





Grundzüge der Informatik 1

Vorlesung 8 - flipped classroom

Teile & Herrsche Algorithmen

Wodurch unterscheiden sich Teile & Herrsche Algorithmen?

- Die Anzahl der Teilprobleme
- Die Größe der Teilprobleme
- Den Algorithmus für das Zusammensetzen der Teilprobleme
- Den Rekursionsabbruch

Wann lohnt sich Teile & Herrsche?

Kann durch Laufzeitanalyse vorhergesagt werden



- Rekursionen

Laufzeiten der Form

$$T(n) = a T(n/b) + f(n)$$

$$(und T(1) = O(1))$$



- Rekursionen

Laufzeiten der Form

$$T(n) = aT(n/b) + f(n)$$

$$(und T(1) = O(1))$$



- Rekursionen

Laufzeiten der Form

$$T(n) = a T(n/b) + f(n)$$

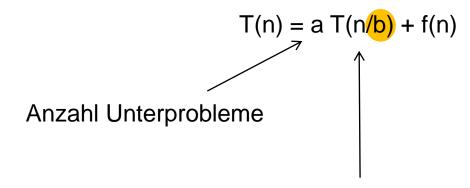
Anzahl Unterprobleme

$$(und T(1) = O(1))$$



Laufzeitanalyse - Rekursionen

Laufzeiten der Form

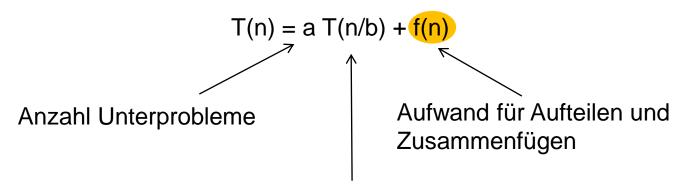


(und T(1) = O(1)) Größe der Unterprobleme (bestimmt Höhe des Rekursionsbaums)



Laufzeitanalyse - Rekursionen

Laufzeiten der Form



(und T(1) = O(1)) Größe der Unterprobleme (bestimmt Höhe des Rekursionsbaums)



- Rekursionen

- Betrachten Sie folgende Laufzeitrekursion
- $T(n) = T(n/2) + n^2$
- T(1) = 1
- Finden Sie eine Lösung für diese Rekursion
- Zeigen Sie die Korrektheit Ihrer Lösung. Sie können annehmen, dass n eine Zweierpotenz ist.



- Rekursionen

Behauptung

• $T(n) = O(n^2)$

Beweis

- Sei n eine Zweierpotenz. Wir zeigen T(n) ≤ 2n²
- Induktionsanfang (n=1)
- $T(n) = 1 \le 2 = 2n^2$
- Induktionsannahme:
- Für 1 ≤ m< n, m Zweierpotenz gilt: T(m) ≤ 2m²</p>



- Rekursionen

Behauptung

• $T(n) = O(n^2)$

Beweis

- Induktionsschluss:
- Sei n>1 eine Zweierpotenz
- Es gilt
- T(n) = T(n/2) + $n^2 \le 2(n/2)^2 + n^2 = 3/2 \ n^2 \le 2 \ n^2$



- Der h-Index ist die größte Anzahl h von Publikationen eines Wissenschaftlers, die jeweils mindestens h-mal zitiert werden
- Seinen nun die Anzahl der Zitierungen pro Publikation in einem sortierten Feld gegeben (von viel zu wenig). Wie kann man effizient den h-Index bestimmen?
- Was ist die Laufzeit Ihres Algorithmus?



hIndex(A,p,r) /* alle Einträge in A größer als 0

- if p=r then return r
- 2. $m = \lceil (p + r) / 2 \rceil$
- 3. **if** A[m] < m **then return** hIndex(A,p,m-1)
- 4. **else return** hIndex(A,m,r)



hIndex(A,p,r) /* alle Einträge in A größer als 0

- 1. if p=r then return r
- 2. $m = \lceil (p + r) / 2 \rceil$
- 3. if A[m] < m then return hIndex(A,p,m-1)</p>
- 4. **else return** hIndex(A,m,r)

Laufzeit

- T(n) = T(n/2) + c
- T(1) = c
- Wie binäre Suche: O(log n)



- Eine Inversion in einem Feld ist ein Indexpaar (i,j), i<j, mit A[i]>A[j]
- Was ist das Feld mit der größten Anzahl an Inversionen und was ist deren Anzahl?
- Entwerfen Sie einen iterativen Algorithmus zur Berechnung der Anzahl Inversionen
- Was ist die Laufzeit Ihres Algorithmus?



- Eine Inversion in einem Feld ist ein Indexpaar (i,j), i<j, mit A[i]>A[j]
- Was ist das Feld mit der größten Anzahl an Inversionen und was ist deren Anzahl?
- Entwerfen Sie einen iterativen Algorithmus zur Berechnung der Anzahl Inversionen
- Was ist die Laufzeit Ihres Algorithmus?
- Eine absteigend sortierte Folge hat n (n-1)/2 Inversionen



Inversionen(A,n)

- 1. C = 0
- 2. **for** i=1 **to** n **do**
- 3. for j=i+1 to n do
- 4. **if** A[i] > A[j] **then** C = C + 1
- 5. return C

Laufzeit

• O(n²)

