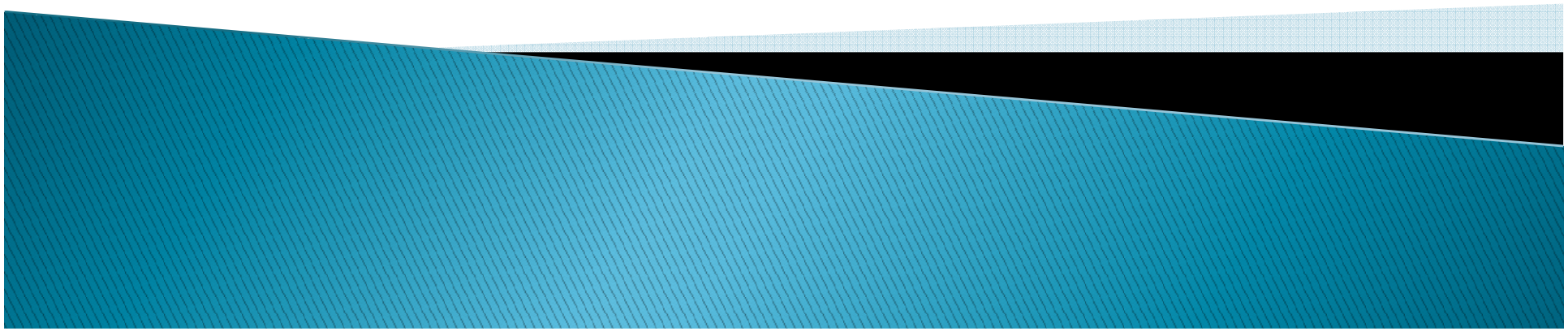
Teile und Herrsche 1



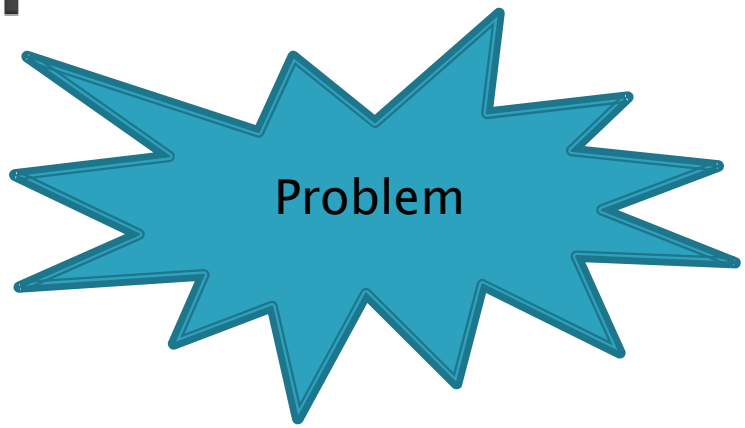
Gliederung

- Die Idee
- Quicksort
  - Verfahrensbeschreibung
  - Funktionsorientierte Beschreibung
  - Effizienzanalyse
  - Internes Suchverfahren
- Mergesort
  - Verfahrensbeschreibung
  - Effizienzanalyse
- Vergleich Quicksort vs. Mergesort

Die Idee

- Divide & Conquer, Divide et impera
- angeblicher Ausspruch von Ludwig XI. (französischer König)
- Entwurfsmethode für Algorithmen

Prinzip



Aufteilen des Gesamtproblems in Kleinere

Lösen der Teilprobleme

Verbinden der Teillösungen zur Gesamtlösung

Merkmale

- Grundsätzlich rekursiv
- Effizienzverbesserung gegenüber anderer Verfahren

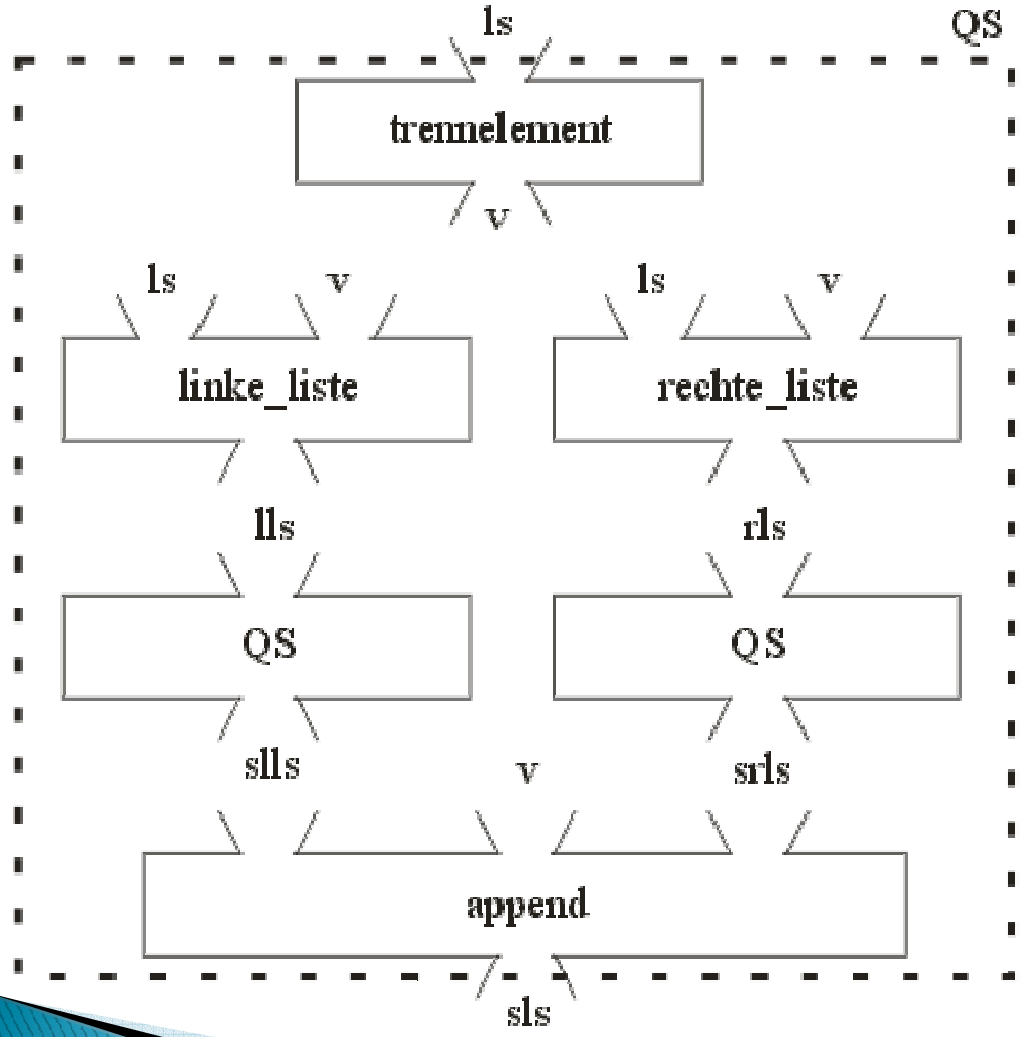
Quicksort

- Musterbeispiel für Teile und Herrsche
- 1962 von C.A.R. Hoare
- Kein zusätzlicher Speicherplatz nötig

Verfahrensbeschreibung

- Bildung von 2 Teillisten anhand eines Vergleichselementes (Pivot-Element)
- Linke Teilliste enthält alle Werte die kleiner sind als das Pivot-Element
- Rechte Teilliste alles was größer oder gleich diesem Element ist
- Auf jede Teilliste wird wieder Quicksort angewandt, bis die Basisfälle eintreten (leere Liste oder Liste enthält nur 1 Element)

Funktionsorientierte Beschreibung



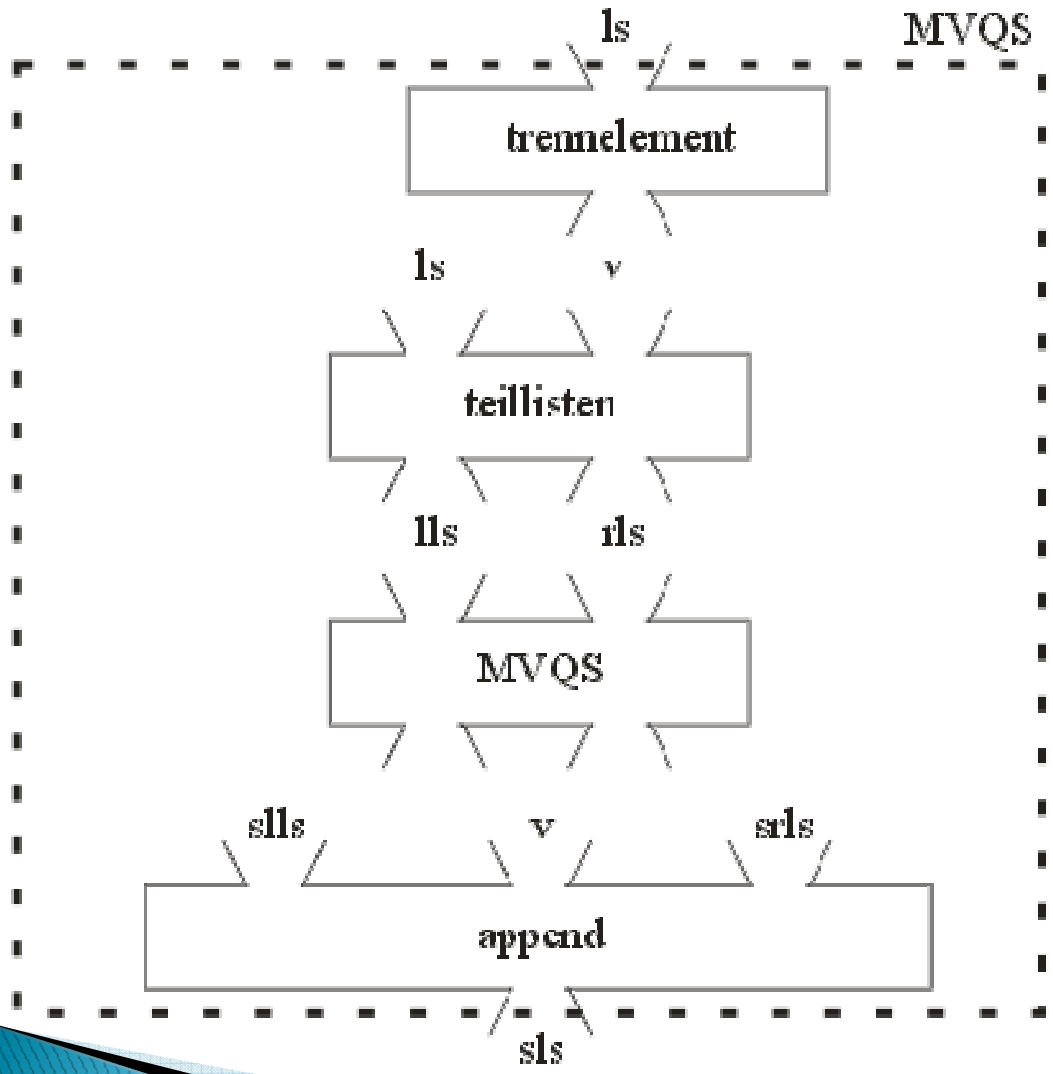
QS	Quicksort
v	Vergleichselement
ls	Gesamtliste
lls	linke Liste
rls	rechte Liste
slls	sortierte lls
srls	sortierte rls
sls	sortierte ls

Multiple-Value-Context

- Funktion besitzt mehrere Rückgabetypen
- Nicht alle Programmiersprachen besitzen eine solche Funktionalität (Java, C, C#,...)
- Ausnahmen: Scheme, Lisp, ...

- 2 Sprachelemente von Nöten:
  - (values v_1 v_2 ... V_n)
  - (call-with-values p c)

Funktionsorientierte Beschreibung



MVQS	Multiple-Value-Quicksort
v	Vergleichselement
ls	Gesamtliste
lls	linke Liste
rls	rechte Liste
slls	sortierte lls
srls	sortierte rls
sls	sortierte ls

Effizienzanalyse best-case

$$T(0) = 0$$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + c \cdot n$$

Anwendung Fall 2
der Meistermethode

$$T(n) = \mathrm{O}(n \cdot \log_2 n)$$

Verarbeitung von 2 Teillisten

Eine Teilliste ist halb so groß wie die Gesamtliste

Zum Aufteilen muss die Ursprungsliste einmal durchlaufen werden.

Effizienzanalyse worst-case

$$T(0) = 0$$

$$T(n) = T(0) + T(n-1) + c \cdot n$$

Leere Teilliste
→ kein Aufwand

Zweite Teilliste enthält Gesamtliste ohne Pivot-Element

Zum Aufteilen muss die Ursprungsliste einmal durchlaufen werden.

Effizienzanalyse worst-case

$$T(0) = 0$$

$$T(n) = T(0) + T(n-1) + c \cdot n$$

Iterationsmethode

$$T(n-1) = T(n-2) + c(n-1)$$

$$T(n-2) = T(n-3) + c(n-2)$$

$$\vdots$$

$$T(1) = T(0) + c(1)$$

Leere Teilliste
→ kein Aufwand

Zweite Teilliste enthält Gesamtliste ohne Pivot-Element

Zum Aufteilen muss die Ursprungsliste einmal durchlaufen werden.

Effizienzanalyse worst-case

$$T(0) = 0$$

$$T(n) = T(0) + T(n-1) + c \cdot n$$

Iterationsmethode

$$T(n-1) = T(n-2) + c(n-1)$$

$$T(n-2) = T(n-3) + c(n-2)$$

$$\vdots$$

$$T(1) = T(0) + c(1) \longrightarrow T(n) = O(n^2)$$

Leere Teilliste
→ kein Aufwand

Zweite Teilliste enthält Gesamtliste ohne Pivot-Element

Zum Aufteilen muss die Ursprungsliste einmal durchlaufen werden.

Effizienzanalyse average-case

$$T(i) = T(n-i-1) = \frac{1}{n}\sum_{j=0}^{n-1} T(j)$$

$$T(n) = T(i) + T(n-i-1) + c \cdot n$$

linke Teilliste

rechte Teilliste

$$T(n) = \frac{2}{n}\sum_{j=0}^{n-1} T(j) + c \cdot n$$

Effizienzanalyse average-case

$$T(n) = \frac{2}{n} \sum_{j=0}^{n-1} T(j) + c \cdot n \qquad | \cdot n$$

Effizienzanalyse average-case

$$T(n) = \frac{2}{n} \sum_{j=0}^{n-1} T(j) + c \cdot n \qquad | \cdot n$$

Beseitigung der Summe:

(1) $$nT(n) = 2\sum_{j=0}^{n-1} T(j) + c \cdot n^2 \qquad | \; n := n+1$$

(2) $$(n-1)T(n-1) = 2\sum_{j=0}^{n-2} T(j) + c \cdot (n-1)^2 \qquad | \; (1)-(2)$$

Effizienzanalyse average-case

$$T(n) = \frac{2}{n}\sum_{j=0}^{n-1} T(j) + c \cdot n \qquad | \cdot n$$

Beseitigung der Summe:

(1) $$nT(n) = 2\sum_{j=0}^{n-1} T(j) + c \cdot n^2 \qquad | \; n := n+1$$

(2) $$(n-1)T(n-1) = 2\sum_{j=0}^{n-2} T(j) + c \cdot (n-1)^2 \qquad | \; (1)-(2)$$

$$nT(n) - (n-1)T(n-1) = 2T(n-1) + 2cn - c \longrightarrow$$

c entfällt für $n \to \infty$

Effizienzanalyse average-case

$$nT(n)-(n-1)T(n-1)=2T(n-1)+2cn$$ | Umstellen

$$\frac{T(n)}{n+1}=\frac{T(n-1)}{n}+\frac{2c}{n+1}$$ | n:=n-1

$$\frac{T(n-1)}{n}=\frac{T(n-2)}{n-1}+\frac{2c}{n}$$

$$\vdots$$ schrittweise Substitution

$$\frac{T(2)}{3}=\frac{T(1)}{2}+\frac{2c}{3}$$

$$\frac{T(n)}{n+1}=\boxed{\frac{T(1)}{2}}+2c\cdot\boxed{\sum_{i=3}^{n+1}\frac{1}{i}}$$

0, da T(1)=0

Harmonische Zahlen

Effizienzanalyse average-case

Harmonische Zahlen: $H_n = \sum_{k=1}^{n} \frac{1}{k} = \frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{n}$

Bekannte Schranken: $\frac{\lfloor \log_2 n \rfloor + 1}{2} < H_n \leq \lfloor \log_2 n \rfloor + 1$

Effizienzanalyse average-case

Harmonische Zahlen: $H_n = \sum_{k=1}^{n} \frac{1}{k} = \frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{n}$

Bekannte Schranken: $\frac{\lfloor \log_2 n \rfloor + 1}{2} < H_n \leq \lfloor \log_2 n \rfloor + 1$

$$\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c \sum_{i=3}^{n+1} \frac{1}{i}$$

Effizienzanalyse average-case

Harmonische Zahlen: $H_n = \sum_{k=1}^{n} \frac{1}{k} = \frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{n}$

Bekannte Schranken: $\frac{\lfloor \log_2 n \rfloor + 1}{2} < H_n \leq \lfloor \log_2 n \rfloor + 1$

$$\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c \sum_{i=3}^{n+1} \frac{1}{i}$$

$$\frac{T(n)}{n+1} = 0 + O(\log_2 n)$$

Effizienzanalyse average-case

Harmonische Zahlen: $H_n = \sum_{k=1}^{n} \frac{1}{k} = \frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{n}$

Bekannte Schranken: $\frac{\lfloor \log_2 n \rfloor + 1}{2} < H_n \leq \lfloor \log_2 n \rfloor + 1$

$$\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c \sum_{i=3}^{n+1} \frac{1}{i}$$

$$\frac{T(n)}{n+1} = 0 + O(\log_2 n)$$

$$T(n) = O(n \log_2 n)$$

Internes Suchverfahren

- Verstoß gegen wichtige Forderung!
  - Im Vergleich zur Vorgabe, darf kein weiterer Speicher verwendet werden
  - → internes Suchverfahren
- Bei der Teillistenbildung werden 2 neue Listen erstellt
- Lösung: Implementierung als Vektor → freier Zugriff auf Elemente

Internes Suchverfahren

- Pivot-Element ist 1. Vektorelement (a[0])
- Linker Zeiger steht auf a[1]
- Rechter Zeiger auf a[n-1] (letztes Element)
- Linker Zeiger wird nach rechts verschoben, bis gilt: a[l] > v
- Rechter Zeiger läuft nach links bis gilt a[r] < v
- Wenn l <= r , dann Tausche a[l] mit a[r] und setze Zeigerwanderung fort
- Sonst, Tausche a[r] mit a[0] und gib Vektor zurück (Pivot-Element an richtiger Position)

Mergesort

- Sortieren durch Mischen oder auch Sortieren durch Verschmelzen genannt
- 1945 von John von Neumann vorgestellt
- Stabiles Sortierverfahren

Verfahrensbeschreibung

- Teilung der Liste in etwa 2 gleich große Teillisten
- Anwendung dieses Schrittes auf die Teillisten (Rekursion)
- Basisfall: nur 1 Element in Liste enthalten
- Vergleich der jeweils ersten Elemente der Teillisten

Funktionsorientierte Beschreibung

- Anwendung von Multiple-Value-Context
- Implementierung mit Hilfe von 3 Funktionen
  - mergesort
  - teillisten
  - verbinden
- Vollständiger Code ist im Wiki zu finden

Effizienzanalyse

- Rekursionsgleichung: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n)$
- Basisfall: $T(1) = 0$
- Ergebnis: $T(n) = O(n \cdot \log_2 n)$

für best-, average- und worstcase

Vergleich Quicksort vs. Mergesort

	Quicksort	Mergesort
best-case	$n \cdot \log_2(n)$	$n \cdot \log_2(n)$
average-case	$n \cdot \log_2(n)$	$n \cdot \log_2(n)$
worst-case	n^2	$n \cdot \log_2(n)$

- Grafische Vergleichsmöglichkeiten im Wiki