

Algorithmen und Datenstrukturen

Sommersemester 2020

Stefan Näher

Universität Trier

naeher@uni-trier.de

Vorlesung 7

13. Mai 2020

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**($k + 1, r$);

PARTITION(ℓ, r)

1. $x \leftarrow A[\ell];$
2. $i \leftarrow \ell + 1;$
3. $j \leftarrow r;$
4. **repeat**
5. **while** $i \leq r \wedge A[i] < x$ **do** $i++;$ **od**
6. **while** $j \geq \ell + 1 \wedge A[j] > x$ **do** $j--;$ **od**
7. **if** $i < j$ **then**
8. $A[i] \longleftrightarrow A[j];$
9. $i++;$
10. $j--;$
11. **fi** ;
12. **until** $i > j;$
13. $A[\ell] \longleftrightarrow A[j];$
14. **return** $j;$

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:

$$QS(0) = QS(1) = 0 \text{ und für } n > 1$$

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

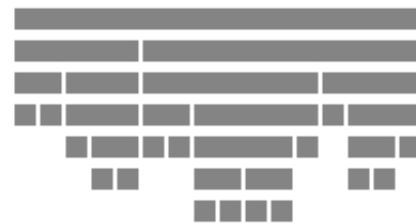
Mit Induktion sieht man leicht $QS(n) \leq \sum_{i=1}^n i = \frac{1}{2} \cdot n \cdot (n+1) = \mathcal{O}(n^2)$.

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

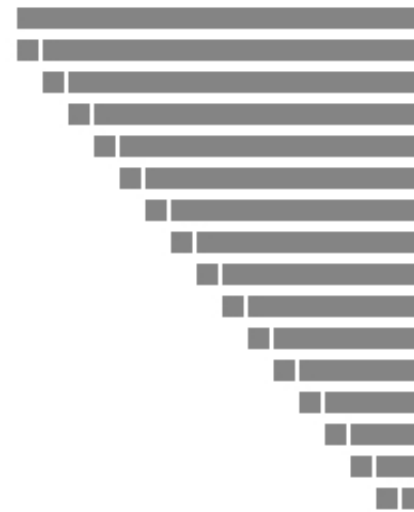
Quicksort—in verschiedenen Fällen



(a)



(b)



(c)

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

Quicksort — Der mittlere Fall

Wir machen jetzt ein paar weitere Annahmen über $A[1..n]$:

1. Alle Elemente sind paarweise verschieden.
2. Jeder der $n!$ vielen möglichen Permutationen ist gleichwahrscheinlich.

Lemma: Die rekursiv zu lösenden Teilprobleme der Größe $k-1$ und $n-k$ erfüllen wiederum die beiden Annahmen.

(ohne Beweis: lästiges Rechnen mit Wahrscheinlichkeiten.)

O.E. im Folgenden: zu sortierende Zahlen aus $\{1, \dots, n\}$.

Quicksort — Der mittlere Fall

$\overline{QS}(n)$: mittlere Anzahl Vergleiche, die Quicksort bei einem Feld der Größe n durchführt.

$\overline{QS}(n)$ ist **Erwartungswert** der Zufallsvariablen # Vergleiche

Es gilt:

$$\overline{QS}(n) = n + \frac{1}{n} \sum_{k=1}^n (\overline{QS}(k-1) + \overline{QS}(n-k)) = n + \frac{2}{n} \sum_{k=0}^{n-1} \overline{QS}(k).$$

Multiplizieren mit n und Ersetzen von n durch $(n+1)$ ergibt

$$n \cdot \overline{QS}(n) = n^2 + 2 \sum_{k=0}^{n-1} \overline{QS}(k) \text{ bzw.}$$

$$(n+1) \cdot \overline{QS}(n+1) = (n+1)^2 + 2 \sum_{k=0}^n \overline{QS}(k).$$

Betrachte die Differenz

$$(n+1) \cdot \overline{QS}(n+1) - n \cdot \overline{QS}(n) = 2n+1 + 2\overline{QS}(n)$$

$\leadsto$

$$\overline{QS}(n+1) \leq 2 + \frac{n+2}{n+1} \overline{QS}(n) \leq 2 + \frac{n+2}{n+1} \left(2 + \frac{n+1}{n} \left(2 + \frac{n}{n-1} \dots \right) \right)$$

$$\overline{QS}(n+1) \leq (n+2) \left(\frac{2}{n+2} + \frac{2}{n+1} + \frac{2}{n} \dots \right) = \mathcal{O}(n \log n)$$

$H_n = \sum_{k=1}^n \frac{1}{k}$ heißt die n -te **Harmonische Zahl**

Es gilt: $H_n \leq \ln n + 1 = \mathcal{O}(\log n)$.

Abschließende Folgerungen

Wir sollten versuchen, den schlimmsten Fall zu vermeiden.

Median of Three

Wähle als Pivot-Element das mittlere von 3 ausgewählten Elementen aus $A[\ell, r]$
z.B. von $A[\ell]$, $A[(\ell + r)/2]$, und $A[r]$

Randomized Quicksort

Wähle ein zufälliges Pivot-Element aus $A[\ell, r]$ statt immer das erste.
Füge dazu am Anfang von $\text{PARTITION}(\ell, r)$ ein:

$k \leftarrow \text{random}(\ell, r);$
 $A[\ell] \longleftrightarrow A[k];$

Diskussion der Sortierverfahren

Wir haben bis jetzt zwei Verfahren kennengelernt, die eine Laufzeit von $\mathcal{O}(n \cdot \log(n))$ versprechen: Mergesort (ganz zu Anfang) und Heapsort.

Alle übrigen Verfahren (Sortieren durch Einfügen oder Auswahl, Bubblesort bzw. Quicksort) haben Laufzeit $\mathcal{O}(n^2)$ im schlimmsten Fall, wobei Quicksort heraussticht, da es im mittleren Fall selbst die oben genannten Verfahren schlägt.

Ist mit $\mathcal{O}(n \log(n))$ das “Ende der Fahnenstange” erreicht ?

Sicher müssen sich Sortierverfahren alle vorliegenden Daten anschauen dürfen, was sofort eine **triviale untere Schranke von $\Omega(n)$** liefert.

Wären aber Sortierverfahren denkbar, die z.B. in $\mathcal{O}(n \log(\log(n)))$ laufen ?

Was heißt eigentlich “Sortieren” ?

Ausgangslage: Folge A von paarweise verschiedenen “Gegenständen” von einem (abstrakten) Datentyp D , d.h.: $A : D[1..n]$.

D soll uns Vergleichsoperatoren $<$, $\leq$ und $=$ zur Verfügung stellen, mit dessen Hilfe wir Vergleiche der Art $A[i] < A[j]$ in $\Theta(1)$ Zeit durchführen können.

Auf der dem Datentyp zugrundeliegenden Menge soll $\leq$ eine lineare Ordnung (manchmal auch totale Ordnung genannt) darstellen, d.h. insbesondere, dass stets $x \leq y$ oder $y \leq x$ für zwei beliebige Elemente gilt.

Ziel: Finde Permutation $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$, sodass

$$A[\pi(1)] < A[\pi(2)] < \dots < A[\pi(n)].$$

Frage: Wie viele Vergleiche muss ein Sortierverfahren S mindestens im schlimmsten Falle durchführen, wenn der Ablauf von S für festes n nur von den Ergebnissen der durchgeführten Vergleiche abhängen darf ?

Exkurs: nochmal Binärbäume

Wir hatten bereits gesehen:

Lemma: Die minimale Höhe eines Binärbaumes mit m Knoten ist $h = \lceil \log_2(m + 1) \rceil - 1$.

Leicht per Induktion einzusehen ist:

Lemma: Ein Binärbaum mit b Blättern hat mindestens $2b - 1$ Knoten insgesamt.

Daraus ergibt sich:

Folgerung: Die minimale Höhe eines Binärbaumes mit b Blättern ist $h = \lceil \log_2(b) \rceil$.

Beweis: b Blätter $\leadsto$ mindestens $2b - 1$ Knoten; um eine kleinstmögliche Höhe zu erzielen, wird diese Anzahl auch angenommen, wodurch für die kleinstmögliche Höhe h folgt:

$$h = \lceil \log_2(2b) \rceil - 1 = \lceil \log_2(b) \rceil.$$

Untere Schranke — ein Beweis

Vergleichsabhängiger Algorithmenverlauf und Notwendigkeit, $n!$ viele Anordnungen herstellen zu müssen, liefert **binären Entscheidungsbaum** mit $n!$ vielen Blättern für jeden Sortieralgorithmus.

Die Anzahl der schlimmstenfalls notwendigen Vergleiche entspricht der Länge eines Pfades von der Wurzel zu irgendeinem Blatt, also der Höhe h_n des Entscheidungsbaumes.

Mit dem folgenden (leicht zu zeigenden) Lemma folgt die behauptete untere Schranke mit obiger Folgerung sofort, denn:

$$h_n = \Omega(\log n!) = \Omega(n \log n)$$

Lemma: $n! \geq (n/2)^{n/2}$. Genaueres würde die **Stirlingsche Formel** liefern.