Bemerkung.
- QuickSort gehört zu den empirisch schnellsten Sortierverfahren.
- Es beruht – wie MergeSort – auf Divide & Conquer
- Im Gegensatz zu MergeSort ist der Divide-Schritt aufwendiger, der Conquer-Schritt dafür einfacher.

8.10 Laufzeitanalyse von QuickSort

Satz 8.1. *QuickSort arbeitet korrekt. Die Laufzeitfunktion ist in $\Theta(n^2)$ für $n = |S|$.*

Laufzeitbeweis. Betrachte Rekursionsbaum. Knoten eines Levels repräsentieren disjunkte Teilmengen von S, da S_1 und S_2 in Schritt 5 disjunkt sind. Aufspalten (Divide) einer Teilmenge S' in zwei disjunkte Teilmengen in Schritt 5 benötigt $|S'| - 1$ Vergleiche und Laufzeit $\mathcal{O}(|S'|)$. Alle auf einem Level durchgeführten Operationen benötigen Laufzeit $\mathcal{O}(n)$. Da rekursive Aufrufe für echt kleinere Teilmengen erfolgen, ist die Rekursionstiefe höchstens n. Daraus folgt die Laufzeitfunktion in $\mathcal{O}(n^2)$. □

Untere Schranke: Wird als Pivot-Element immer das kleinste (oder größte) Element gewählt, so ist die gesamte Anzahl an Vergleichen

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \in \mathcal{O}(n^2)$$

Satz 8.2. *Wird in QuickSort als Pivot-Element der Median von S gewählt, so ist die Rekursionstiefe nur $\mathcal{O}(\log n)$ für $n = |S|$.*

Beweis. Die Größe der Teilmenge halbiert sich mit jedem Level des Rekursionsbaums. Daher gibt es nur $\mathcal{O}(\log n)$ Level. □

Problem: Wie kann der Median von S effizient berechnet werden?

- Später: Berechnung des Medians in $\mathcal{O}(n)$
- Es gibt also theoretische QuickSort-Variante mit Laufzeit $\Theta(n \log n)$
- Diese Variante ist jedoch empirisch deutlich ineffizienter

Satz 8.3. *Wählt man das Pivot-Element zufällig und gleichverteilt aus S, so ist die erwartete Laufzeit von QuickSort in $\mathcal{O}(n \log n)$ für $n = |S|$.*

Beweis. Es sei $f(n)$ Erwartungswert der Laufzeit für n Elemente, $f(0) \coloneqq 0$. Dann gibt es ein $c \in \mathbb{R}_{>0}$ mit $f(1) \leq c$ und

$$f(n) \leq c \cdot n + \frac{1}{n} \sum_{i=1}^{n} \big(f(i-1) + f(n-i)\big) = c \cdot n + \frac{2}{n} \sum_{i=1}^{n-1} f(i)$$

Zeige mit vollständiger Induktion 2. Art, dass $f(n) \leq 2c \cdot n \cdot (1 + \ln n)$ für alle $n \in \mathbb{N}_{>0}$. Induktionsanfang ($n = 1$):

$$f(1) \leq c \leq 2c = 2c \cdot 1 \cdot (1 + \ln 1)$$

Induktionsschluss:

$$\begin{aligned} f(n) &\leq c \cdot n + \frac{2}{n} \sum_{i=1}^{n-1} (2c \cdot i \cdot (1 + \ln i)) \\ &\leq c \cdot n + \frac{2c}{n} \int_1^n (2x(1 + \ln x))dx \\ &= c \cdot n + \frac{2c}{n} \left[x^2 + (1 + \ln x) + \frac{x^2}{2} \right]_1^n \\ &= 2c \cdot n \cdot (1 + \ln n) - \frac{c}{n} \\ &\leq 2c \cdot n \cdot (1 + \ln n) \end{aligned}$$

□

8.11 Nützliche Eigenschaften von Sortierverfahren

	Laufzeit	stabil	in place
SelectionSort	$\Theta(n^2)$	ja	ja
InsertionSort	$\Theta(n^2)$	ja	ja
MergeSort	$\Theta(n \log n)$	ja	nein
QuickSort	$\Theta(n^2)$	nein	(ja)
Random QuickSort	$\Theta(n \log n)$	nein	(ja)

Annahme: Ein Sortierverfahren bekommt als Input ein Array. Die Elemente des Arrays besitzen jeweils einen Wert (nicht notwendig verschieden). Der Output ist ein nach Werten sortiertes Array.

Definition 8.4. (i) Ein Sortierverfahren heißt stabil, falls Elemente mit demselben Wert im Output in derselben Reihenfolge wie im Input auftauchen.

(ii) Ein Sortierverfahren arbeitet in Place, falls es zusätzlich zum Eingabearray nur konstant viel Speicher zum Zwischenspeichern der zu sortierenden Einträge benötigt.

8.12 Median und i-te geordnete Statisik

Definition 8.5. Es sei A eine Menge von n (paarweise verschiedenen) Zahlen und $1 \leq i \leq n$.

(a) Die i-te geordnete Statistik ist das i-te Element in der aufsteigenden Sortierung von A.

(b) Ist n ungerade, so nennt man die $\frac{n+1}{2}$-te geordnete Statistik auch Median von A.

(c) Ist n gerade, so nennt man die $\frac{n}{2}$-te geordnete Statistik auch (unteren) Median von A.

Algorithm 1: Das Auswahlproblem

Input: Menge A von paarweise verschiedenen Zahlen, $i \in \{1, 2, \ldots, n\}$
Output: i-te geordnete Statistik von A

Naheliegender Algorithmus:

(1) Sortiere A aufsteigend

(2) Gib das i-te Element der Sortierung zurück

Worst-Case-Laufzeit: $\Theta(n \log n)$. Kann man den Median auch in $\Theta(n)$ finden?

8.13 Divide & Conquer für das Auswahlproblem

Algorithm 2: Algorithmus in Linearzeit für das Laufzeitproblem

```
1  Auswahl(A, i):
2  wähle Pivot-Element x ∈ A
3  A_<x ← {a ∈ A | a < x}
4  A_>x ← {a ∈ A | a > x}
5  if |A_<x| = i − 1 then
6      return x
7  end
8  else if |A_<x| ≥ i then
9      return Auswahl(A_<x, i)
10 end
11 else
12     return Auswahl(A_>x, i − 1 − |A_<x|)
13 end
```

Bemerkung.
- Der Algorithmus terminiert, da die rekursiven Aufrufe nur für kleinere Mengen erfolgen, die Rekursionstiefe ist durch $n = |A|$ beschränkt.
- Der Algorithmus liefert nach Konstruktion das korrekte Ergebnis.

Beobachtung. Wählt man das Pivot-Element x in Linearzeit so, dass

$$\max\{|A_{<x}|, |A_{>x}|\} \leq \lfloor d \cdot |A| \rfloor$$