8. Übung AuD

Dominic Deckert

6. Januar 2017

Previously on ...

- strukturierte Datentypen
- Listen & Bäume
- Wahrheitswerte und Operatoren

Aufgabe

a)
gegeben: Liste l
gesucht: $f(l) = 1$, wenn Differenz benachbarter Listenelemente ≤ 1 (iterativ)
b)
gegeben: Baum-Pointer p
gesucht: Am Baum außerhalb des Speichers alle Blätter löschen

Sortieren

Warum Sortieren?

Sortieren

Warum Sortieren?
Beschleunigtes Suchen auf großen Datenmengen.

Quicksort

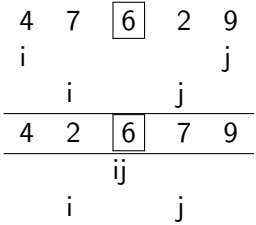
Sortier-Algorithmus für Listen
Basiert auf “Divide-and-Conquer”-Prinzip

Quicksort

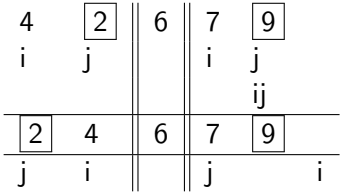
Sortier-Algorithmus für Listen
Basiert auf “Divide-and-Conquer”-Prinzip
Idee: Teile die Liste in zwei kleinere Teile und ordne diese rekursiv
Algorithmus: siehe OHP

2)

2)



2)



Hinweis: Das rechte i begibt sich über die Grenze des (lokalen) Quicksort-Aufrufes hinaus, er bricht dort also den Sortierschritt ab

2)

$2 \parallel 4 \parallel 6 \parallel 7 \parallel 9$

Extra

Komplexität:

- Worst case: $\mathbb{O}(n^2)$
- Best case: $\mathbb{O}(n \cdot log(n))$

Heap

Hilfs-Datenstruktur: Binärbaum
Eigenschaften:

- vollständig
- Knoten-Wert $>$ Kinder-Werte (Heap-Eigenschaft)

Heapsort

1. Erzeuge aus den Daten einen Heap
   - 1.1 Ordne die Daten ebenenweise zu einem Binärbaum an
   - 1.2 Solange Heap-Eigensachaft nicht hergestellt ist:
     Lasse einen Knoten (soweit wie möglich) unter größtes Kind sinken
     $\rightarrow$ Knoten von unten nach oben durchprobieren
2. Nutze den Heap zum Sortieren
   - 2.1 Tausche die Wurzel und das unterste "freie" Element und setze das Hinuntergetauschte fest
   - 2.2 Lasse die Wurzel soweit wie möglich sinken

3)

zu ordnen: 2, 0, 9, 3, 5, 8, 4, 1, 6, 7
siehe Tafel

Extra

Komplexität:

- Worst case: $\mathbb{O}(n \cdot log(n))$
- Best case: $\mathbb{O}(n \cdot log(n))$