
            return BinSearch-Ric(x, A, i, m − 1)
        else      ▷ A[m] < x
            return BinSearch-Ric(x, A, m + 1, j)
        end if
    end if
end if
```



## Relazioni lineari a partizione bilanciata

$$T(n) = \begin{cases} c & \text{se } n \leq 1 \\ T\left(\frac{n}{2}\right) + d & \text{altrimenti} \end{cases}$$

(in 1 2 ricerca binarie semplici)

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + d \\ &= T\left(\frac{n}{4}\right) + d + d = T\left(\frac{n}{4}\right) + 2d \\ &\dots \\ &= T\left(\frac{n}{2^k}\right) + dk \quad \text{se} \quad k \leq \log_2 n \end{aligned}$$

con $k = \log_2 n$ (assumiamo $n$ sia potenza di 2)

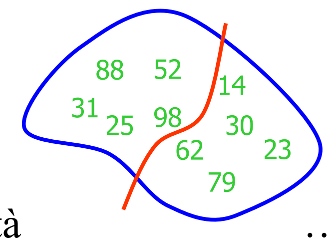
$$= T(1) + d \cdot \log_2 n = c + d \cdot \log_2 n$$

quindi

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

(con $T(n) = T(n/b) + d$ viene lo stesso ordine di grandezza)

# Ordinamento per fusione




dividi l’insieme in due

Affida una metà ad un amico …

… e l’altra metà ad un altro

25,31,52,88,98

14,23,30,62,79

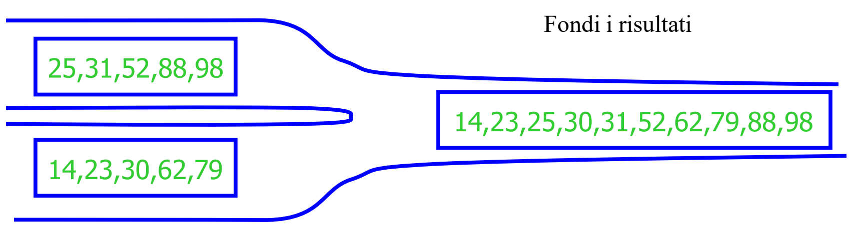
41

# Ordinamento per fusione

25,31,52,88,98

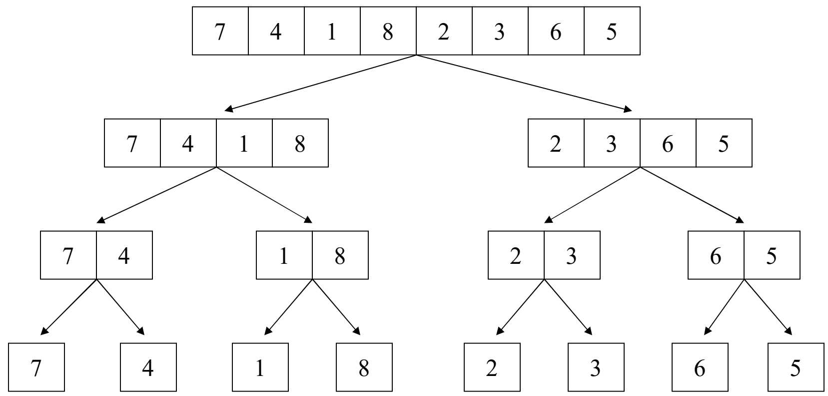
14,23,30,62,79

Fondi i risultati



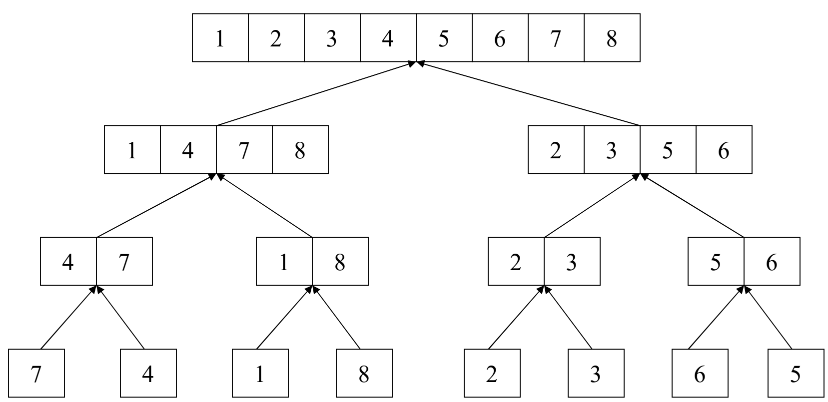

42

# Merge Sort: esecuzione




43

# Merge Sort: esecuzione




44

# Ordinamento per fusione

```
MERGE-SORT(A)
if length(A) = 1 then
    return A
else
    k ← ⌊length(A)/2⌋
    B ← MERGE-SORT(A[1..k])
    C ← MERGE-SORT(A[k + 1..length(A)])
    return MERGE(B, C)
end if
```

45

# Ordinamento per fusione

```
MERGE(B, C)
if B = [] then
    return C
else
    if C = [] then
        return B
    else
        if B[1] ≤ C[1] then
            return [B[1], MERGE(B[2..length(B)], C)]
        else
            return [C[1], MERGE(B, C[2..length(C)])]
        end if
    end if
end if
```

$T_{Merge}(n) = T_{Merge}(n-1) + d$

dove $n$ è il numero totale di elementi in $B$ e $C$

46

# Ordinamento per fusione

MergeSort($A$, *primo*, *ultimo*)
// Pre: $A$ è un vettore, $primo \leq ultimo <$ dimensione di $A$
// Post: ordina $A$ in senso non decrescente
**if** $primo < ultimo$ **then**
    $mezzo \leftarrow \lfloor(primo + ultimo) / 2\rfloor$
    MergeSort($A$, *primo*, *mezzo*)
    MergeSort($A$, $mezzo + 1$, *ultimo*)
    Merge($A$, *primo*, *ultimo*, *mezzo*)

47

# Ordinamento per fusione

Merge ($A$, *primo*, *ultimo*, *mezzo*)
//Pre: $primo \leq mezzo \leq ultimo <$ dimensione di $A$
    $A[primo..mezzo]$, $A[mezzo + 1..ultimo]$ ordinati
//Post: $A$ $[primo..ultimo]$ è ordinato
$i \leftarrow primo$, $j \leftarrow mezzo + 1$, $k \leftarrow 0$
**while** $i \leq mezzo$ **and** $j \leq ultimo$ **do**
    **if** $A[i] \leq A[j]$ **then** $B[k] \leftarrow A[i]$, $i \leftarrow i + 1$
        **else** $B[k] \leftarrow A[j]$, $j \leftarrow j + 1$
    $k \leftarrow k + 1$
**if** $i \leq mezzo$ **then** // $j > ultimo$, dunque $A[j..ultimo] = \varnothing$
    $B[k..\ ultimo - primo] \leftarrow A[i..mezzo]$
**else** // $i > mezzo$, dunque $A[i..mezzo] = \varnothing$
    $B[k..\ ultimo - primo] \leftarrow A[j..ultimo]$
$A[primo..ultimo] \leftarrow B[0..ultimo - primo]$

inv. $A[i..mezzo]$, $A[j..ultimo]$ ordinati, $B[1..k-1]$ ordinato ed i suoi el. sono quelli di $A[primo..i-1]$ e $A[mezzo+1..j-1]$

48

## Complessità di Merge-Sort

$$T(n) = 2T(n/2) + T_{\text{Merge}}(n)$$

E' facile vedere che $T_{\text{Merge}}(n) \in \Theta(n)$ e dunque:

$$T(n) = \begin{cases} c & \text{se } n \leq 1 \\ 2T(n/2) + n & \text{se } n > 1 \end{cases}$$

49

## Complessità di Merge-Sort

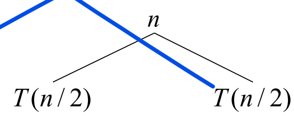
$T(n) = 2T(n/2) + n$

$T(n)$

50

## Complessità di Merge-Sort

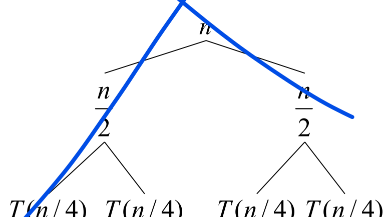
$$T(n) = 2T(n/2) + n$$




51

## Complessità di Merge-Sort

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

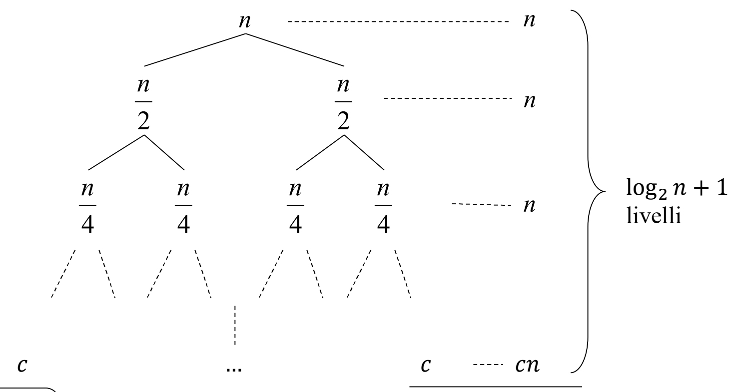



52

# Complessità di Merge-Sort

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

Albero di ricorsione




$$\log_2 n \cdot n + cn \in \Theta(n \log n)$$

Si può fare meglio di Merg-Sort?

53

# Relazioni lineari a partizione bilanciata

Lineare

Lavoro polinomiale di divisione/combinazione

$$T(n) = \begin{cases} d & \text{se } n = 1 \\ aT(n/b) + cn^{\beta} & \text{se } n > 1 \end{cases}$$

Coefficiente costante

Partizioni bilanciate

54

## Relazioni lineari a partizione bilanciata

Teorema. Se $a \geq 1; b \geq 2; c > 0; d, \beta \geq 0$, posto $\alpha = \log a / \log b$ allora :

$$\begin{cases} T(n) \in O(n^{\alpha}) & \text{se } \alpha > \beta \\ T(n) \in O(n^{\alpha} \log n) & \text{se } \alpha = \beta \\ T(n) \in O(n^{\beta}) & \text{se } \alpha < \beta \end{cases}$$

$$T(n) = \begin{cases} d & \text{se } n = 1 \\ aT(n/b) + cn^{\beta} & \text{se } n > 1 \end{cases}$$

Si intravedono applicazioni?

55

## Relazioni lineari a partizione bilanciata

Teorema. Se $a \geq 1; b \geq 2; c > 0; d, \beta \geq 0$, posto $\alpha = \log a / \log b$ allora :

$$\begin{cases} T(n) \in O(n^{\alpha}) & \text{se } \alpha > \beta \\ T(n) \in O(n^{\alpha} \log n) & \text{se } \alpha = \beta \\ T(n) \in O(n^{\beta}) & \text{se } \alpha < \beta \end{cases}$$

$$T(n) = \begin{cases} d & \text{se } n = 1 \\ aT(n/b) + cn^{\beta} & \text{se } n > 1 \end{cases}$$

Ricerca binaria

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

$$a = 1, b = 2, \alpha = \log 1 / \log 2 = 0 = \beta$$

$$T(n) \in O(n^{\alpha} \log n)$$
$$= O(\log n)$$

56

## Relazioni lineari a partizione bilanciata

Teorema. Se $a \geq 1; b \geq 2; c > 0; d, \beta \geq 0$, posto $\alpha = \log a / \log b$ allora:

$$\begin{cases} T(n) \in O(n^{\alpha}) & \text{se } \alpha > \beta \\ T(n) \in O(n^{\alpha} \log n) & \text{se } \alpha = \beta \\ T(n) \in O(n^{\beta}) & \text{se } \alpha < \beta \end{cases}$$

$$T(n) = \begin{cases} d & \text{se } n = 1 \\ aT(n/b) + cn^{\beta} & \text{se } n > 1 \end{cases}$$

Merge Sort

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

$$a = b = 2, \alpha = \log 2 / \log 2 = 1 = \beta$$

$$T(n) \in O(n^{\alpha} \log n) = O(n \log n)$$

57

## Quick Sort

```
QUICK-SORT(A)
if length(A) > 1 then
    p ← PARTITION(A)
    QUICK-SORT(A[1..p−1])
    QUICK-SORT(A[p+1..length(A)])
end if
```

58

# Quick Sort, caso medio

$$T(n) = \begin{cases} a & \text{se } n \leq 1 \\ \frac{1}{n}\sum_{k=1}^{n}(T(k-1)+T(n-k)) + bn + c & \text{altrimenti} \end{cases}$$

$$T(n) = \frac{2}{n}\sum_{i=0}^{n-1} T(i) + bn + c$$
$$nT(n) = 2\sum_{i=0}^{n-1} T(i) + bn^2 + cn$$

sostituendo $n$ con $n-1$:

$$(n-1)T(n-1) = 2\sum_{i=0}^{n-2} T(i) + b(n-1)^2 + c(n-1)$$

59

# Quick Sort, caso medio

$$nT(n) = 2\sum_{i=0}^{n-1} T(i) + bn^2 + cn$$

Sottraendo

$$(n-1)T(n-1) = 2\sum_{i=0}^{n-2} T(i) + b(n-1)^2 + c(n-1)$$

si ottiene

$$nT(n) - (n-1)T(n-1) = \left(2\sum_{i=0}^{n-1} T(i) + bn^2 + cn\right) - \left(2\sum_{i=0}^{n-2} T(i) + b(n-1)^2 + c(n-1)\right)$$
$$nT(n) - (n-1)T(n-1) = 2T(n-1) + c + b(2n-1)$$
$$nT(n) = (n+1)T(n-1) + c + b(2n-1)$$

60

# Quick Sort, caso medio

dividendo a $n(n+1)$

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{c + b(2n-1)}{n(n+1)}$$

Introducendo $D(n) = T(n)/(n+1)$ abbiamo

$$D(n) = D(n-1) + \frac{c + b(2n-1)}{n(n+1)}$$

dove $\frac{c+b(2n-1)}{n(n+1)} \in O(1/n)$. Di conseguenza

$$D(n) \in O\left(\sum_{i=1}^{n} \frac{1}{i}\right) \in O(\log n)$$

e quindi

$$T(n) \in O(n \log n)$$

61