Tutorium Algorithmen 1

03 · Suchen und Sortieren · 6.5.2024

Peter Bohner Tutorium 3

Binäre Suche

Karlsruher Institut für Technologie

Gegeben ein sortiertes Array A

Ist 42 in A?

Nein!

?	?	?	?	?	?	?	?	33	35	36	39	44	56	?	?	?	?

Peter Bohner - peter.bohner@student.kit.edu

Binäre Suche

Problem:

Wir möchten *schnell* nach einem Element in einem *sortierten* Array suchen

Idee:

Wir beginnen in der Mitte des Arrays und halbieren unseren Suchraum. Wiederhole auf dem neuen Teilarray.

```
find(A, x):
    if x < Element in der Mitte von A then
        find(linke Hälfte von A, x)
    else
        find(rechte Hälfte von A, x)
```

Binäre Suche funktioniert nur auf sortierten Datenstrukturen mit Random Access!!!

Peter Bohner - peter.bohner@student.kit.edu

Aufgabe: k-Sum

Gegeben sei ein Array A natürlicher Zahlen und ein k $\in \mathbb{N}$

1. Beschreibe einen Algorithmus mit Laufzeit $\mathcal{O}(n^2)$, der zwei Indizes i, j mit $A[i]+A[j] = k$ bestimmt.

Sei nun A sortiert. Bestimme nun einen Algorithmus mit einer Laufzeit in

2. $\mathcal{O}(n \log n)$.
3. **Bonus:** $\mathcal{O}$(n)

Peter Bohner - peter.bohner@student.kit.edu

Lösung: k-Sum

1. Probiere jede Kombination von i und j.
   Das erste Paar mit $A[i] + A[j] = k$ ist eine Lösung.
2. Suche für alle i in A nach dem Index j mit $A[j] = k - A[i]$. Verwende dafür binäre Suche.
   Ein i, für das ein j gefunden wird, ist mit dem j eine Lösung.

Lösung: *k*-Sum

3. Beschreibung

- Nutze zwei Pointer, die von außen nach innen wandern
- Die Elemente, auf die sie zeigen, ergeben die Lösungskandidaten
- Kandidat zu klein $\Rightarrow$ linken Pointer nach rechts schieben
- Kandidat zu groß $\Rightarrow$ rechten Pointer nach links schieben
- Summe = k $\Rightarrow$ Lösung gefunden
- Pointer treffen sich $\Rightarrow$ es gibt keine Lösung

Korrektheit:

- Zu jedem Zeitpunkt wissen wir, dass das Lösungspaar (wenn vorhanden) zwischen den Zeigern liegt.
- Wenn der Kandidat zu groß ist, kann das rechteste Element nicht zur Lösung gehören, weil es selbst mit dem kleinsten Element zusammen zu groß ist.
- Analog, wenn der Kandidat zu klein ist.

Peter Bohner - peter.bohner@student.kit.edu

Mergesort

MERGESORT(A):

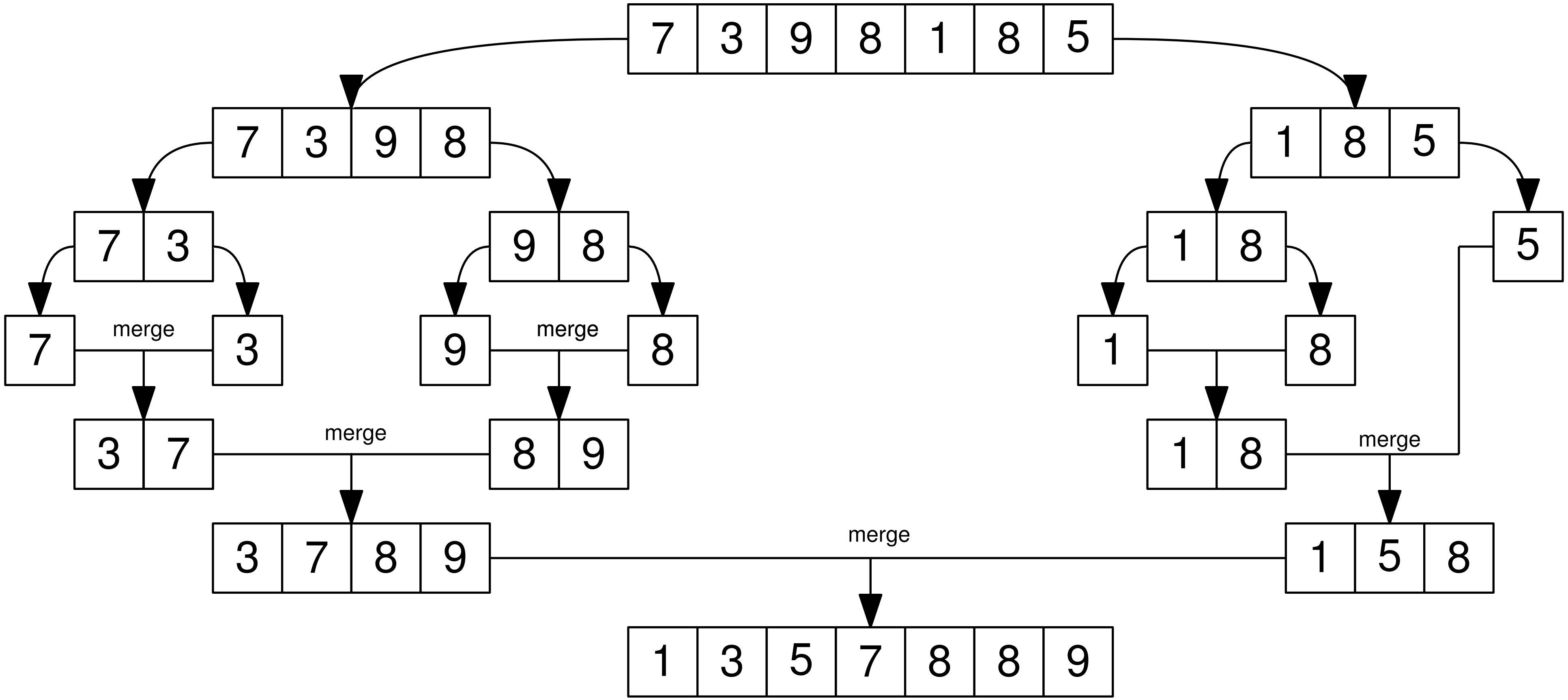
```
if A.size() ≤ 1 then return A
else
    sortierteErsteHälfte := Mergesort(erste hälfte von A)
    sortierteZweiteHälfte := Mergesort(zweite hälfte von A)
return merge(sortierteErsteHälfte, sortierteZweiteHälfte)
```

Aufgabe

Sortiere die Folge $\langle 7, 3, 9, 8, 1, 8, 5 \rangle$ mit Mergesort.

Peter Bohner - peter.bohner@student.kit.edu

Mergesort



Peter Bohner - peter.bohner@student.kit.edu

Mergesort

MERGESORT(A):

```
if A.size() ≤ 1 then return A
else
    sortierteErsteHälfte := Mergesort(erste hälfte von A)
    sortierteZweiteHälfte := Mergesort(zweite hälfte von A)
return mergesortierteErsteHälfte, sortierteZweiteHälfte
```

- Welche Bedingung müssen die Parameter von **merge** erfüllen?
- Wie schnell ist **merge**?
- Was ist die Laufzeit von Mergesort im Best-Case?
- Was ist die Laufzeit von Mergesort im Worst-Case?

Quicksort

Karlsruher Institut für Technologie

$\textsc{Quicksort}(A)$:

---

if A.**size()** ≤ 1 **then return** A
else
  Wähle pivot $\in A$ mit Magie
  $A_< :=$ Elemente $a_i \in A$ mit $a_i <$pivot
  $A_= :=$ Elemente $a_i \in A$ mit $a_i =$pivot
  $A_> :=$ Elemente $a_i \in A$ mit $a_i >$pivot
  return Quicksort($A_<$) $\cdot A_= \cdot$ **Quicksort($A_>$)**

Aufgabe

Sortiere die Folge $\langle 7, 3, 9, 8, 1, 8, 5 \rangle$ mit Quicksort.

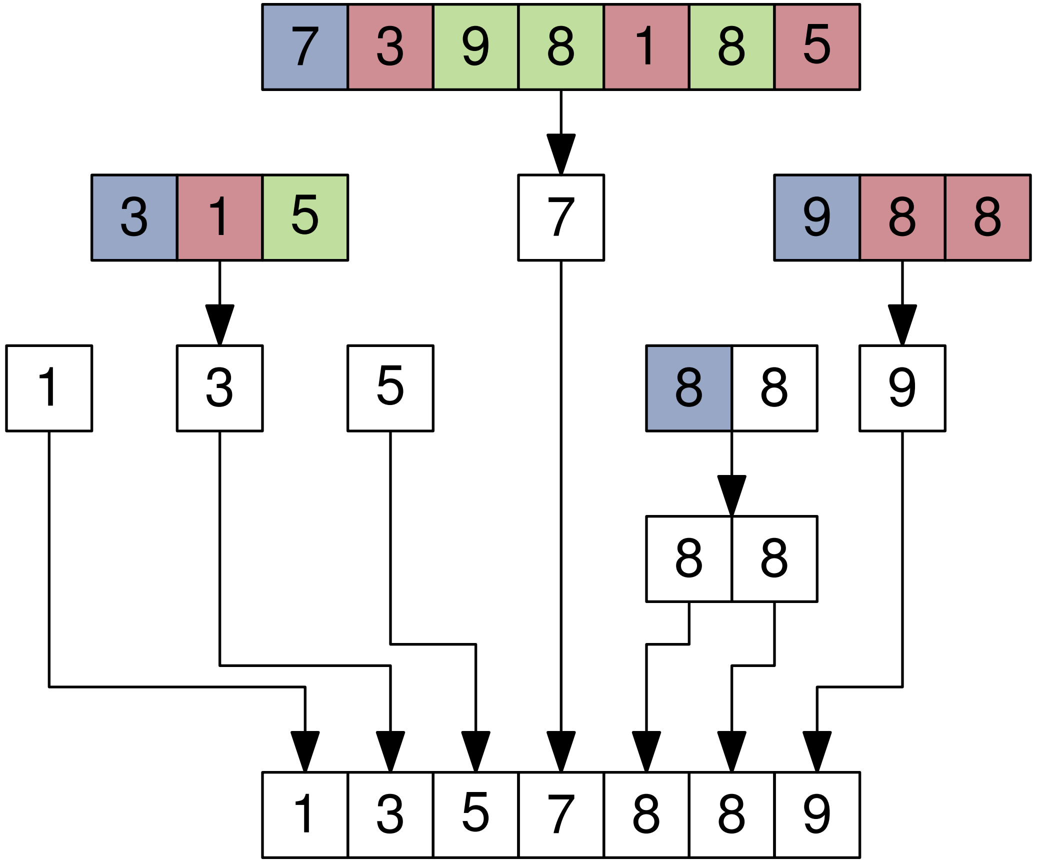
Peter Bohner - peter.bohner@student.kit.edu

Quicksort

Pivot

$<$ Pivot

$>$ Pivot



Quicksort

QUICKSORT(A):

if A**.size()** ≤ 1 **then return** A
else
  Wähle pivot $\in A$ mit Magie
  $A_<$ $:=$ Elemente $a_i \in A$ mit $a_i <$pivot
  $A_=$ $:=$ Elemente $a_i \in A$ mit $a_i =$pivot
  $A_>$ $:=$ Elemente $a_i \in A$ mit $a_i >$pivot
  return Quicksort($A_<$**)** $\cdot A_= \cdot$ **Quicksort(**$A_>$**)**

- Warum funktioniert Quicksort?
- Wovon hängt die Laufzeit ab?
- Was sind besonders schlechte Pivots?
- Was sind besonders gute Pivots?
- Was ist die LZ von Quicksort im Worst-Case?
- Was ist die LZ von Quicksort im Best-Case?

Aufgabe: Verschiedene Sortieralgorithmen

Betrachtet folgende Sortieralgorithmen:

- Mergesort
- Quicksort (immer linkestes Element als Pivot gewählt)
- Insertion Sort

Welche Laufzeit haben die Sortieralgorithmen auf einer...

- streng steigenden Folge
- konstanten Folge (alle Elemente gleich)
- streng fallenden Folge

Bonus: Was kennt ihr noch für (lustige) Sortieralgorithmen?

Peter Bohner - peter.bohner@student.kit.edu

Aufgabe: Verschiedene Sortieralgorithmen

	Mergesort	Quicksort	Insertion Sort
steigend	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n)$
konstant	$\Theta(n \log n)$	$\Theta(n)$	$\Theta(n)$
fallend	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$

Peter Bohner - peter.bohner@student.kit.edu

Aufgabe: Autoverleih

Du bist Manager eines Autoverleihs, der Fahrzeuge in $k \in \mathbb{N}$ Fahrzeugtypen besitzt. Vom Fahrzeugtyp $i \in \mathbb{Z}_k$ sind c_i Fahrzeuge vorhanden. Dir liegen die nächsten n Buchungen, jeweils mit Start- und Endzeitpunkt sowie Fahrzeugtyp, vor.

Finde einen Algorithmus, der in $\mathcal{O}(n \log n + k)$ Zeit entscheidet, ob allen Buchungen entsprochen werden kann.

- Teile die Buchungen in zwei Events auf, eins für die Ausleihe und eins für die Rückgabe
- Sortiere die Events nach Zeitpunkt, bei gleichem Zeitpunkt wird Rückgabe vor Ausleihe sortiert
- Erstelle ein Array für die aktuelle Zahl verfügbarer Autos pro Fahrzeugtyp, initialisiert mit c
- Betrachte die Events in Reihenfolge, verringere die Zahl verfügbarer Autos eines Typs bei einer Ausleihe, erhöhe sie bei einer Rückgabe
- Gibt es keinen Zeitpunkt, zu dem eine negative Anzahl eines Fahrzeugtyps vorhanden sind, kann allen Buchungen entsprochen werden

Peter Bohner - peter.bohner@student.kit.edu

Aufgabe: Ternary Search

- Gegeben:
  - Array der Größe n
  - Existenz von Index $i_{min} \in \{0, \dots, n-1\}$ mit:
    $A[i] > A[j]$ für $i < j \leq i_{min}$
    $A[i] < A[j]$ für $i_{min} \leq i < j$
- i_{min} nicht bekannt
- Aufgabe:
  - Gib einen möglichst effizienten Algorithmus an, der das Minimum in A, also $\min_{i \in \{0,\dots,n-1\}} A[i]$ ausgibt
  - Begründe dessen Korrektheit
  - Bestimme die asymptotische Laufzeit deines Algorithmus

Tipps:
- Das ist in $o(n)$ möglich
- Kann man vielleicht ähnlich wie bei binärer Suche vorgehen?
- Betrachte die Einträge an zwei verschiedenen Indizes und versuche die Position des Minimums einzugrenzen

Peter Bohner - peter.bohner@student.kit.edu

Lösung: Ternary Search

- Wähle zwei Indizes $i_0 < i_1$, die das Array (bis auf Runden) dritteln
- Vergleiche $A[i_0]$ mit $A[i_1]$
- Schließe das Drittel auf der Seite des größeren Eintrages aus, bzw. beide wenn $A[i_0] = A[i_1]$
- Führe Algorithmus für verbleibendes Teilarray aus
- Wenn Größe des Teilarrays ≤ 3: bestimme das Minimum der verbleibenden Einträge und gib dieses aus

Ternary Search – Korrektheit

- $A[i_{min}]$ ist das gesuchte Minimum
- i_{min} ist immer im betrachteten Teilarray:
- Zu Beginn: ganzes Array wird betrachtet
- In jedem Schritt gilt:
  - Falls $A[i_0] < A[i_1]$: Minimum kann nicht im rechten Drittel sein, da sonst $i_0 < i_1 \leq i_{min}$ aber $A[i_0] \not> [i_1]$ gelten würde
    $\implies i_{min} < i_1$
  - Falls $A[i_0] > A[i_1]$: Minimum kann nicht im linken Drittel sein, da sonst $i_{min} \leq i_0 < i_1$ aber $A[i_0] \not< A[i_1]$ gelten würde
    $\Rightarrow i_{min} > i_0$
  - Falls $A[i_0] = A[i_1]$: Minimum muss im mittleren Drittel sein, da sonst $i_0 < i_1 \leq i_{min}$ aber $A[i_0] \not> A[i_1]$ oder $i_{min} \leq i_0 < i_1$ aber $A[i_0] \not< A[i_1]$ gelten würde
    $\implies i_0 < i_{min} < i_1$

  $\Rightarrow$ in jedem Fall liegt i_{min} auch im nächsten Teilarray
- Das betrachtete Teilarray wird in jedem Schritt kleiner $\Rightarrow$ der Algorithmus terminiert
- Das Minimum $A[i_{min}]$ liegt im letzten Teilarray und wird daher ausgegeben

Peter Bohner - peter.bohner@student.kit.edu

Ternary Search – Laufzeit

- In jedem Schritt gilt: das als nächstes betrachtete Teilarray ist höchstens $\frac{2}{3}$ mal so groß wie das gerade betrachtete Teilarray
- Für die Anzahl der Schritte s gilt also: $n\left(\frac{2}{3}\right)^s \geq 1 \Leftrightarrow n \geq \left(\frac{3}{2}\right)^s \Leftrightarrow \log_{1,5}(n) \geq s$
  $\Rightarrow s \in \mathcal{O}(() \log n)$
- Konstante Laufzeit pro Schritt
- Daher: Laufzeit in $\mathcal{O}(\log n)$

Peter Bohner - peter.bohner@student.kit.edu

Zusammenfassung

Was haben wir heute gemacht?

- Binäre Suche
- Sortieren

Worauf könnt ihr euch nächste Woche freuen?

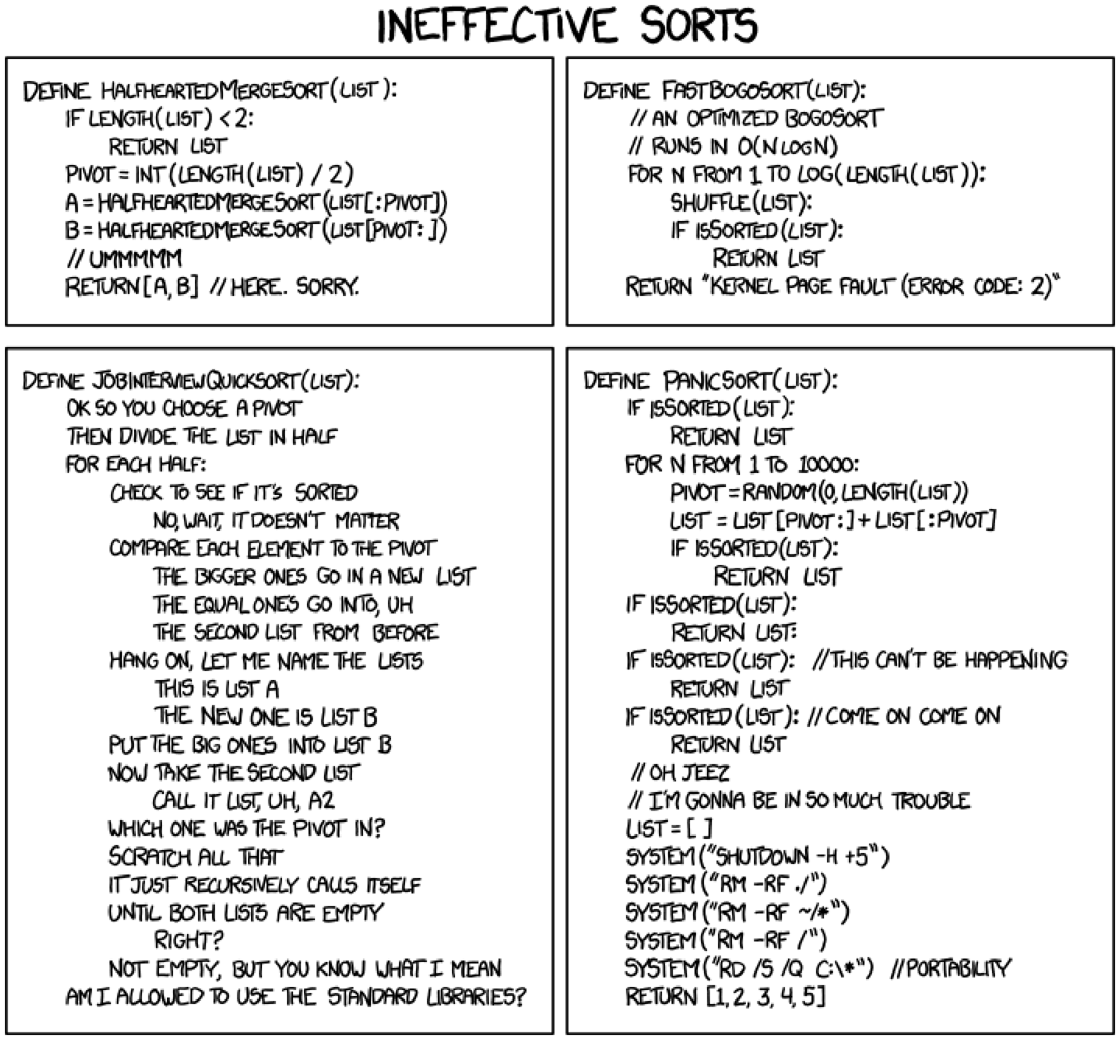
- Ganzzahliges Sortieren

Peter Bohner - peter.bohner@student.kit.edu

Fragen?

Fragen!

Ende



https://xkcd.com/1667/