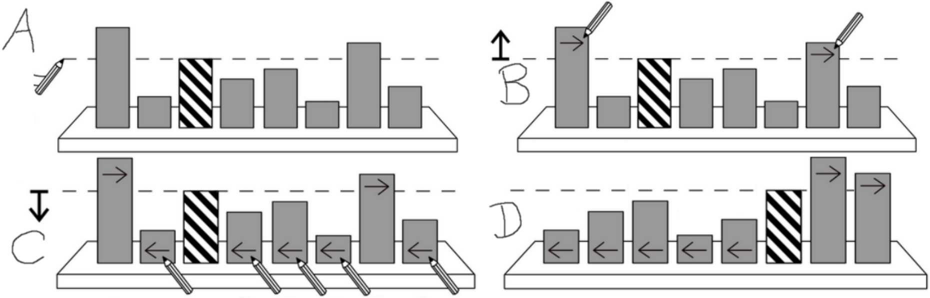
Algorithmen I
Tutorium 33

Woche 6 | 01. Juni 2018

Daniel Jungkind (daniel.jungkind@student.kit.edu)

INSTITUT FÜR THEORETISCHE INFORMATIK



Inhalt

Quicksort

Bucketsort

Zu Blatt #5

Durchschnitt: etwa 72 % der Punkte

mergesOrt läuft in $\Theta(|R|)$. **?**

Zu Blatt #5

Durchschnitt: etwa 72 % der Punkte

mergesOrt läuft in $\Theta(|R|)$. **Falsch.**

In $\Theta(n \cdot |R|)$ bzw. $\Theta(n \log |R|)$
(je nach Implementierung und für $n := |A|$).
Es macht aber $\mathcal{O}(|R|)$ **Rekursionsabstiege**!

Quiz

MergeSort läuft im Best-Case in $\mathcal{O}(n)$. **?**

Quiz

MergeSort läuft im Best-Case in $\mathcal{O}(n)$. **Falsch.** **Immer** in $\Theta(n \log n)$.

Quiz

MergeSort läuft im Best-Case in $\mathcal{O}(n)$. **Falsch.** **Immer** in $\Theta(n \log n)$.

MergeSort sortiert stabil. **?**

Quiz

MergeSort läuft im Best-Case in $\mathcal{O}(n)$. **Falsch.** **Immer** in $\Theta(n \log n)$.

MergeSort sortiert stabil. **Wahr.**

Quiz

MergeSort läuft im Best-Case in $\mathcal{O}(n)$. **Falsch.** **Immer** in $\Theta(n \log n)$.

MergeSort sortiert stabil. **Wahr.**

Wir können nicht schneller als $\Theta(n \log n)$ sortieren. **?**

Quiz

MergeSort läuft im Best-Case in $\mathcal{O}(n)$. **Falsch.** **Immer** in $\Theta(n \log n)$.

MergeSort sortiert stabil. **Wahr.**

Wir können nicht schneller als $\Theta(n \log n)$ sortieren. **Falsch.**

Vergleichsbasiert ja. Gibt aber auch noch andere Sortierverfahren (Stay tuned!)

Quiz

MergeSort läuft im Best-Case in $\mathcal{O}(n)$. **Falsch.** **Immer** in $\Theta(n \log n)$.

MergeSort sortiert stabil. **Wahr.**

Wir können nicht schneller als $\Theta(n \log n)$ sortieren. **Falsch.**

Vergleichsbasiert ja. Gibt aber auch noch andere Sortierverfahren (Stay tuned!)

Bei Hashtabellen ist der Hashwert $h(e)$ der Key eines Elements. **?**

Quiz

MergeSort läuft im Best-Case in $\mathcal{O}(n)$. **Falsch.** **Immer** in $\Theta(n \log n)$.

MergeSort sortiert stabil. **Wahr.**

Wir können nicht schneller als $\Theta(n \log n)$ sortieren. **Falsch.**

Vergleichsbasiert ja. Gibt aber auch noch andere Sortierverfahren (Stay tuned!)

Bei Hashtabellen ist der Hashwert $h(e)$ der Key eines Elements. **Falsch.**

Quiz

MergeSort läuft im Best-Case in $\mathcal{O}(n)$. **Falsch.** **Immer** in $\Theta(n \log n)$.

MergeSort sortiert stabil. **Wahr.**

Wir können nicht schneller als $\Theta(n \log n)$ sortieren. **Falsch.**

Vergleichsbasiert ja. Gibt aber auch noch andere Sortierverfahren (Stay tuned!)

Bei Hashtabellen ist der Hashwert $h(e)$ der Key eines Elements. **Falsch.**

Eine Familie von Hashfunktionen ist universell, falls für alle $x \neq y$ gilt: $\mathbb{P}_h[h(x) = h(y)] = \frac{1}{m}$. **?**

Quiz

MergeSort läuft im Best-Case in $\mathcal{O}(n)$. **Falsch.** **Immer** in $\Theta(n \log n)$.

MergeSort sortiert stabil. **Wahr.**

Wir können nicht schneller als $\Theta(n \log n)$ sortieren. **Falsch.**

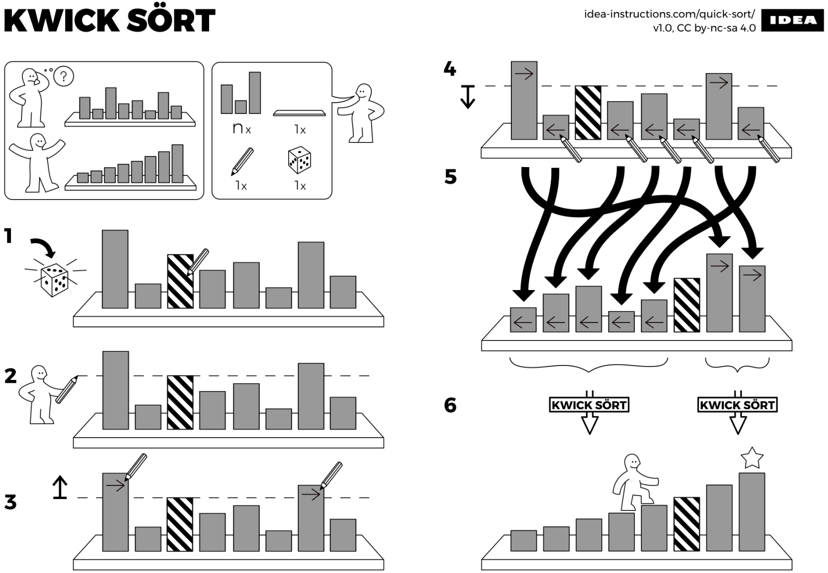
Vergleichsbasiert ja. Gibt aber auch noch andere Sortierverfahren (Stay tuned!)

Bei Hashtabellen ist der Hashwert $h(e)$ der Key eines Elements. **Falsch.**

Eine Familie von Hashfunktionen ist universell, falls für alle $x \neq y$ gilt: $\mathbb{P}_h[h(x) = h(y)] = \frac{1}{m}$. **Wahr.**

QUICKSORT

Eine erquickende Neuerung


KWICK SÖRT
idea-instructions.com/quick-sort/
v1.0, CC by-nc-sa 4.0
IDEA
n x
1x
1x
1x
1
2
3
4
5
6
KWICK SÖRT
KWICK SÖRT

Quicksort

- Erinnerung: Array sortierbar durch Einteilung in sortierten und unsortierten Bereich

$\Rightarrow$ **Idee: „Semi-Sortierung“**
Wähle beliebiges Pivotelement p (in $O(1)$) und teile auf (in $O(n)$):

$\leq p$	$> p$

Diese Teile dann rekursiv weitersortieren.

Quicksort

- Erinnerung: Array sortierbar durch Einteilung in sortierten und unsortierten Bereich

$\Rightarrow$ **Idee: „Semi-Sortierung“**
Wähle beliebiges Pivotelement p (in $O(1)$) und teile auf (in $O(n)$):

$\leqslant p$	$> p$

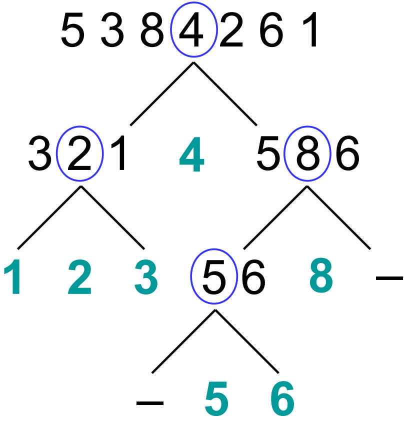
Diese Teile dann rekursiv weitersortieren.

Quicksort – Beispiel

Sortiere $A = (5, 3, 8, 4, 2, 6, 1) :$ **array**$[1..n]$ **of** $\mathbb{N}$ mit Quicksort. Wähle als Pivot $p(A) := A[\lceil \frac{n}{2} \rceil]$. Zeichne dazu den Rekursionsbaum.

Quicksort – Beispiel

Sortiere $A = (5, 3, 8, 4, 2, 6, 1)$: **array**$[1..n]$ **of** $\mathbb{N}$ mit Quicksort. Wähle als Pivot $p(A) := A[\lceil \frac{n}{2} \rceil]$. Zeichne dazu den Rekursionsbaum.



Quicksort (mit partition)

- Wie effizient und platzsparend aufteilen?
  $\Rightarrow$ **partition**! $O(1)$ Platz und $O(n)$ Zeit
  (Siehe nächste Folien...)
- **Laufzeit**: Master-Theorem nicht anwendbar, da Größe der rekursiven Aufrufe **nicht** in Voraus bekannt
- Worst-Case $\Theta(n^2)$ möglich
- Vorlesung sagt: **Erwartete** Laufzeit in $\Theta(n \log n)$

Quicksort (mit partition)

- Wie effizient und platzsparend aufteilen?
  $\Rightarrow$ **partition**! $O(1)$ Platz und $O(n)$ Zeit
  (Siehe nächste Folien...)
- **Laufzeit**: Master-Theorem nicht anwendbar, da Größe der rekursiven Aufrufe **nicht** in Voraus bekannt
- Worst-Case $\Theta(n^2)$ möglich
- Vorlesung sagt: **Erwartete** Laufzeit in $\Theta(n \log n)$

Quicksort (mit partition)

Karlsruher Institut für Technologie

- Wie effizient und platzsparend aufteilen?
  $\Rightarrow$ **partition**! $O(1)$ Platz und $O(n)$ Zeit
  (Siehe nächste Folien...)
- **Laufzeit**: Master-Theorem nicht anwendbar, da Größe der rekursiven Aufrufe **nicht** in Voraus bekannt
- Worst-Case $\Theta(n^2)$ möglich
- Vorlesung sagt: **Erwartete** Laufzeit in $\Theta(n \log n)$

Quicksort (mit partition)

- Wie effizient und platzsparend aufteilen?
  $\Rightarrow$ **partition**! $O(1)$ Platz und $O(n)$ Zeit
  (Siehe nächste Folien...)
- **Laufzeit**: Master-Theorem nicht anwendbar, da Größe der rekursiven Aufrufe **nicht** in Voraus bekannt
- Worst-Case $\Theta(n^2)$ möglich
- Vorlesung sagt: **Erwartete** Laufzeit in $\Theta(n \log n)$

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

1	8	6	9	1	7	0

Hier klicken, um das Beispiel zu überspringen.

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

		p				
1	8	6	9	1	7	0

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

						p
1	8	0	9	1	7	6

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

						p
1	8	0	9	1	7	6

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$\leqslant p$						p
1	8	0	9	1	7	6

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$\leqslant p$						p
1	8	0	9	1	7	6

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$\leqslant p$	$> p$					p
1	8	0	9	1	7	6

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$\leqslant p$	$> p$					p
1	8	0	9	1	7	6

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$\leqslant p$		$> p$				p
1	0	8	9	1	7	6

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$\leqslant p$		$> p$				p
1	0	8	9	1	7	6

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$\leqslant p$		$> p$				p
1	0	8	9	1	7	6

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$\leqslant p$		$> p$				p
1	0	8	9	1	7	6

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$\leqslant p$			$> p$			p
1	0	1	9	8	7	6

Quicksort (mit partition)

Karlsruher Institut für Technologie

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$\leqslant p$			$> p$			p
1	0	1	9	8	7	6

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$\leqslant p$			$> p$			p
1	0	1	9	8	7	6

Quicksort (mit partition)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$\leqslant p$			p	$> p$		
1	0	1	6	8	7	9

Quicksort (mit partition)

Schema aus der Vorlesung

Beispiel: Partitionierung, $k = 1$

p, $\bar{i}$, $\underline{j}$

3	6	8	1	0	7	2	4	5	9
$\overline{\underline{9}}$	6	8	1	0	7	2	4	5	3
$\overline{9}$	$\underline{6}$	8	1	0	7	2	4	5	3
$\overline{9}$	6	$\underline{8}$	1	0	7	2	4	5	3
$\overline{9}$	6	8	$\underline{1}$	0	7	2	4	5	3
1	$\overline{6}$	8	9	$\underline{0}$	7	2	4	5	3
1	0	$\overline{8}$	9	6	$\underline{7}$	2	4	5	3
1	0	$\overline{8}$	9	6	7	$\underline{2}$	4	5	3
1	0	2	$\overline{9}$	6	7	8	$\underline{4}$	5	3
1	0	2	$\overline{9}$	6	7	8	4	$\underline{5}$	3
1	0	2	$\overline{9}$	6	7	8	4	5	3
1	0	2	3	6	7	8	4	5	9

Quicksort (mit partition)

Laufzeit, wenn alle Zahlen gleich sind?

$\Rightarrow \Theta(n^2)$

Quicksort (mit partition)

Laufzeit, wenn alle Zahlen gleich sind?
$\Rightarrow \Theta(n^2)$

Quicksort (mit partition)

Laufzeit, wenn alle Zahlen gleich sind?
$\Rightarrow \Theta(n^2)$

Quicksort (mit partition) ist stabil. **?**

Quicksort (mit partition)

Laufzeit, wenn alle Zahlen gleich sind?
$\Rightarrow \Theta(n^2)$

Quicksort (mit partition) ist stabil. **Falsch.**

„Durcheinandermischen" bei partition macht's kaputt.

Quicksort (mit partition)

Laufzeit, wenn alle Zahlen gleich sind?
$\Rightarrow \Theta(n^2)$

Quicksort (mit partition) ist stabil. **Falsch.**

„Durcheinandermischen“ bei partition macht’s kaputt.

Quicksort ist in-place. **?**

Quicksort (mit partition)

Laufzeit, wenn alle Zahlen gleich sind?
$\Rightarrow \Theta(n^2)$

Quicksort (mit partition) ist stabil. **Falsch.**

„Durcheinandermischen“ bei partition macht’s kaputt.

Quicksort ist in-place. **Je nachdem!**

Rekursionsaufrufe benötigen $\Theta(\log n)$ (vernachlässigbar) viel Platz ($\Rightarrow$ Stack-Overhead). Abgesehen davon **kein** weiterer Verwaltungsaufwand.

Quicksort (besseres partition)

Aller guten Dinge sind drei!

- Worst-Case von eben: schlecht ☹

⇒ Besser: **Drei-Wege-Partitionierung**!

- Führe einen zusätzlichen Bereich $= p$ ein:

$< p$	$= p$	$> p$

Quicksort (besseres partition)

Aller guten Dinge sind drei!

- Worst-Case von eben: schlecht ☹

$\Rightarrow$ Besser: **Drei-Wege-Partitionierung**!

- Führe einen zusätzlichen Bereich $= p$ ein:

$< p$	$= p$	$> p$

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

3	7	0	5	1	5	5	8	1

Hier klicken, um das Beispiel zu überspringen.

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

3	7	0	5	1	5	5	8	1

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

								p
3	7	0	1	1	5	5	8	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

								p
3	7	0	1	1	5	5	8	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$								p
3	7	0	1	1	5	5	8	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$								p
3	7	0	1	1	5	5	8	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$	$> p$							p
3	7	0	1	1	5	5	8	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$	$> p$							p
3	7	0	1	1	5	5	8	5

Quicksort (mit Drei-Wege-Partitionierung)

Karlsruher Institut für Technologie

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$		$> p$						p
3	0	7	1	1	5	5	8	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$		$> p$						p
3	0	7	1	1	5	5	8	5

Quicksort (mit Drei-Wege-Partitionierung)

Karlsruher Institut für Technologie

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$			$> p$					p
3	0	1	7	1	5	5	8	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$			$> p$					p
3	0	1	7	1	5	5	8	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$				$> p$				p
3	0	1	1	7	5	5	8	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$				$> p$				p
3	0	1	1	7	5	5	8	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$				$> p$			$= p$	
3	0	1	1	7	8	5	5	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$				$> p$			$= p$	
3	0	1	1	7	8	5	5	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$				$> p$			$= p$	
3	0	1	1	7	8	5	5	5

Quicksort (mit Drei-Wege-Partitionierung)

Karlsruher Institut für Technologie

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$				$> p$			$= p$	
3	0	1	1	7	8	5	5	5

Quicksort (mit Drei-Wege-Partitionierung)

Karlsruher Institut für Technologie

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$				$> p$		$= p$		
3	0	1	1	7	8	5	5	5

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$					$= p$			
3	0	1	1	7	5	5	5	8

Quicksort (mit Drei-Wege-Partitionierung)

Beispiel

Partitioniere A : **array**$[0..n-1]$ mit Pivotwahl $p(A) := A[\lfloor \frac{n}{3} \rfloor]$ (mit $n := |A|$)

$< p$				$= p$			$> p$	
3	0	1	1	5	5	5	7	8

Quicksort (mit Drei-Wege-Partitionierung)

Laufzeit, wenn alle Elemente gleich sind?

$\Rightarrow \Theta(n)$

Quicksort (mit Drei-Wege-Partitionierung)

Laufzeit, wenn alle Elemente gleich sind?
$\Rightarrow \Theta(n)$

Quicksort

Auf Listen

- Wie müsste man vorgehen, um Quicksort auf einfach verketteten Listen anzuwenden (ohne die Liste in ein Array umzuwandeln)?

  $\Rightarrow$ Wähle als Pivot $p := \mathit{head.next}$.
  partition: Laufe durch die Liste und teile Elemente auf zwei Listen $\ell_\leq$ und $\ell_>$ auf.
  Sortiere rekursiv $\ell_\leq$ und $\ell_>$ und verbinde sie anschließend.

- Wie leicht lässt sich hierbei ein Worst-Case erreichen? Womit könnte man das vermeiden?

  $\Rightarrow$ Wegen eingeschränkter Pivot-Wahl:
  Schon fast sortiert $\rightsquigarrow$ Worst-Case
  $\Rightarrow$ Viele gleiche Elemente $\rightsquigarrow$ Drei-Wege-Partition!
  $\Rightarrow$ Generell: Auf verketteten Listen lieber **Mergesort**.

Quicksort

Auf Listen

- Wie müsste man vorgehen, um Quicksort auf einfach verketteten Listen anzuwenden (ohne die Liste in ein Array umzuwandeln)?
  $\Rightarrow$ Wähle als Pivot $p := \mathit{head}.\mathit{next}$.
  partition: Laufe durch die Liste und teile Elemente auf zwei Listen $\ell_{\leqslant}$ und $\ell_{>}$ auf.
  Sortiere rekursiv $\ell_{\leqslant}$ und $\ell_{>}$ und verbinde sie anschließend.
- Wie leicht lässt sich hierbei ein Worst-Case erreichen? Womit könnte man das vermeiden?
  $\Rightarrow$ Wegen eingeschränkter Pivot-Wahl:
  Schon fast sortiert $\rightsquigarrow$ Worst-Case
  $\Rightarrow$ Viele gleiche Elemente $\rightsquigarrow$ Drei-Wege-Partition!
  $\Rightarrow$ Generell: Auf verketteten Listen lieber **Mergesort**.

Quicksort

Auf Listen

- Wie müsste man vorgehen, um Quicksort auf einfach verketteten Listen anzuwenden (ohne die Liste in ein Array umzuwandeln)?
  $\Rightarrow$ Wähle als Pivot $p := head.next$.
  partition: Laufe durch die Liste und teile Elemente auf zwei Listen $\ell_\leqslant$ und $\ell_>$ auf.
  Sortiere rekursiv $\ell_\leqslant$ und $\ell_>$ und verbinde sie anschließend.
- Wie leicht lässt sich hierbei ein Worst-Case erreichen? Womit könnte man das vermeiden?
  $\Rightarrow$ Wegen eingeschränkter Pivot-Wahl:
  Schon fast sortiert $\rightsquigarrow$ Worst-Case
  $\Rightarrow$ Viele gleiche Elemente $\rightsquigarrow$ Drei-Wege-Partition!
  $\Rightarrow$ Generell: Auf verketteten Listen lieber **Mergesort**.

Quicksort

...nicht-rekursiv?

- Wie könnte eine iterative Implementierung von Quicksort aussehen?
  $\Rightarrow$ Speichere „Rekursionsparameter" als Tupel (ℓ, r) auf einem **Stack**, welcher mit einer „großen Schleife" abgearbeitet wird (*faked recursion*)
  Eine Queue ginge in *diesem* Fall (!) auch, wäre halt nicht ganz so intuitiv.
- Was wären mögliche Vorteile/Nachteile?
  $+$ Rekursive Aufrufe werden durch einen platzsparenderen Ersatz gespeichert
  $-$ Implementierungsaufwand: Echte Rekursion ist hübscher :P

Quicksort

...nicht-rekursiv?

- Wie könnte eine iterative Implementierung von Quicksort aussehen?
  $\Rightarrow$ Speichere „Rekursionsparameter" als Tupel (ℓ, r) auf einem **Stack**, welcher mit einer „großen Schleife" abgearbeitet wird (*faked recursion*)

  Eine Queue ginge in *diesem* Fall (!) auch, wäre halt nicht ganz so intuitiv.
- Was wären mögliche Vorteile/Nachteile?
  $+$ Rekursive Aufrufe werden durch einen platzsparenderen Ersatz gespeichert

  $-$ Implementierungsaufwand: Echte Rekursion ist hübscher :P

Quicksort

...nicht-rekursiv?

- Wie könnte eine iterative Implementierung von Quicksort aussehen?
  $\Rightarrow$ Speichere „Rekursionsparameter“ als Tupel (ℓ, r) auf einem **Stack**, welcher mit einer „großen Schleife“ abgearbeitet wird (*faked recursion*)

  Eine Queue ginge in *diesem* Fall (!) auch, wäre halt nicht ganz so intuitiv.
- Was wären mögliche Vorteile/Nachteile?
  + Rekursive Aufrufe werden durch einen platzsparenderen Ersatz gespeichert
  − Implementierungsaufwand: Echte Rekursion ist hübscher :P

Quicksort

... vs. InsertionSort

- Bei „ausreichend kleinen“ Bereichen wird üblicherweise statt einem Rekursionsaufruf *InsertionSort* verwendet. Warum?

$\Rightarrow$ + Quicksort gut auf **größeren** Arrays: Vertauschen einzelner Elemente **billiger** als ganze Bereiche verschieben
− Quicksort bürokratisch ($O(n^2)$) auf **kleineren** Arrays: Zu viel Vertauschen + Rekursionsoverhead.

$\Rightarrow$ *InsertionSort* auf kleinen Arrays linear: „Kurze“ Strecken zum Einsortieren.

Quicksort

... vs. InsertionSort

- Bei „ausreichend kleinen“ Bereichen wird üblicherweise statt einem Rekursionsaufruf *InsertionSort* verwendet. Warum?

$\Rightarrow$ Quicksort gut auf **größeren** Arrays: Vertauschen einzelner Elemente **billiger** als ganze Bereiche verschieben

Quicksort bürokratisch ($O(n^2)$) auf **kleineren** Arrays: Zu viel Vertauschen $+$ Rekursionsoverhead.

$\Rightarrow$ *InsertionSort* auf kleinen Arrays linear: „Kurze“ Strecken zum Einsortieren.

Quicksort

... vs. InsertionSort

- Bei „ausreichend kleinen“ Bereichen wird üblicherweise statt einem Rekursionsaufruf *InsertionSort* verwendet. Warum?

$\Rightarrow$ + Quicksort gut auf **größeren** Arrays: Vertauschen einzelner Elemente **billiger** als ganze Bereiche verschieben
− Quicksort bürokratisch ($O(n^2)$) auf **kleineren** Arrays: Zu viel Vertauschen $+$ Rekursionsoverhead.

$\Rightarrow$ *InsertionSort* auf kleinen Arrays linear: „Kurze“ Strecken zum Einsortieren.

Sortieralgorithmen – Showdown

	Mergesort	Quicksort
In-place?		
Ablauf		
Stabil?		
Laufzeit		
Cache-...		

Sortieralgorithmen – Showdown

	Mergesort	Quicksort
In-place?	Nur auf verketteten Listen*	**Ja***
Ablauf		
Stabil?		
Laufzeit		
Cache-...		

* abgesehen vom Verwaltungsoverhead durch Rekursion

Sortieralgorithmen – Showdown

	Mergesort	Quicksort
In-place?	Nur auf verketteten Listen*	**Ja***
Ablauf	Zuerst Rekursion, danach linearer Aufwand**	Zuerst linearer Aufwand, danach Rekursion
Stabil?		
Laufzeit		
Cache-...		

* abgesehen vom Verwaltungsoverhead durch Rekursion

** abgesehen von Listenzertrennung in linearer Zeit (zur Mitte muss gelaufen werden)

Sortieralgorithmen – Showdown

Karlsruher Institut für Technologie

	Mergesort	Quicksort
In-place?	Nur auf verketteten Listen*	**Ja***
Ablauf	Zuerst Rekursion, danach linearer Aufwand**	Zuerst linearer Aufwand, danach Rekursion
Stabil?	Möglich	**Mit Partition**: Nein (nicht in-place: Möglich)
Laufzeit		
Cache-...		

* abgesehen vom Verwaltungsoverhead durch Rekursion

** abgesehen von Listenzertrennung in linearer Zeit (zur Mitte muss gelaufen werden)

Sortieralgorithmen – Showdown

	Mergesort	Quicksort
In-place?	Nur auf verketteten Listen*	**Ja***
Ablauf	Zuerst Rekursion, danach linearer Aufwand**	Zuerst linearer Aufwand, danach Rekursion
Stabil?	Möglich	**Mit Partition**: Nein (nicht in-place: Möglich)
Laufzeit	**garantiert** in $\Theta(n \log n)$	**erwartet** in $\Theta(n \log n)$ Worst-Case $\Theta(n^2)$
Cache-...		

* abgesehen vom Verwaltungsoverhead durch Rekursion

** abgesehen von Listenzertrennung in linearer Zeit
(zur Mitte muss gelaufen werden)

Sortieralgorithmen – Showdown

	Mergesort	Quicksort
In-place?	Nur auf verketteten Listen*	**Ja***
Ablauf	Zuerst Rekursion, danach linearer Aufwand**	Zuerst linearer Aufwand, danach Rekursion
Stabil?	Möglich	**Mit Partition**: Nein (nicht in-place: Möglich)
Laufzeit	**garantiert** in $\Theta(n \log n)$	**erwartet** in $\Theta(n \log n)$ Worst-Case $\Theta(n^2)$
Cache-...	unfreundlich	freundlich

* abgesehen vom Verwaltungsoverhead durch Rekursion

** abgesehen von Listenzertrennung in linearer Zeit (zur Mitte muss gelaufen werden)

Sortieralgorithmen – Showdown

	Mergesort	Quicksort
In-place?	Nur auf verketteten Listen*	**Ja***
Ablauf	Zuerst Rekursion, danach linearer Aufwand**	Zuerst linearer Aufwand, danach Rekursion
Stabil?	Möglich	**Mit Partition**: Nein (nicht in-place: Möglich)
Laufzeit	**garantiert** in $\Theta(n \log n)$	**erwartet** in $\Theta(n \log n)$ Worst-Case $\Theta(n^2)$
Cache-...	unfreundlich	freundlich
	Hat einen sprechenden Namen	Heißt Quicksort, muss also gut sein

* abgesehen vom Verwaltungsoverhead durch Rekursion

** abgesehen von Listenzertrennung in linearer Zeit (zur Mitte muss gelaufen werden)

BUCKETSORT

„Hashing für Arme“

Bucketsort

Alles im Eimer?

- n Elemente **beschränkter** Größe (also $\forall e: \ e \in \{a, ..., b\}$)
- Lege an buckets : array[a..b] of List of Element
- Schmeiße jedes Element e in seinen Eimer: buckets[e].pushBack(e) (hinten anhängen)
- Am **Ende**: „Eimer" zusammenhängen

$\Rightarrow$ Array sortiert.

- **Laufzeit**: $O(n+k)$
- **Aufpassen** bei großen/unbeschränkten k!

Bucketsort

Alles im Eimer?

- n Elemente **beschränkter** Größe (also $\forall e: \; e \in \{a, ..., b\}$)
- Lege an $\;$ buckets : **array**$[a..b]$ **of** List **of** Element
- Schmeiße jedes Element e in seinen Eimer: buckets$[e]$.pushBack(e) (hinten anhängen)
- Am **Ende**: „Eimer“ zusammenhängen

$\Rightarrow$ Array sortiert.

- **Laufzeit**: $O(n+k)$
- **Aufpassen** bei großen/unbeschränkten k!

Bucketsort

Alles im Eimer?

- n Elemente **beschränkter** Größe (also $\forall e: \; e \in \{a, ..., b\}$)
- Lege an $\;\text{buckets} : \textbf{array}[a..b] \textbf{ of } \text{List} \textbf{ of } \text{Element}$
- Schmeiße jedes Element e in seinen Eimer: $\text{buckets}[e].\text{pushBack}(e)$ (hinten anhängen)
- Am **Ende**: „Eimer“ zusammenhängen

$\Rightarrow$ Array sortiert.

- **Laufzeit**: $O(n + k)$
- **Aufpassen** bei großen/unbeschränkten k!

Bucketsort

Alles im Eimer?

- n Elemente **beschränkter** Größe (also $\forall e: \; e \in \{a, ..., b\}$)
- Lege an buckets : **array**$[a..b]$ **of** List **of** Element $\quad (k := |\text{buckets}|)$
- Schmeiße jedes Element e in seinen Eimer: buckets$[e]$.pushBack(e) (hinten anhängen)
- Am **Ende**: „Eimer" zusammenhängen

$\Rightarrow$ Array sortiert.

- **Laufzeit**: $O(n + k)$
- **Aufpassen** bei großen/unbeschränkten k!

Bucketsort

Karlsruher Institut für Technologie

Generisches Bucketsort

- Eimer nicht unbedingt **Listen**
- Eimer nicht unbedingt nur für **eine Größe**
  $\Rightarrow$ **Intervalle** möglich
  $\Rightarrow$ In diesem Fall: Buckets müssen am Ende noch **sortiert** werden! (Z. B. mit *InsertionSort*)
  $\Rightarrow$ Dafür empfehlenswert: Elemente **gleichverteilt** auf Buckets

$\Rightarrow$ Bucketsort ist **kein** Sortieralgorithmus für sich, sondern eine **Familie** von Sortieralgorithmen.

Bucketsort

Aufgabe 1: Bucket, bucket Kuchen

Sortiert folgende Liste mit Bucketsort:

$\langle 36, 78, 50, 1, 92, 15, 43, 99, 64 \rangle$.

Verwendet dabei 5 Buckets in den Intervallen:

0 bis 19, 20 bis 39, 40 bis 59, 60 bis 79 und 80 bis 99.

Bucketsort

Lösung zu Aufgabe 1

0–19	20–39	40–59	60–79	80–99
$\langle 1, 15 \rangle$	$\langle 36 \rangle$	$\langle 50, 43 \rangle$	$\langle 78, 64 \rangle$	$\langle 92, 99 \rangle$

$\Rightarrow \langle 1, 15, 36, 43, 50, 64, 78, 92, 99 \rangle$

„Nerd’s Heaven“

Rumänische Volkstänze FTW!

- InsertionSort: https://www.youtube.com/watch?v=ROalU379l3U
- SelectionSort: https://www.youtube.com/watch?v=Ns4TPTC8whw
- Mergesort: https://www.youtube.com/watch?v=XaqR3G_NVoo
- Quicksort: https://www.youtube.com/watch?v=ywWBy6J5gz8

… u. v. m.

„Nerd’s Heaven“

Rumänische Volkstänze FTW!

- InsertionSort: https://www.youtube.com/watch?v=ROalU379l3U
- SelectionSort: https://www.youtube.com/watch?v=Ns4TPTC8whw
- Mergesort: https://www.youtube.com/watch?v=XaqR3G_NVoo
- Quicksort: https://www.youtube.com/watch?v=ywWBy6J5gz8

... u. v. m.

„Nerd’s Heaven“

Rumänische Volkstänze FTW!

- InsertionSort:
  https://www.youtube.com/watch?v=ROalU379l3U
- SelectionSort:
  https://www.youtube.com/watch?v=Ns4TPTC8whw
- Mergesort:
  https://www.youtube.com/watch?v=XaqR3G_NVoo
- Quicksort:
  https://www.youtube.com/watch?v=ywWBy6J5gz8

... u. v. m.

Danke für die Aufmerksamkeit! ☺

http://xkcd.com/1185