Kap. 4.1.2: Binäre Suche ff

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

Fakultät für Informatik, TU Dortmund

VO nach 1. Übungstest

10. VO DAP2 SS 2009 19. Mai 2009

BinarySearch (nicht-rekursiv)

```
Procedure BinarySearch(A,s,l,r)
var Index m
while l ≤ r do {
    m:= ⌊(l+r)/2⌋          // Mitte bestimmen
    if A[m].key==s then return m
    if A[m].key>s then r:=m-1
    else                   l:=m+1   // A[m].key<s
}
return 0
```

Aufruf: BinarySearch(A,s,1,n)

Skript-Variante: BinarySearch (nicht-rek.)

```
Procedure BinarySearch(A,s,l,r)
(1)  var Index m
(2)  repeat
(3)      m:= ⌊(l+r)/2⌋
(4)      if s < A[m].key then r:=m-1
(5)      else                 l:=m+1
(6)  until s == A[m].key or l > r
(7)  if s==A[m].key then return m
(8)  else return 0
```

Aufruf: BinarySearch(A,s,1,n)

Analyse von BinarySearch

Annahmen:

- Skript-Variante: nicht-rekursiv
- die Daten sind schon sortiert
- wir zählen nur die Vergleiche in Zeile (4)
- Annahme: $n=2^k$-1 für geeignetes k

Best Case: $C_{best}(n)$=1=$\Theta(1)$

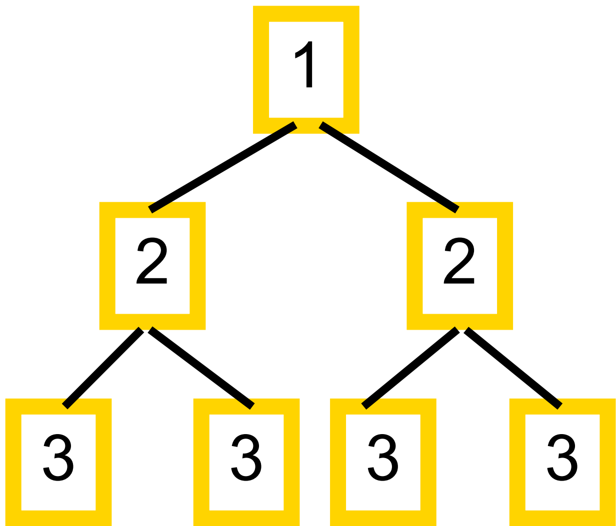
Worst Case: $C_{worst}(n)=?$

$$2^k - 1 = \sum_{i=1}^{k} 2^{i-1} = n$$

für erfolgreiche
und erfolglose Suche

Worst Case: $C_{worst}(n) = \log(n+1) = \Theta(\log n)$

Position:	1	2	3	4	5	6	7
Anzahl Vergleiche:	3	2	3	1	3	2	3



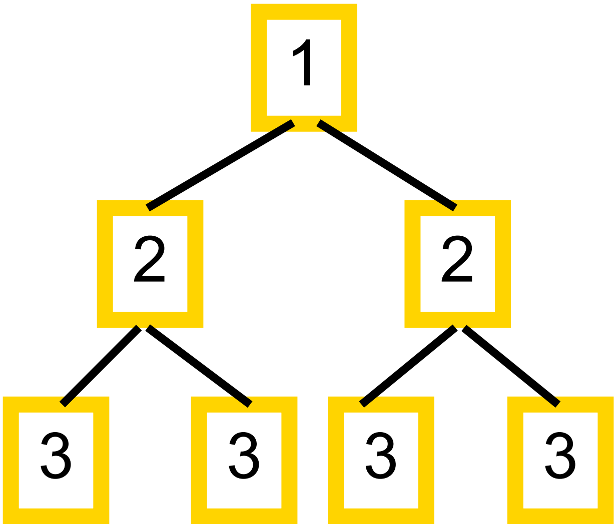
Anzahl Schritte	Anzahl Positionen	Summe
1	$1=2^0$	1
2	$2=2^1$	3
3	$4=2^2$	7
k	2^{k-1}	$\sum 2^{i-1}$

Average Case Analyse von BinarySearch

Durchschnittliche Zeit für erfolgreiche Suche:

$$C_{avg}(n) = \frac{1}{n}\sum_{i=1}^{k} i2^{i-1} = \ldots =$$

$$= \frac{n+1}{n}\log(n+1) - 1 = \theta(\log n)$$



Anzahl Schritte	Anzahl Positionen	Produkt	Summe
1	$1=2^0$	1	1
2	$2=2^1$	4	5
3	$4=2^2$	12	17
k	2^{k-1}	$k2^{k-1}$	$\sum i2^{i-1}$

Binäre Suche / Diskussion

- Binäre Suche ist für große n sehr empfehlenswert!
- z.B. $\log 1000 = 10$; $\log 10^6 \approx 20$
- Binäre Suche ist nur sinnvoll, wenn sich die Daten nicht allzu oft ändern

- In der Praxis oft Interpolationssuche: wähle m als erwartete Position des Suchschlüssels
- hier ist $C_{avg}(n)=\Theta(\log \log n)$
- aber $C_{worst}(n)=\Theta(n)$, z.B. „AAAA…AAABZ"

z.B. $\log \log 10^6 \approx 5$

Ein kleines Rätsel

- Wie oft muss man ein Blatt Papier (0,1 mm dick) falten, bis es eine Dicke erreicht, die der Entfernung von Erde zu Mond entspricht?

  (363.258 km = 363.258.000.000 mm)

- **Lösung:** 42 Mal, denn:

  $0{,}1 \times 2^k \geq 363{,}258 \times 10^9$

  $\Rightarrow k \geq \log (363{,}258 \text{ x } 10^{10}) \approx 41{,}7$

Exkurs: Binäre Suche bei InsertionSort

```
for k:=2,…,n {
   s:=A[k].key
   i:=k
   while i>1 and A[i−1].key>s {
      A[i].key:=A[i−1].key
      i:=i−1
   }
   A[i].key:=s
}
```

Ersetze die Suche nach der richtigen Position für A[k] durch BinarySearch(A,s,1,k-1)

jedoch: return l statt 0, falls s nicht gefunden wird

bisheriges Insertion-Sort

- **Anzahl der Schlüsselvergleiche:**

  $C_{best}(n)=\Theta(n)$ und $C_{avg}(n)=C_{worst}(n)=\Theta(n^2)$

- **Anzahl der Datenbewegungen:**

  $M_{best}(n)=\Theta(n)$ und $M_{avg}(n)=M_{worst}(n)=\Theta(n^2)$

BinarySearch Insertion-Sort

- **Anzahl der Schlüsselvergleiche:**

  $C_{best}(n)=\Theta(n)$ und $C_{worst}(n)=\Theta(n \log n)$

$$
\begin{aligned}
C_{worst}(n) &= \sum_{i=1}^{n-1} \lceil \log(i+1) \rceil = \sum_{i=2}^{n} \lceil \log i \rceil \\
&\leq \sum_{i=2}^{n} (\log i + 1) = \log(n!) + n - 1 \\
\text{Stirling-Formel} &\approx n \log n - n \log e + O(\log n) \\
&\approx n \log n - 1,4427n
\end{aligned}
$$

BinarySearch Insertion-Sort

- **Anzahl der Datenbewegungen:**

  $M_{best}(n)=\Theta(n)$ und $M_{avg}(n)=M_{worst}(n)=\Theta(n^2)$

- **Eigenschaften:**
  - in situ?
  - adaptiv?
  - stabil?

4.1.3 Geometrische Suche / Idee

Annahme: die Liste ist bereits sortiert und sei 2^k die größte Zweierpotenz mit $2^k \leq n$

Idee: Vergleiche die Daten an den Positionen $2^0, 2^1, 2^2, \ldots, 2^k$ so lange mit dem Suchschlüssel s

- bis s gefunden wurde (dann STOP) oder
- ein Datum $A[2^m].key > s$ gefunden wurde für ein m

In diesem Fall: Starte BinarySearch auf den Plätzen $2^{m-1}+1, \ldots, 2^m-1$

Falls $A[2^k] < s$, dann BinarySearch auf $2^k+1, \ldots, n$

Analyse von Geometrischer Suche

Best Case: $C_{best}(n) = \Theta(1)$

Worst Case: $C_{worst}(n) = \Theta(\log n)$

- in Startphase $k+1 \leq \log n + 1$ Vergleiche
- für BinarySearch maximal $\log (n+1)$ Vergleiche
- zusammen sind dies höchstens $2\lceil \log (n+1) \rceil$ Vergleiche
- insgesamt sind dies höchstens doppelt so viele Vergleiche wie BinarySearch

Geometrischer Suche / Diskussion

- Geometrische Suche ist für Daten mit $s < A[2^m].key$, für m klein ($m \leq n^{1/2}$), sehr effizient.
- Dann ist die Anzahl der Vergleiche durch $2m+1$ nach oben beschränkt.

- Geometrische Suche auch gut geeignet, wenn der Suchbereich unbekannt ist: suche einfach in $2^0, 2^1, 2^2, \ldots$ bis s gefunden wurde oder ein Element mit $A[2^m].key > s$.
- Dann BinarySearch auf $2^{m-1}+1, \ldots, 2^m-1$