Vorlesungsmitschrieb

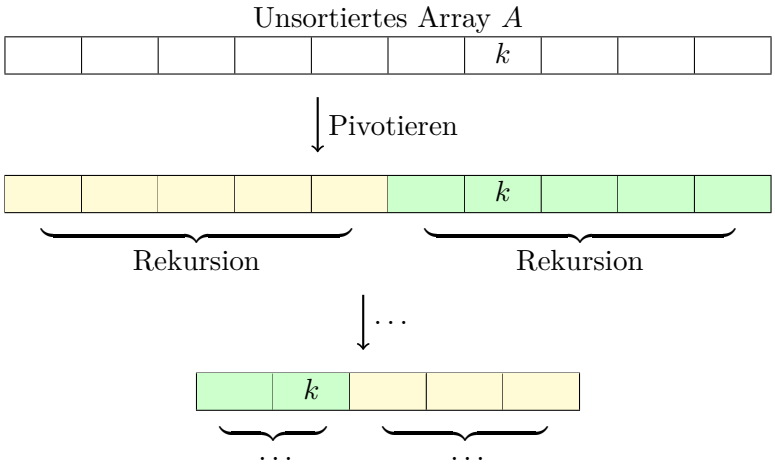
Algorithmen und Berechenbarkeit

Vorlesung 05

Letztes Update: 2017/11/18 - 12:47 Uhr

Quick-Select (Randomisiert)

Es soll in einem unsortierten Array bzw. einer unsortierten Liste A das k-kleinste Element gefunden werden. Sei A zum Beispiel $[3, 5, 1, 4, 7]$ und $k = 2$, so soll der Algorithmus den Wert 3 zurückgeben, was in der Eingabe dem k-kleinsten Wert entspricht.
Spontan und einfach zu implementieren ist vor allem diese Methode: Sortiere die Eingabedatenstruktur mit Quick- bzw. Merge-Sort und gebe $A[i]$ zurück. Bei diesem Ansatz in die Laufzeit maßgeblich vom verwendeten Sortieralgorithmus abhängig. Es fällt außerdem auf, dass eine sortierte Liste die Antwort nun für jedes k sehr schnell liefern kann. Das bedeutet, der Algorithmus ist nicht auf das Nötigste beschränkt und *macht zu viel*.
Eine einfache Optimierung erschließt sich beim Betrachten der Skizze: Ist bereits bekannt, dass sich das k-te Element nicht im betrachteten Abschnitt der Datenstruktur aufhält, so muss dieser Teil nicht sortiert werden (gelb).




```
Select(A, k):
    Wähle Pivotelement p zufällig gleichverteilt
    A_1 := Array-Teil bis Index p
    A_2 := Array-Teil ab Index p

    if k ≤ A_1.length
        Select(A_1, k)
    else if k > A_2.length
        Select(A_2, k − A_1.length)
```

Laufzeit von Quick-Select (Randomisiert)

- Fall immer der Median getroffen wird, so ergibt sich $n + \frac{1}{2} + \frac{1}{4} ... \leq 2n$
- Mit X_{ij} wie bei Quicksort: $E[X_{ij}] = \frac{2}{j-i+1}$.

  Es gibt nun die drei Fälle
  - $k < i < j$
  - $i < k < j$
  - $i < j < k$

$\Rightarrow$ Die Laufzeit ist $\mathcal{O}(n)$. Schneller ist nicht möglich, da jedes Element betrachtet werden muss.

Quick-Select (Deterministisch)

In einigen Anwendungsszenarien ist es wichtiger, dass ein Ergebnis konstant schnell vorliegt als *wahrscheinlich schnell*.
Wurde im randomisierten Algorithmus für Quick-Select die Wahl des Pivotelements dem Zufall überlassen, so soll das Pivotelement nun nach einem festen Ablauf ermittelt werden. Dabei ist es wichtig, dass das Pivotelement nicht perfekt gewählt sein muss, sondern nur *ausreichend gut*.

Wahl des Pivotelements

1. Man gruppiere die Elemente der Eingabestruktur zu je 5 Elementen.
2. Man berechnet in jeder Gruppe den Median (geht in sechs Vergleichen: $\mathcal{O}(n)$). Nun hat man $\frac{n}{5}$ Gruppenmediane ($M[i] :=$ Median der Gruppe i).
3. Man berechnet **rekursiv** den Median der Gruppenmediane: `Select(`$M, \frac{\frac{n}{5}}{2}$`)`
4. Der Median der Gruppenmediane wird als Pivotelement verwendet

Pivotelement gut genug

- Sei M der Median der Gruppenmediane. Von den $\frac{n}{5}$ Gruppenmedianen sind die Hälfte ($\frac{n}{5} : 2 = \frac{n}{10}$) größer bzw. kleiner als M.
- In jeder Gruppe, deren Gruppenmedian $< M$ ist, sind 3 Elemente $< M$ bzw. in jeder Gruppe, deren Gruppenmedian $> M$ ist, sind 3 Elemente $> M$.
- Zusammen ergibt das

  $\rightarrow\ 3 \cdot \frac{n}{10}$ Elemente $< M$ oder eben

  $\rightarrow\ 3 \cdot \frac{n}{10}$ Elemente $> M$

$\Rightarrow$ Wählt man also nach dieser Methode das Pivotelement, so erhält man mindestens einen $30\% - 70\%$ Split.

Pivotelement schnell genug gefunden

Für die Laufzeit ergibt sich

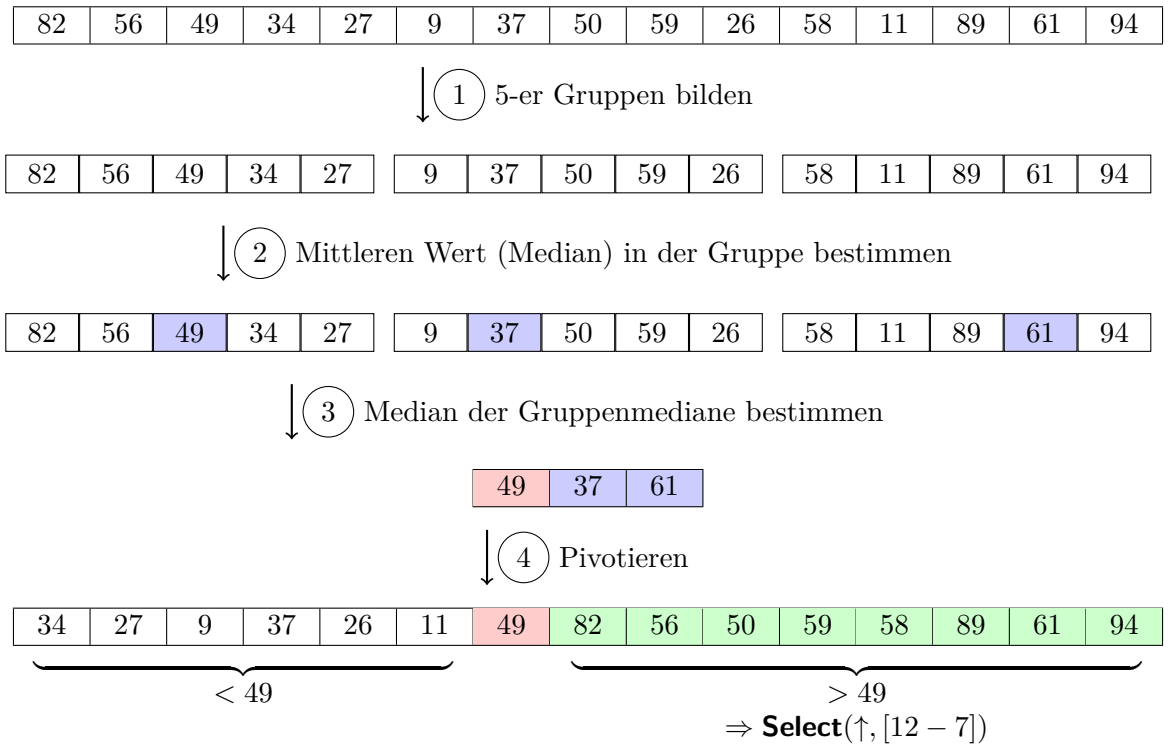
$$T(n) \leq \underbrace{T(\frac{n}{5})}_{\substack{\text{Median}\\ \text{der}\\ \text{Mediane}}} + \underbrace{T(\frac{7}{10} \cdot n)}_{\substack{\text{Quick-Sort-}\\ \text{Rekursion}}} + \underbrace{\Theta(n)}_{\substack{\text{Pivotieren}\\ +\\ \text{Verwaltung}}}$$

Per Induktion kann nun gezeigt werden $T(n) \leq \gamma \cdot n$. Im Induktionsschritt zeigt sich dann

$$\begin{aligned} T(n) &\overset{\text{IV}}{\leq} \gamma \cdot \frac{n}{5} + \gamma \cdot \frac{7}{10} \cdot n + c \cdot n \\ &= n \cdot (\frac{9}{10} \cdot \gamma + c) \\ &\leq \gamma \cdot n \end{aligned}$$

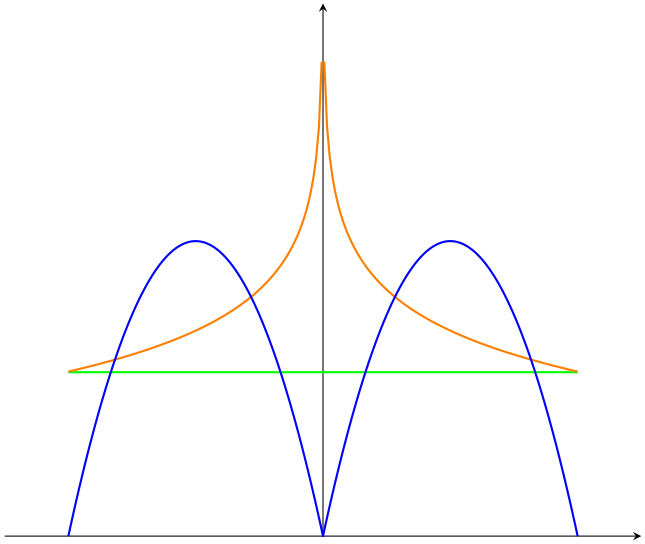
Dies ist der Fall für $\gamma = 10 \cdot c \qquad \Longrightarrow T(n) \in \mathcal{O}(n)$
Nebeneffekt: Der Median wird in Linearzeit berechnet (im Beispiel des Quick-Sort-Pivots).

Beispiel: Select (A, 12)



Concentration Bounds

Bisher wurden hauptsächlich Analysen mithilfe des Erwartungswerts durchgeführt. Der Erwartungswert macht jedoch keine Aussage über die Verteilung.



Man kann die Verteilung eines Algorithmus mit Erwartungswert L generisch analysieren: Der Algorithmus läuft $\alpha \cdot L$ Schritte. Ist er danach nicht fertig, wird er neugestartet.

$\rightarrow$ Die Wahrscheinlichkeit, in einem Durchlauf nicht fertig zu werden, ist $< \frac{1}{\alpha}$.

$\rightarrow$ Die Wahrscheinlichkeit, in k Durchläufen nicht fertig zu werden, ist $\leq \frac{1}{\alpha} \cdot \frac{1}{\alpha} \cdot \ ... = \frac{1}{\alpha^k}$.

Beispiel 1

Gegeben ist begrenzte Zeit $t \underbrace{>>}_{\substack{\text{viel} \\ \text{größer}}} L$

- Wie soll α gewählt werden?
- Für gegebenes α können maximal $\frac{t}{\alpha \cdot l}$ Runden durchlaufen werden.

Beispiel 2

- Seien $\alpha = 2$ und $k = \log(n)$: Dann beträgt die erwartete Laufzeit $\frac{1}{n}$.
- Seien $\alpha = 2$ und $k = 2 \cdot \log(n)$: Dann beträgt die erwartete Laufzeit $\frac{1}{n^2}$.

  Für den zweiten Fall liefert Markov:

$$P(\text{Algorithmus braucht länger als } 2 \cdot \alpha \cdot L \cdot \log(n)) \leq \frac{1}{2 \cdot \alpha \cdot \log(n)}$$