Algorithmen & Komplexität

Angelika Steger
Institut für Theoretische Informatik

steger@inf.ethz.ch

Weitere Beispiele Effizienter Algorithmen

Anzahl Vergleiche bei Binärer Suche

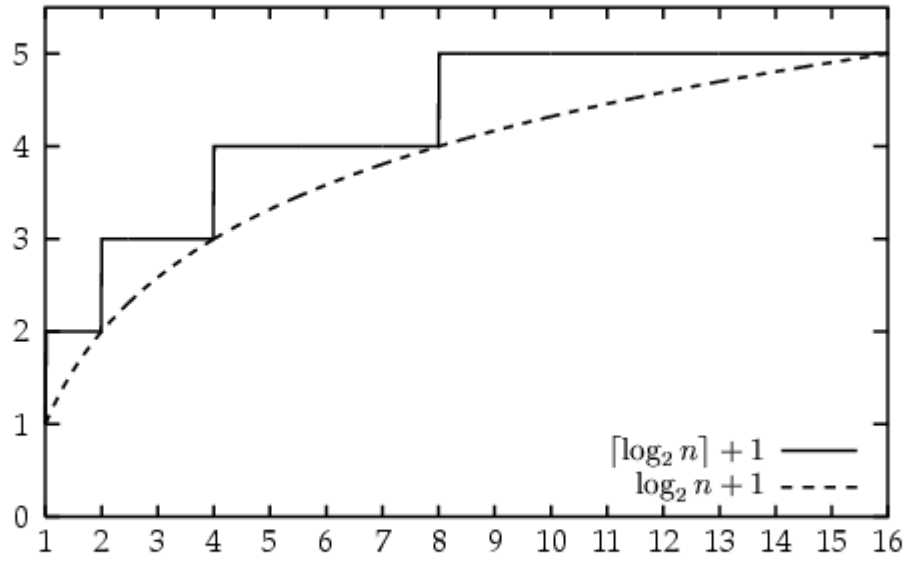


Abbildung 2.1: Anzahl Vergleiche $B_n = \lceil \log_2 n \rceil + 1$ bei einer binären Suche.

Für asymptotische Analyse:

Laufzeit von Binärer Suche ist O(log n)

MergeSort – Analyse (3)

Satz:

Um ein Feld der Grösse n zu sortieren genügen $2n \cdot (\log_2 n + 1)$ Vergleiche bzw. Laufzeit $O(n \cdot \log_2 n) = O(n \log(n))$.

Bemerkung:

Genauere Analyse zeigt, dass sogar

$$C_n \leq n \cdot \lfloor \log_2 n \rfloor + 2n$$

gilt.

QuickSort – Analyse (2)

Satz:

Um ein Feld der Grösse n zu sortieren benötigt QuickSort im schlimmsten Fall $\frac{1}{2}\, n(n-1)$ Vergleiche bzw. Laufzeit $\Theta(n^2)$.

Bemerkung:

- Trotzdem gilt QuickSort als ein (in der Praxis und leicht abgewandelt) als sehr effizientes Sortierverfahren.
- Mathematisch exakt kann man dies begründen indem man die erwartete Laufzeit betrachtet, wobei der Erwartungswert über alle n! Reihenfolgen der Zahlen 1, …,n gebildet wird.

  Man erhält: erwartete Laufzeit = $O(n \log n)$

Sortieren – Untere Schranken

Satz:

Jeder Algorithmus, der n Elemente mit Hilfe von „if“-Abfragen sortiert, benötigt mindestens $\Theta(n \log n)$ Vergleiche.

Beweisidee:

Gegenspieler-Beweis …

BucketSort

Algorithmus 2.3 BucketSort

Eingabe: Feld $a[1..n]$ mit Elementen aus $\mathcal{U} = \{0, 1, \ldots, N-1\}$.
Ausgabe: Feld $a[1..n]$, wobei die Einträge aufsteigend sortiert sind.

```
for i = 0 to N − 1 do b[i] = 0;
for i = 1 to n do b[a[i]] = b[a[i]] + 1;
k := 0;
for i = 0 to N − 1 do
    for j = 1 to b[i] do
        k:= k+1; a[k] = i;
```

Satz:

Eine Folge aus n Zahlen aus dem Bereich U = {0,1,…,N-1} lässt sich mit BucketSort in Zeit O(n+N) sortieren.

BucketSort - Verbesserungen

Satz:

Eine Folge aus n Zahlen aus dem Bereich $U = \{0,1,\ldots,N^k-1\}$ lässt sich mit BucketSort in Zeit $O(k \cdot (n+N))$ sortieren.

2.3 Medianbestimmung

Selektionsproblem:

Gegeben: Menge $S=\{a_1,\ldots,a_n\}$; $k \in N$

Aufgabe: Gebe das k-te kleinste Element aus.

Annahme: a_i paarweise verschieden

(allg. Fall geht ähnlich, aber in der Notation aufwendiger …)

Bemerkung:

Medianbestimmung: $k = \lceil n/2 \rceil$

1. Ansatz

- Sortiere die a_i`s.
- Gebe das k-te Element aus.

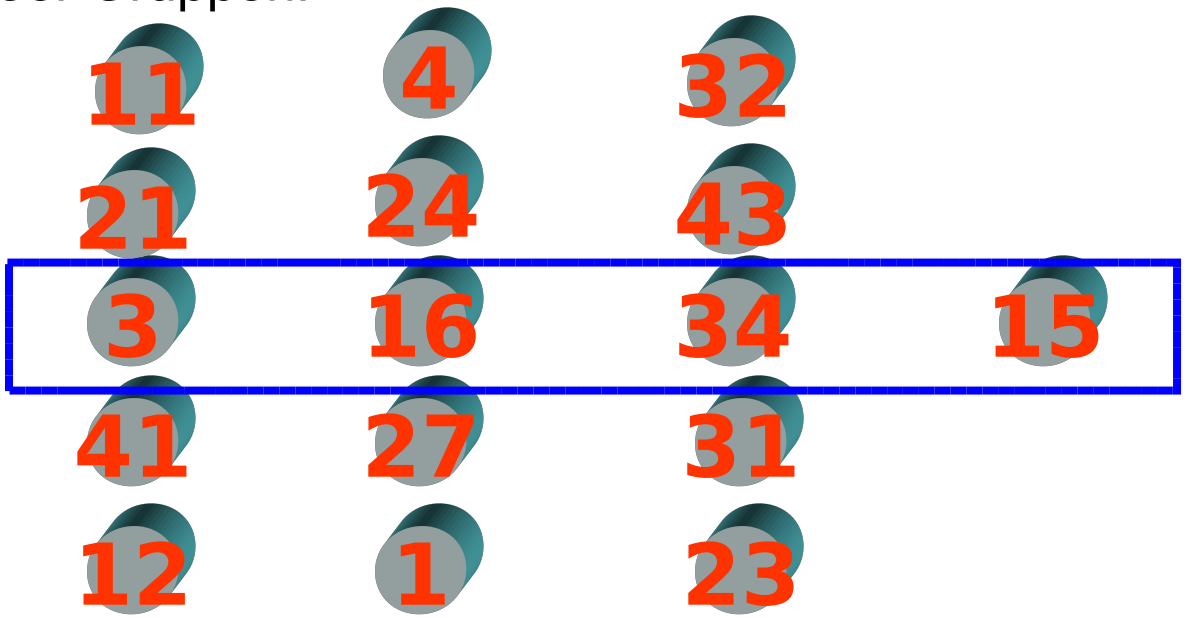
Korrektheit: √

Laufzeit: O(n log n) √

Ziel: O(n)

Select - Beispiel

5er Gruppen:



Stelle Median jeder Gruppe in die Mitte ….

Bestimme Median … rekursiv

k=2

Partitioniere Elemente: ‹ < 16 〉, 16 , ‹ > 16 〉

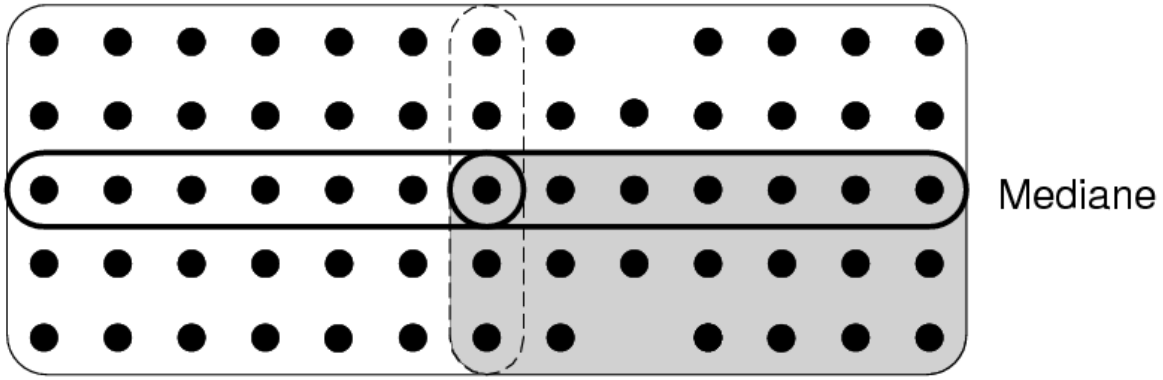
Select

Algorithmus 2.4 SELECT

Eingabe: Menge $S = \{a_1, \ldots, a_n\}$, Index k mit $1 \leq k \leq n$.
Ausgabe: Das k-te kleinste Element in S.

func SELECT(S, k)
if $(n \leq 15)$ **then**
    Sortiere die Menge S mit MERGESORT und gebe das k-te kleinste Element aus;
else
    Teile Elemente aus S in $\lceil \frac{n}{5} \rceil$ Gruppen auf, $\lfloor \frac{n}{5} \rfloor$ davon mit jeweils genau 5 Elementen;
    Bestimme den Median jeder Gruppe;
    Bestimme rekursiv den Median dieser Mediane, nenne ihn p;
    Partitioniere $S \setminus \{p\}$:
        $S_1 := \{s \in S \mid s < p\}$,
        $S_2 := \{s \in S \mid s > p\}$,
    if $|S_1| = k - 1$ **then**
        return p
    elseif $k \leq |S_1|$ **then**
        SELECT(S_1, k)
    else
        SELECT$(S_2, k - 1 - |S_1|)$;

Anschaulich



- 5-er Gruppen und ihre Mediane (im Bild aufsteigend angeordnet)
- In der Mitte: Der Median der Mediane (rekursiv bestimmt)
- Bestimme den Rang m des Median der Mediane
- Vergleiche diesen mit dem Rang k des gesuchten Elements
- Dann: mind. 3/10 aller Elemente können ausgeschlossen werden
- Warum? Angenommen $m \geq k$. Dann haben die Elemente im grau schraffierten Bereich ebenfalls Rang grösser k.
- Korrektheit: Aus Definition des Algorithmus klar… ☺

Laufzeit

Satz

Select benötigt für die Bestimmung des k-kleinsten Elements einer n-elementigen Menge (n ≥ 3), höchstens 22(n – 2) Vergleiche.

Beweis

S_n := max. Anzahl Vergleiche bei einer n-elementigen Menge.

$$S_n \leq 6 \cdot \lceil \frac{n}{5} \rceil + S_{\lceil \frac{n}{5} \rceil} + n - 1 + S_{\lceil \frac{7n}{10} + 2 \rceil}$$

- Aus 5er-Gruppen den Median bestimmen: 6 Vergleiche pro Gruppe
- Algorithmus rekursiv zur Bestimmung des Median der Mediane aufrufen
- Partitionierung der Elemente nach dem Median der Mediane
- Algorithmus rekursiv auf Menge mit $\lfloor \frac{7n}{10} + 2 \rfloor$ Elementen anwenden

2.4 Schnelle Matrixmultiplikation

Problem: Matrixmultiplikation

$$C = A \cdot B$$

„Zeile mal Spalte“: Definition der Matrixmultiplikation

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

„Zeile mal Spalte“: $\Theta(n^3)$ Operationen (Addition oder Multiplikation)

n Multiplikationen, n – 1 Additionen pro Zeile/Spalte Paar.

Es gibt n^2 Paare.

Algorithmus von Strassen

Frage: Berechnet der Algorithmus „Zeile mal Spalte“ das Produkt zweier Matrizen mit der minimalen Anzahl Operationen?

(Überraschende) Antwort: Nein!

Algorithmus von Strassen: $O(n^{2.81})$ Operationen

Algorithmus von Strassen

Spezialfall: A und B sind 2x2 Matrizen

Wir berechnen das Produkt auf spezielle Weise

$$C = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix} = \begin{pmatrix} m_2 + m_3 & m_1 + m_2 + m_5 + m_6 \\ m_1 + m_2 + m_4 - m_7 & m_1 + m_2 + m_4 + m_5 \end{pmatrix}$$

8 Multiplikationen

7 Multiplikationen

$$\begin{aligned} m_1 &= (a_{21} + a_{22} - a_{11})(b_{22} - b_{12} + b_{11}) \\ m_2 &= a_{11}b_{11} \\ m_3 &= a_{12}b_{21} \\ m_4 &= (a_{11} - a_{21})(b_{22} - b_{12}) \\ m_5 &= (a_{21} - a_{22})(b_{12} - b_{11}) \\ m_6 &= (a_{12} - a_{21} + a_{11} - a_{22})b_{22} \\ m_7 &= a_{22}(b_{11} + b_{22} - b_{12} - b_{21}) \end{aligned}$$

Korrektheit: einfach nachrechnen … ☺

Anzahl Operationen: 7 Multiplikationen und 21 Additionen (bzw 15 Additionen, bei geschickter Speicherung von Zwischenergebnissen)

Laufzeit

f(n) := Anzahl der elementaren Operationen (Additionen und Multiplikationen) die für die Multiplikation zweier nxn Matrizen benötigt werden.

Für eine $2^k x 2^k$ Matrix sind

15 $2^{k-1} x 2^{k-1}$ Matrixadditionen und

7 $2^{k-1} x 2^{k-1}$ Matrixmultiplikationen nötig

$$f(n) = f(2^k) \leq 7 \cdot f(2^{k-1}) + 15 \cdot 2^{2(k-1)}$$

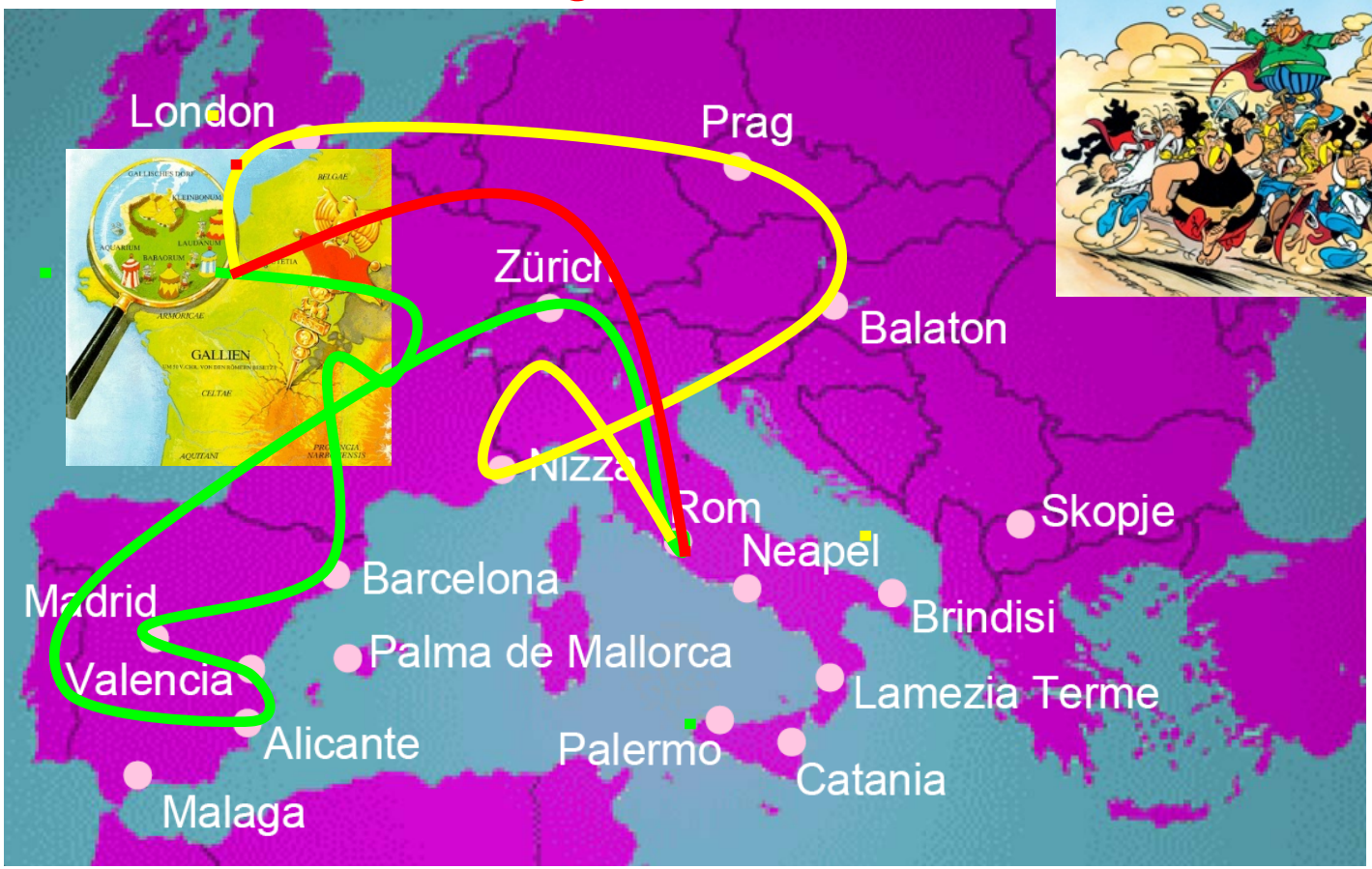
Satz

Das Produkt zweier nxn Matrizen kann in Zeit $O(n^{2.81})$ berechnet werden.

Coppersmith & Winograd: $O(n^{2.3727})$ bester bekannter Exponent (aber Algorithmus eher von theoretischem Interesse)

2.5 Kürzeste Pfade in Graphen

Alle Wege führen nach Rom!



Aber welcher ist der kürzeste?