

Algorithmen und Datenstrukturen

Sommersemester 2020

Stefan Näher

Universität Trier

naeher@uni-trier.de

Vorlesung 6

7. Mai 2020

Laufzeitanalyse von Heapsort

Satz: Heapsort hat Laufzeit $\mathcal{O}(n \log n)$.

Beweis:

- Die Aufbauphase hat Laufzeit $\mathcal{O}(n)$
- Die Selektionsphase hat Laufzeit $\mathcal{O}(n \log n)$

Analyse der Aufbauphase

Es sei $h(i)$ die Höhe des von Knoten i im Heap A .

Der Aufwand für $SINK(i, n)$ lässt sich also mit $\mathcal{O}(h(i))$ abschätzen.

Beobachtung: Die Anzahl der Knoten auf Höhe h ist höchstens $\frac{n}{2^h}$.

Dann sind die Gesamtkosten aller $SINK$ -Aufrufe

$$\sum_{i=1}^{n/2} h(i) \leq \sum_{h=1}^{\lceil \log_2(n) \rceil} \left(h \cdot \frac{n}{2^h} \right) \leq n \cdot \sum_{h=1}^{\infty} \frac{h}{2^h} = n \cdot \sum_{h=1}^{\infty} h \cdot \left(\frac{1}{2} \right)^h \leq 2 \cdot n$$

Integrierende Reihe

$$\sum_{i=0}^{\infty} i \cdot x^i = \frac{x}{(1-x)^2} = 2 \quad \text{für } x = \frac{1}{2}$$

Analyse der Selektionsphase

Da $h \leq \lceil \log_2(n) \rceil$, ist der Gesamtaufwand für die Selektionsphase: $\mathcal{O}(n \cdot \log n)$.

Varianten & Geschichte

Heapsort ist ein 1964 von Robert W. Floyd und J. W. J. Williams entwickeltes, relativ schnelles Sortierverfahren. Es handelt sich um eine Verbesserung von Selectionsort.

BottomUp-Heapsort

ist ein Sortieralgorithmus, der u.a. 1990 von Ingo Wegener vorgestellt wurde und im Durchschnitt besser als Quicksort arbeitet, falls man Vergleichsoperationen hinreichend stark gewichtet. Im Durchschnittsfall benötigt BottomUp-Heapsort nur $n \log_2(n) + \mathcal{O}(n)$ Schlüsselvergleiche, selbst im schlimmsten Fall nur $n \log_2(n) + \mathcal{O}(n \log \log(n))$. Es wird ausgenutzt, dass “meist” in die Nähe der Blattebene abgesenkt werden muss.

Quicksort: Sortieren durch Teilen:

Wie Heapsort geht Quicksort auf Hoare zurück.

Es zählt “im Mittel” bis heute zu den schnellsten Sortierverfahren.

Im schlimmsten Fall zeigt es jedoch **quadratische Laufzeit**.

Quicksort: Die Grundidee

Sortiere $A[1..n]$ nach dem Teile-und-Herrsche-Prinzip.

1. Teile: Partitioniere das Feld A bzgl. des ersten Elements $y = A[1]$ in 2 Mengen

$L = \{x \in A \mid x < A[1]\}$ und $R = \{x \in A \mid x > A[1]\}$.

sei $k = |L| + 1$

speichere: L in $A[1..k-1]$, y in $A[k]$, R in $A[k+1..n]$.

SKIZZE

2. Herrsche: Wende das Verfahren rekursiv auf $A[1..k-1]$ und $A[k+1..n]$ an.

3. Zusammenfügen: nichts zu tun.

$y = A[1]$ heißt auch **Pivot-Element** der Partitionierung.

Quicksort: Die rekursive Funktion

1. **QUICKSORT**(ℓ, r)
2. // sortiert das Teilfeld $A[\ell..r]$ aufsteigend
3. **if** $\ell \geq r$ **then**
4. // Verankerung (nichts zu tun)
5. **return** ;
6. **fi**
7. $k \leftarrow$ **PARTITION**(ℓ, r);
8. **QUICKSORT**($\ell, k - 1$);
9. **QUICKSORT**($k + 1, r$);

Quicksort: PARTITION

SKIZZE

Analyse: Der schlimmste Fall

Die Kosten eines Aufrufs von $\text{QUICKSORT}(\ell, r)$ sind:

- 1) die unmittelbaren Aufteilungskosten: $\mathcal{O}(r - \ell + 1)$ Vergleiche,
- 2) die Kosten der rekursiven Aufrufe.

Sei $QS(n)$ die max. Zahl von Vergleichen, die Quicksort auf einem Feld der Länge n durchführt. Dann gilt:

$QS(0) = QS(1) = 0$ und für $n > 1$

$$QS(n) = n + \max_{1 \leq j \leq n} (QS(j-1) + QS(n-j))$$

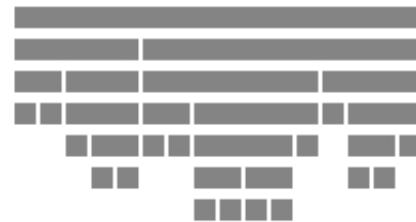
Mit Induktion sieht man leicht $QS(n) \leq \sum_{i=1}^n i = \frac{1}{2} \cdot n \cdot (n+1) = \mathcal{O}(n^2)$.

Hinweis: Bei der Eingabe $1, 2, 3, \dots, n$ führt Quicksort $\mathcal{O}(n^2)$ Vergleiche durch.

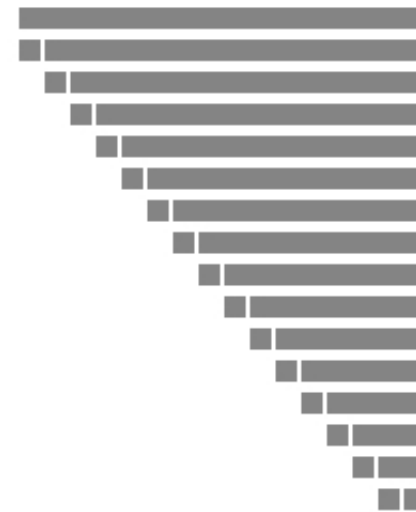
Quicksort—in verschiedenen Fällen



(a)



(b)



(c)

(a) Der Idealfall, (b) ein mittlerer Fall, (c) der schlechteste Fall