Algorithmus der Woche
Binäre Suche

Prof. Dr. Thomas Seidl
Dipl.-Inform. Jost Enderle
RWTH Aachen, Lehrstuhl für Informatik 9

Problem

- Gegeben: sortierte Liste von Einträgen
  - z. B. CD- oder Bücher-Regal, Telefonbuch, Lexikon
  - Sortierung nach bestimmtem Schlüssel (z. B. alphabetisch nach Name)

- Frage: Wie finde ich möglichst schnell einen Eintrag mit einem bestimmten Schlüsselwert?
  - z. B. Suche nach Eintrag mit Namen „Nelly“

Erster Ansatz

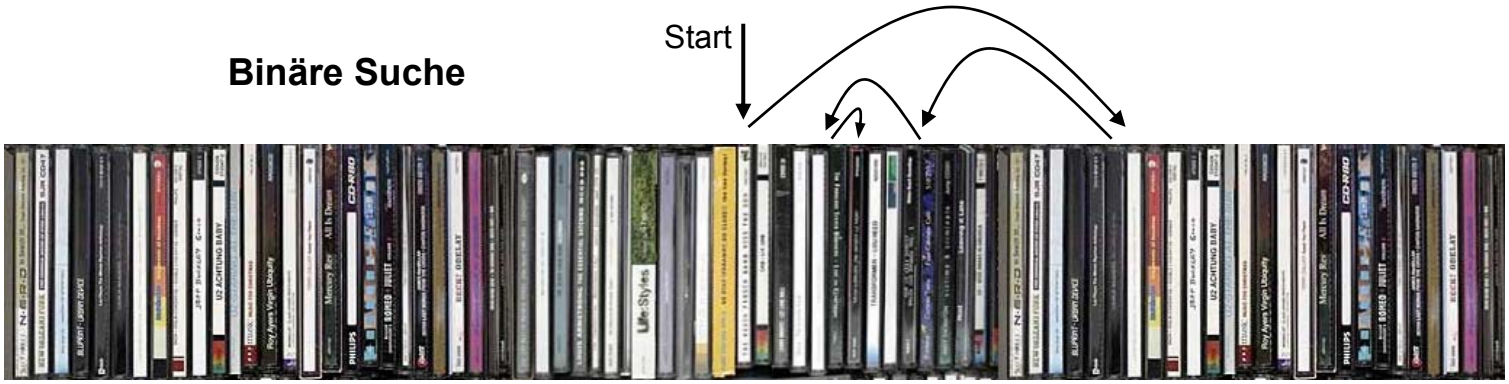
- Sequenzielle (lineare) Suche
  - Beginn der Suche ganz am Anfang der Liste
  - Jeder Eintrag wird angeschaut, bis gesuchter Schlüssel gefunden

Analyse des ersten Ansatzes

- Nachteile der sequenziellen Suche
  - Sortierung wird nicht ausgenutzt
  - Je größer der Schlüssel, desto mehr Einträge müssen durchsucht werden
    - Im Durchschnitt: Hälfte der Einträge muss durchsucht werden
    - Im schlimmsten Fall: alle Einträge müssen durchsucht werden (z. B. wenn gesuchter Eintrag nicht vorhanden ist)
  - Bei doppelt so vielen Einträgen muss man doppelt so lange suchen
    - Allgemein: Bei n-mal so vielen Einträgen muss man n-mal so lange suchen
    - D. h. der Suchaufwand ist proportional zur Anzahl der Einträge

Zweiter Ansatz

- Binäre Suche
  - Beginn der Suche in der Mitte der Liste
  - Fallunterscheidung für betrachteten Eintrag:
    - Falls Schlüssel kleiner als Suchschlüssel: rechts weitersuchen
    - Sonst: in linker Hälfte der Liste weitersuchen
  - Für linke oder rechte Hälfte wieder mittleren Eintrag betrachten
    - Halbierung fortsetzen, bis Suchschlüssel gefunden



Algorithmus

BinaereSuche(A, Schluessel, links, rechts)

```
1  while links ≤ rechts do
2        mitte := (links + rechts)/2     {Ergebnis runden}
3        if A[mitte] = Schluessel then return mitte
4        if A[mitte] > Schluessel then rechts := mitte – 1
5        if A[mitte] < Schluessel then links := mitte + 1
6  endwhile
7  return „nicht gefunden“
```

Ausgabe: Position von „Schluessel“ in Liste A zwischen „links“ und „rechts“

Beispiel

- Suche nach „Nelly“ in CD-Regal mit 500 Einträgen
  (Regal[n] kennzeichnet die n-te CD im Regal)
  1. BinaereSuche(Regal, „Nelly“, 1, 500)
     - Regal[250] = „Kelly Family“ $\Rightarrow$ rechts weitersuchen
  2. links = 251, rechts = 500
     - Regal[375] = „Rachmaninov“ $\Rightarrow$ links weitersuchen
  3. links = 251, rechts = 374
     - Regal[312] = „Lionel Hampton“ $\Rightarrow$ rechts weitersuchen
  4. links = 313, rechts = 374
     - Regal[343] = „Nancy Sinatra“ $\Rightarrow$ rechts weitersuchen
  5. links = 344, rechts = 374
     - Regal[359] = „Nelly“ $\Rightarrow$ gefunden!

Analyse (1)

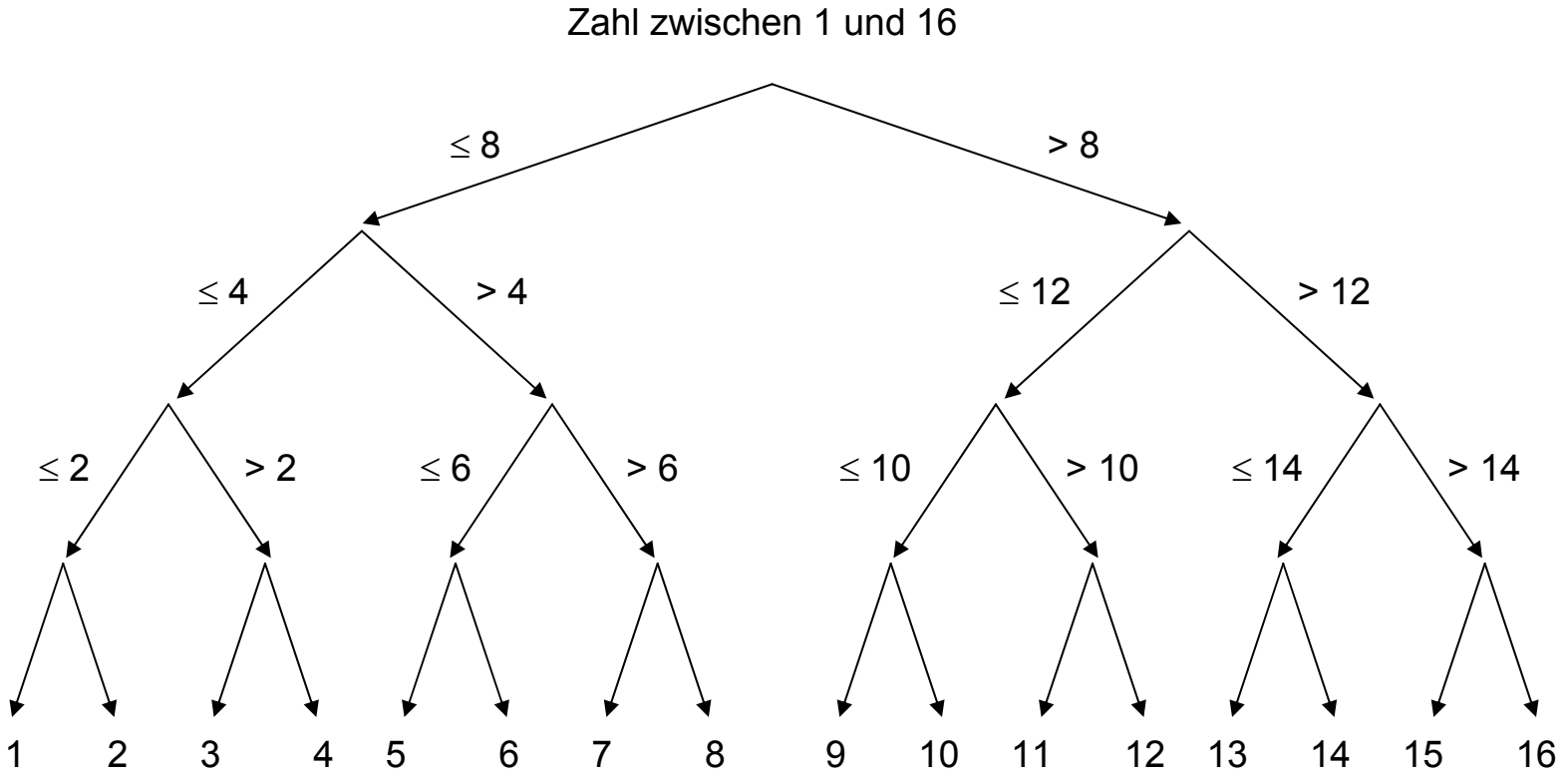
- Wie lange muss man höchstens suchen?
  - Gegeben: Liste mit N Einträgen
  - Gesucht: Anzahl der höchstens benötigten Suchschritte
- Anders herum:
  - Gegeben: k Suchschritte (Vergleiche)
  - Gesucht: Anzahl der Einträge, die durchsucht werden können
    - mit 1 Vergleich: 2 Einträge
    - mit 2 Vergleichen: 4 Einträge
    - mit 3 Vergleichen: 8 Einträge
    - ...
    - mit k Vergleichen: $\underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{\text{k mal}} = 2^k$ Einträge

Analyse (2)

- Nochmal: Wie viele Vergleiche für N Einträge?
  - Mit k Vergleichen lassen sich $N = 2^k$ Einträge dursuchen
  - Wie lässt sich k aus N bestimmen?
    - Umkehrfunktion zur Zweier-Potenz: Zweier-Logarithmus $\log_2$
    - Damit: $k = \log_2 N$
  - Beispiel:
    - Wie viele Vergleiche k für N = 10.000 Einträge?
    - $k = \log_2 10.000 \approx 13,29 \Rightarrow$ 14 Vergleiche
  - Ratespiel:
    - Ich denke mir eine Zahl zwischen 1 und 1.000.
    - Finde mit 10 Ja/Nein-Fragen die Zahl heraus!
    - Klappt mit binärer Suche: $\log_2 1.000 \approx 9,97 \Rightarrow$ 10 Vergleiche

Suchbaum

Darstellung der binären Suche durch einen Suchbaum



Zusammenfassung

- Problem: Suche eines Eintrags in einer sortierten Liste der Länge N
- Sequenzielle (lineare) Suche:
  - Maximale Anzahl der Vergleiche: N
  - Durschnittliche Anzahl der Vergleiche: N/2
  - funtioniert auch bei unsortierten Listen
- Binäre Suche
  - Maximale und durchschnittliche Anzahl Vergleiche: ca. $\log_2 N$
  - Weitere Informationen:
    - http://www-i9.informatik.rwth-aachen.de/algwoche/binsuche