Quicksort

am häufigsten eingesetztes Sortierverfahren

Vorteile:

- average-case Laufzeit: $O(n \log n)$
- kleine Konstanten, wenig Overhead

Nachteile:

- worst-case Laufzeit: $\Theta(n^2)$
- empfindlich bzgl. Variation der Implementierung

Idee: divide and conquer

- Wähle ein Element x als Pivot,
- Partitioniere die Eingabe in Elemente kleiner bzw. größer als x (bei Gleichheit beliebig); in-place Realisierung: Permutiere die Einträge so, dass kleinere Elemente vor und größere Elemente nach x stehen,
- Sortiere beide Teilfelder rekursiv.

Pseudocode

algorithm *PARTITION* (A, p, r)
//Annahmen: $p < r$; $A[p-1] \leq \min\{A[p], \ldots, A[r]\}$
1 $x := A[r]$; $i := p - 1$; $j := r$;
2 **while** (true)
3 **repeat** $i := i + 1$; **until** $(A[i] \geq x)$;
4 **repeat** $j := j - 1$; **until** $(A[j] \leq x)$;
5 **if** $(i < j)$
6 *swap*$(A[i], A[j])$;
7 **else**
8 *swap*$(A[i], A[r])$;
9 **return** i;

algorithm *QUICK-SORT* (A, p, r)
1 **if** $(p < r)$
2 $q :=$ *PARTITION* (A, p, r);
3 *QUICK-SORT* $(A, p, q - 1)$;
4 *QUICK-SORT* $(A, q + 1, r)$;

Korrektheit von PARTITION

Behauptung (Invarianten): Es gelten:

1. unmittelbar nach Zeile 2:

$$\begin{aligned}\max\{A[k] \mid p-1 \le k \le i\} \;&\le\; x \;\le\\ \min\{A[l] \mid\quad j \le l \le r\},&\end{aligned}$$

2. unmittelbar nach Zeile 3 oder 4 bei $i < j$:

$$\begin{aligned}\max\{A[k] \mid p-1 \le k < i\} \;&\le\; x \;\le\\ \min\{A[l] \mid\quad j < l \le r\},&\end{aligned}$$

3. unmittelbar nach Zeile 3 oder 4 bei $i = j$:

$$\begin{aligned}\max\{A[k] \mid p-1 \le k < i\} \;&\le\; x \;\le\\ \min\{A[l] \mid\quad j \le l \le r\}.&\end{aligned}$$

Beweis: Induktion über die Anzahl der Durchläufe der *while*-Schleife.

Korrektheit von PARTITION

Behauptung: PARTITION liefert $q \in \{p, \ldots, r\}$:

(A) $\max\{A[k] \mid p \leq k < q\} \leq A[q] \leq \min\{A[l] \mid q < l \leq r\}$

Beweis: Betrachte den ersten Zeitpunkt t mit $i = j =: h$. Beachte: Invariante (3) gilt, insbesondere $A[h] \geq x$.

- **Fall I)** t liegt nach Zeile 3:
  Zeile 4 wird noch genau einmal ausgeführt, danach $j = i - 1 = h - 1$;
  Zeile 8 wird ausgeführt, danach $A[h] = x$;
  $h(= i)$ wird zurückgegeben;
  Behauptung gilt mit $q := h$.
- **Fall II.a)** t liegt nach Zeile 4, $A[h] > x$:
  Wie **(I)**.
- **Fall II.b)** t liegt nach Zeile 4, $A[h] = x$:
  Zeile 4 wird verlassen;
  Zeile 8 wird (ohne Wirkung) ausgeführt;
  $h(= i = j)$ wird zurückgegeben;
  Behauptung gilt mit $q := h$.

Laufzeit von PARTITION

Behauptung: PARTITION führt auf einem Array der Länge n entweder n oder $n+1$ Schlüsselvergleiche (also Vergleiche in den Zeilen 3 und 4) aus.
Beweis: Betrachte den Wert $(j-i)$:
nach der Initialisierung: $(j-i)=n$;
mit jeder Ausführung der Zeile 3 oder 4: Dekrementierung von $(j-i)$ um 1;
bei der Terminierung: $j=i-1$ (Fall (I) oder (II.a)) oder $j=i$ (Fall (II.b)).
Beobachtung: paarweise verschiedene Schlüssel:
$x=A[r]\neq A[h]$ für alle $h\in\{p,\ldots,r-1\}$,
also Fall (II.b) mit $h<r$ und $A[h]=x$ ausgeschlossen,
damit $n+1$ Schlüsselvergleiche.

Behauptung: Die Laufzeit von PARTITION auf einem Array der Länge n ist $\Theta(n)$.
Beweis: Konstante Zeit in den Zeilen 1,8,9;
auf jede Ausführung der Zeile 6 folgt mindestens eine Ausführung der Zeile 3;
also Laufzeit bis auf konstante Faktoren: n.

Korrektheit von QUICKSORT

Annahme: $A[p-1] \leq \min\{A[p], \ldots, A[r]\}$ (sentinel $A[0] = -\infty$ beim Sortieren von $A[1 \cdot\cdot\, n]$).

Behauptung: Der Aufruf QUICK-SORT(A, p, r) terminiert und permutiert $A[p \cdot\cdot\, r]$ so, dass: $\forall i \in \{p, \ldots, r-1\}: \; A[i] \leq A[i+1]$ **(B)**.

Beweis: (Induktion über $(r-p)$)

I.A. $(r-p) < 1$: Beh. gilt trivialerweise.

I.S. $(r-p) \geq 1$: Terminierung folgt aus der I.V. Fallunterscheidung nach i:

- **Fall I)** $i \in \{p, \ldots, q-2\} \cup \{q+1, \ldots, r-1\}$: (B) gilt nach I.V.
- **Fall II)** $i = q-1$: (B) gilt wegen Eigenschaft (A) von PARTITION.
- **Fall III)** $i = q$: wie (II).

Bemerkung: Annahme (sentinel) überflüssig bei Modifikation von PARTITION:

4 **repeat** $j := j-1$; **until** $(i \geq j$ **or** $A[j] \leq x)$;

(etwas ineffizienter wegen zusätzlicher Abfrage in einer „inneren“ Schleife).

Laufzeit von QUICKSORT

Annahme: paarweise verschiedene Schlüssel.

QUICK-SORT auf einem Array der Länge n:
$C(n) :=$ Anzahl der Schlüsselvergleiche,
$T(n) :=$ Laufzeit.

worst case: immer Maximum als Pivot:
Teilfeldgröße m reduziert sich immer um 1;
$C(n) = \sum_{m=2}^{n}(m+1) = \frac{1}{2}n^2 + \frac{3}{2}n - 2$
$T(n) \in \Theta(n^2)$.

best case: immer Median als Pivot:
Teilfeldgröße wird immer mindestens halbiert;
$C(n) \leq 2C(\lfloor n/2 \rfloor)+n+1 \Rightarrow C(n) \in O(n \log n)$
$C(n) \in \Omega(n \log n)$ (Induktion)
$T(n) \in \Theta(n \log n)$.
Bemerkung: gilt auch, wenn eine Konstante $\alpha > 0$ existiert, so dass kein Teilfeld mehr als Faktor α kleiner ist als das aktuelle Feld.
Bemerkung: Median kann man in Linearzeit finden, also Abwandlung von Quicksort realisierbar, deren Laufzeit immer $\Theta(n \log n)$ ist (unpraktisch wegen großer Konstanten).

Laufzeit von QUICKSORT

Annahme: Für jedes $k \in \{1, \ldots, n\}$ ist das Pivot mit Wahrscheinlichkeit $1/n$ das k-te Element in der sortierten Folge (gilt z.B., wenn Eingaben zufällige Permutationen sind).

average case: Mit Wahrscheinlichkeit $1/n$ ergeben sich Teilprobleme der Größe $k-1$ und $n-k$. Sei C_n die erwartete Anzahl der Schlüsselvergleiche ($C_0 = C_1 = 0$):

$$C_n = n+1+\frac{1}{n}\sum_{k=1}^{n}(C_{k-1}+C_{n-k}) = n+1+\frac{2}{n}\sum_{k=0}^{n-1} C_k$$

$$\left.\begin{array}{rcl} n\ C_n & = & n(n+1)+2\sum_{k=0}^{n-1} C_k \\ (n-1)C_{n-1} & = & n(n-1)+2\sum_{k=0}^{n-2} C_k \end{array}\right\} \Rightarrow$$

$$nC_n-(n-1)C_{n-1} = 2n+2C_{n-1} \Rightarrow$$

$$C_n = 2+\frac{n+1}{n}C_{n-1}$$

Laufzeit von QUICKSORT

$$\frac{C_n}{n+1} = \frac{2}{n+1} + \frac{C_{n-1}}{n} = \ldots = \frac{2}{n+1} + \frac{2}{n} + \cdots + \frac{2}{4} + \frac{C_2}{3}$$

$$\Rightarrow C_n = 2 + 2(n+1)\left(\frac{1}{n} + \cdots + \frac{1}{4} + \frac{1}{2}\right) \qquad (C_2 = 3)$$

$$\Rightarrow C_n = 2 + 2(n+1)\left(H_n - \frac{4}{3}\right) \quad \left(H_n = \sum_{k=1}^{n} \frac{1}{k}\right)$$

$$\Rightarrow C_n \leq 2 + 2(n+1)\left(\ln n - \frac{1}{3}\right) \quad (\ln n < H_n \leq \ln n + 1)$$

$$\Rightarrow C_n = 2n \ln n - O(n)$$

$$\Rightarrow C_n \approx 1.387\, n \log n - O(n)$$

Es folgt direkt, dass die erwartete Laufzeit von QUICK-SORT (mit den genannten Annahmen) $\Theta(n \log n)$ ist.

Bemerkung: Wir haben die Tatsache benutzt, dass die Teilprobleme wieder zufällig sind (Übungsaufgabe). Der Sachverhalt wird einfacher, wenn man die randomisierte Variante von Quicksort (wie nachfolgend beschrieben) benutzt.

Varianten von Quicksort

Randomized Quicksort: Pivot zufällig.

Vor Zeile 2: *swap*$(A[r], A[$*random*$(p, r)])$; *random*(p, r) liefert zufällig (mit gleicher Wahrscheinlichkeit) eine Zahl aus $\{p, \ldots, r\}$.

Realisierung mit (Pseudo-)Zufallszahl-Generator.

Analyse exakt wie vorher.

- keine Annahmen über die Eingabe
- keine fixierte worst-case Eingaben mehr

Median-of-3 Partition:

Pivot: Median von drei zufällig gewählten Elementen.

- $C_n \approx 1.188\ n \log n - O(n)$
- empirisch etwa 5% schneller

Varianten von Quicksort

Gesonderte Behandlung kleiner Teilfelder:

Teilfelder der Länge kleiner als M (z.B. für ein M im Bereich 5-20) gesondert behandeln (z.B. mit Insertion-Sort sortieren).

- spart viele rekursive Aufrufe
- empirisch etwa 10% schneller

„Nicht-rekursive“ Implementierung:

Explizite Verwendung eines Stacks (siehe Übungsaufgabe).

- Stack-Tiefe auf $O(\log n)$ beschränkbar

Berücksichtigung von Duplikat-Schlüsseln:

Bei häufiger Gleichheit unter den Schlüsseln: andere Partitionierungsmethoden.

Dutch National Flag Problem: Bei Pivot x erzeuge drei Teilfelder wie folgt:

$< x$	$= x$	$> x$