Algorithmen 1 SS 2013 – Tutorium 7

5. Tutorium

Sarah Lutteropp

21. Mai 2013

Übersicht

3. Übungsblatt

Abschreiben ...

Gibt 0 Punkte.

Zeitmanagement ...

Fangt rechtzeitig an, das Blatt zu machen. Gibt erwartet mehr Punkte.

Aufgabe 2.b

Darauf achten, dass die Fallunterscheidung komplett ist!
$a \in \{a\}$, aber **nicht** $a \in a$!

3. Übungsblatt

Möchte jemand vorrechnen?

Das allgemeine Sortierproblem

Allgemeine Definition des Sortierproblems

- Eingabe: Folge von n Zahlen $< a_1, a_2, \ldots, a_n >$
- Ausgabe: Permutation (Umordnung) $< a'_1, a'_2, \ldots, a'_n >$, sodass $a'_1 \leq a'_2 \leq \cdots \leq a'_n$ gilt
- Zu sortierende Zahlen sind sogenannte Schlüssel

Eigenschaften von Sortieralgorithmen

Stabil

Algorithmus behält Reihenfolge der Schlüssel bei, wenn diese gleich sind.

In-place

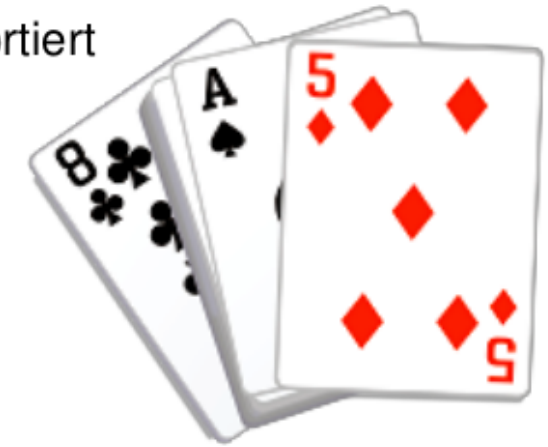
Der Algorithmus benötigt zum Sortieren keinen zusätzlichen Speicherplatz.

Insertion Sort

"Sortieren durch Einfügen"

Analog zum Einsortieren von Spielkarten in einer Hand

- Anfangen mit leerer Hand
- Schrittweises Aufnehmen und Einfügen der nächsten Karte
- Korrekte Einfügeposition wird durch Vergleichen der bereits aufgenommenen Karten von rechts nach links ermittelt
- Karten auf der Hand sind zu jedem Zeitpunkt sortiert



Insertion Sort

```
INSERTIONSORT(A)
1 for j = 2 to A.länge
2     schlüssel = A[j]
3     // Füge A[j] in sortierte Sequenz A[1..j − 1] ein
4     i = j - 1
5     while i > 0 und A[i] > schlüssel
6         A[i + 1] = A[i]
7         i = i - 1
8     A[i + 1] = schlüssel
```

Aktuell zu sortierender Schlüssel

Suche korrekte Einfügeposition

Verschiebe größere Schlüssel nach hinten

Füge Schlüssel an korrekter Position ein

Best-Case: $\mathcal{O}(n)$
Worst-Case: $\mathcal{O}(n^2)$
Average-Case: $\mathcal{O}(n^2)$

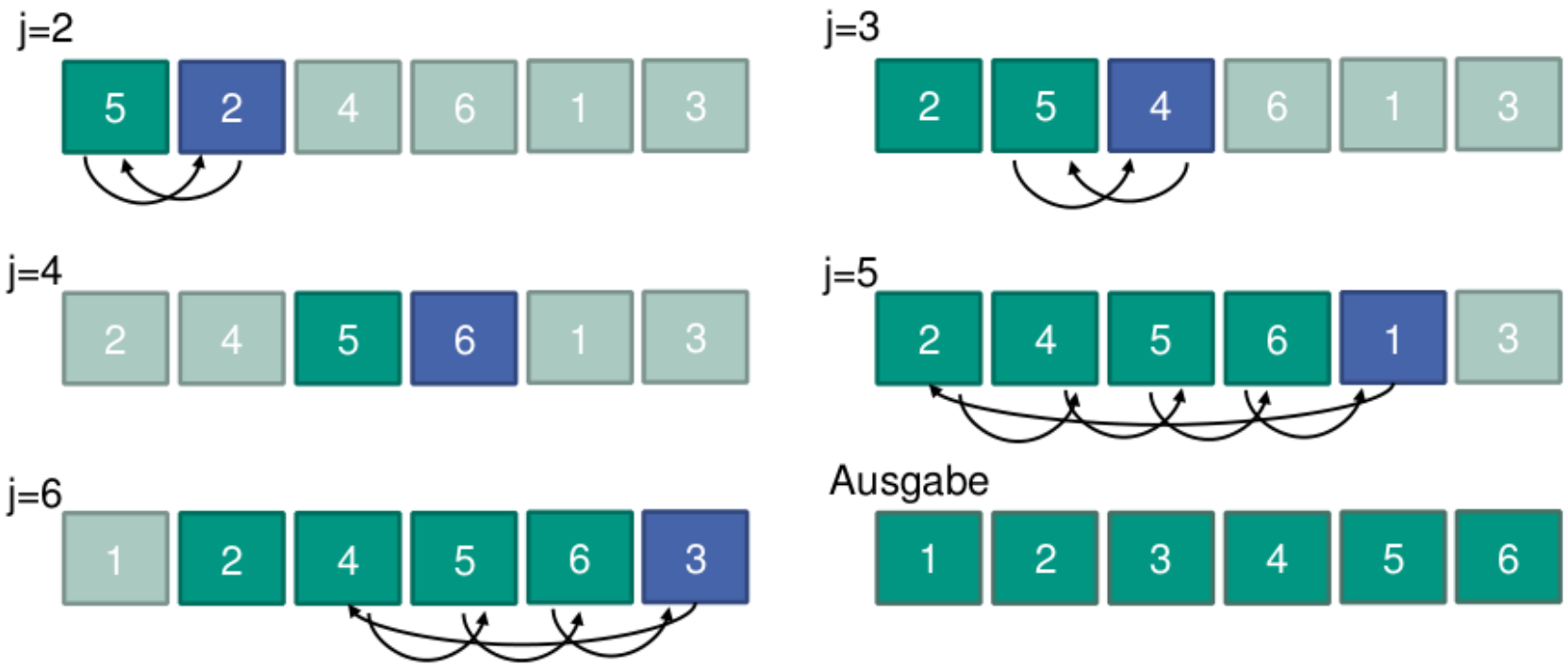
Aufgabe

Insertion Sort

Sortiere die Zahlenfolge 5,2,4,6,1,3 mit Insertion Sort.

Insertion Sort

Beispiel



Exkurs: Selection Sort

"Sortieren durch Auswählen"

Grundlegender Ablauf

- Suche das Minimum in Folge $<a_1, a_2, \ldots, a_n>$, also das Element mit dem kleinsten Sortierschlüssel
- Vertausche dieses Minimum mit dem ersten, unsortierten Element der Folge
- Man erhält im linken Teil der Folge eine sortierte Teilfolge der Länge 1 und rechts eine unsortierte Teilfolge der Länge $n - 1$.
- Wiederhole Algorithmus für die unsortierte Teilfolge bis die sortierte Teilfolge die Länge n hat

Exkurs: Selection Sort

```
SELECTIONSORT(A)
1    n = A.länge
2    links = 1
3    repeat
4        min = links
5        for i = links + 1 to n
6            if A[i] < A[min]
7                min = i
8        temp = A[min]
9        A[min] = A[links]
10       A[links] = temp
11       links = links + 1
10   until links ≥ n
```

Initialisierung des Algorithmus

Starte Minimussuche

Bestimme neues Minimum durch Vergleich mit aktuellem Minimum

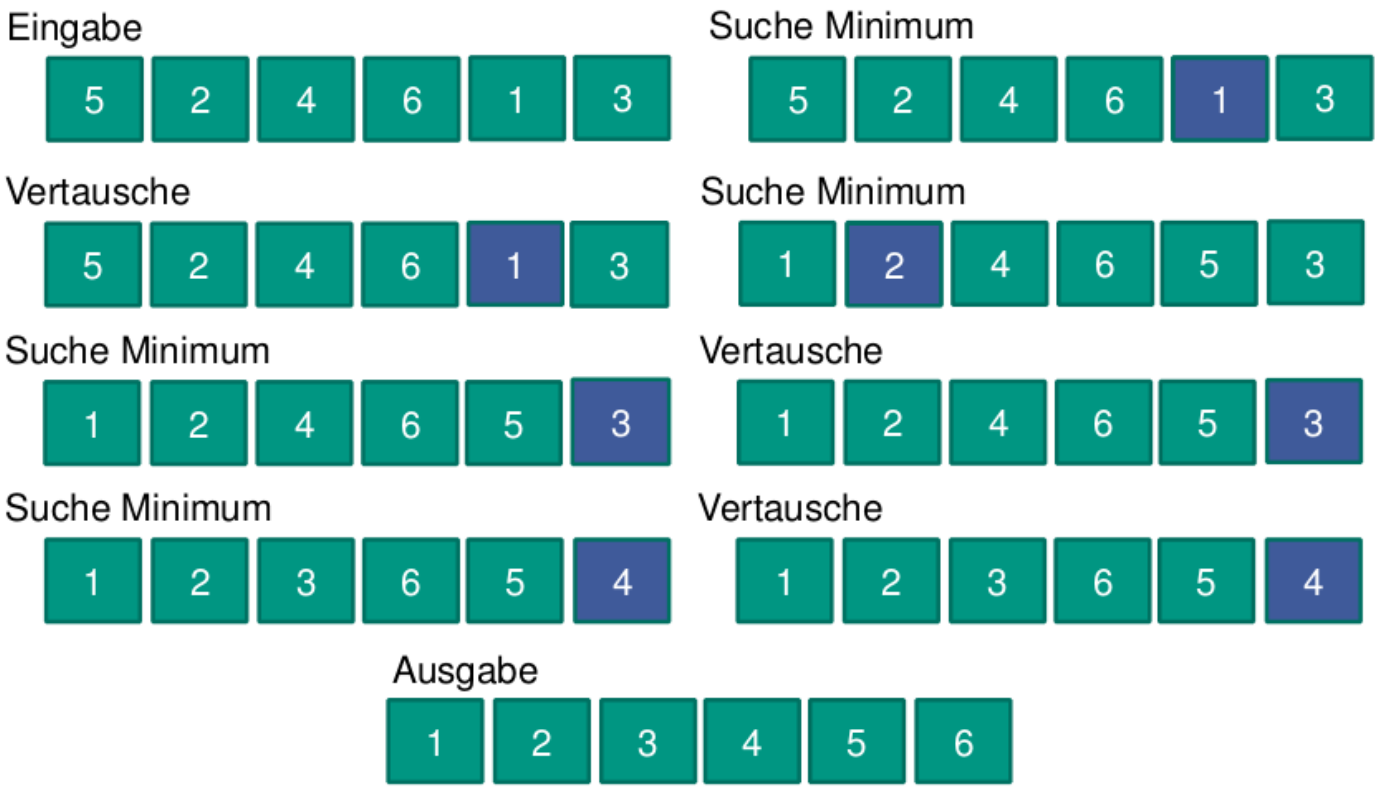
Vertausche Minimum mit erstem Element

Wiederhole dies für alle Elemente

Best-Case, Worst-Case, Average-Case: $\mathcal{O}(n^2)$

Exkurs: Selection Sort

Beispiel



Untere Schranke für das Sortierproblem

Für jeden vergleichenden Sortieralgorithmus sind $\Omega(n \log n)$ Operationen erforderlich.

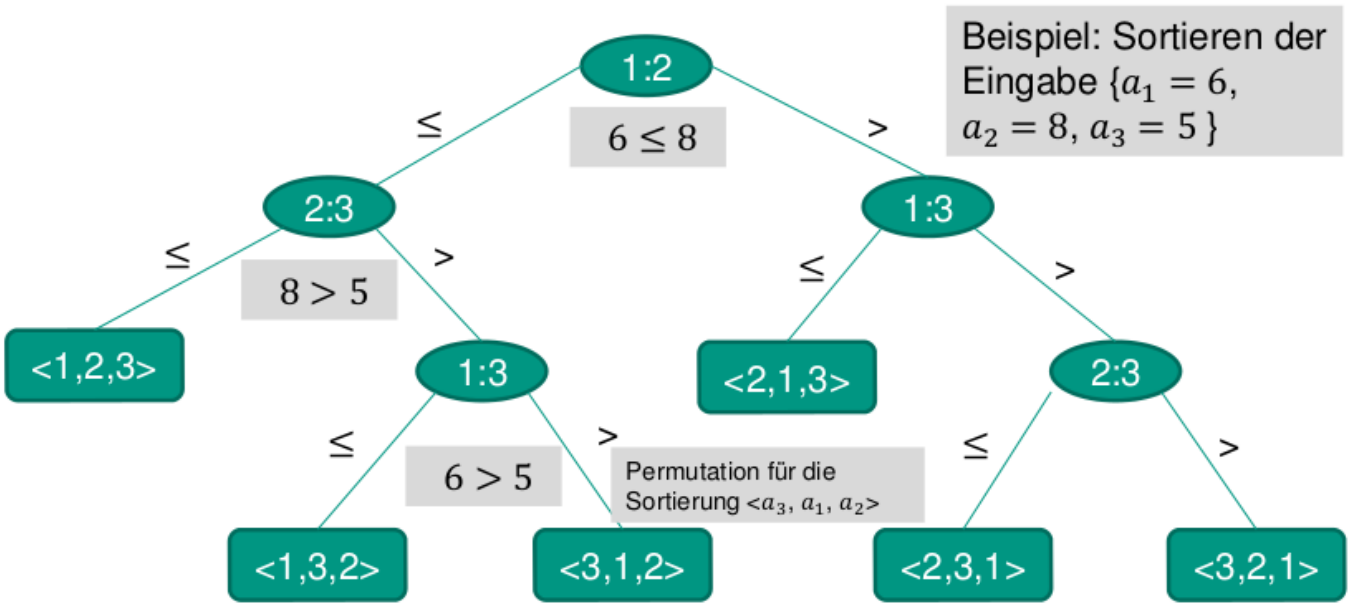
Vergleichsbasierte Sortieralgorithmen

- Annahme: Eingabe der Form $< a_1, a_2, \ldots, a_n >$
- Um relative Reihenfolge festzustellen, Vergleich zweier Elemente der Form $a_i < a_j$, $a_i > a_j$, $a_i == a_j$, $a_i \leq a_j$ oder $a_i \geq a_j$
  - Werte werden nicht betrachtet

Untere Schranke für das Sortierproblem

Für jeden vergleichenden Sortieralgorithmus sind $\Omega(n \log n)$ Operationen erforderlich.

Entscheidungsbaum



$n! \leq 2^h \Rightarrow h \geq \log(n!) = \Omega(n \log n)$

Mergesort

“Sortieren durch Mischen”

Analog zum mischen eines Stapels von Karten

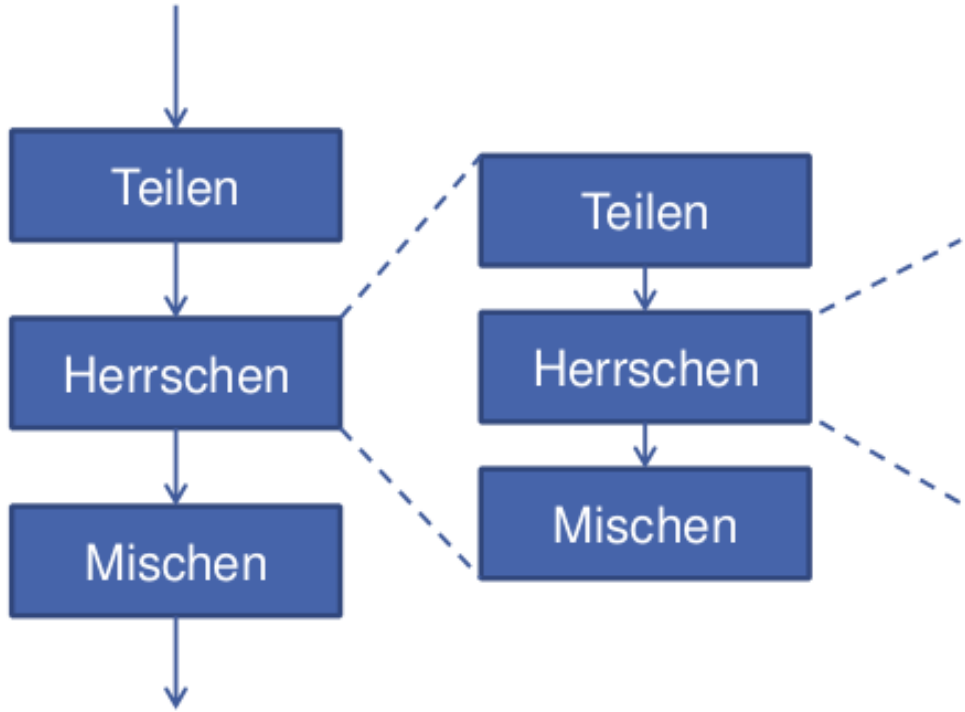
- Besteht der Stapel nur aus einer Karte, sind wir fertig
- Ansonsten teile den Stapel in zwei Hälften und sortiere diese rekursiv
- Sobald die Teile sortiert sind, mische die beiden Stapel gleichzeitig von oben nach unten nach dem Reißverschlussprinzip

Mergesort

"Sortieren durch Mischen"

Vorgehen

- Teile Folge A in zwei Teilfolgen A_1, A_2 mit je $n/2$ Elementen auf
- Sortiere die zwei Teilfolgen A_1, A_2 rekursiv mithilfe von *Mergesort*
- Mische die zwei sortierten Teilfolgen A'_1, A'_2 um die sortierte Lösung zu erhalten



Rekursion bricht bei Teilfolgenlänge 1 ab

Mergesort

- Sortieren des Feldes A der Größe l
  - Algorithmus in Methoden $MERGESORT$ und $MERGE$ aufgeteilt
  - $MERGESORT(A, p, r)$ sortiert alle Elemente im Feld $A[p..r]$
    - Erster Aufruf mit $MERGESORT(A, 1, l)$
  - $MERGE(A, p, q, r)$ mischt die beiden sortierten Teilfelder $A[p..q]$ und $A[q+1..r]$

- $MERGESORT(A, p = 1, r = l)$

```
1 if p < r
2     q = ⌊(p + r)/2⌋
3     MERGESORT(A, p, q)
4     MERGESORT(A, q + 1, r)
5     MERGE(A, p, q, r)
```

Mitte des Feldes q bestimmen

$MERGESORT$ rekursiv für beide Teilfelder aufrufen

$MERGE$ führt Teilfelder zusammen

Best-Case, Worst-Case, Average-Case: $\mathcal{O}(n \log n)$

Mergesort

- Sortieren des Feldes A der Größe l
  - Algorithmus in Methoden $MERGESORT$ und $MERGE$ aufgeteilt
  - $MERGESORT(A, p, r)$ sortiert alle Elemente im Feld $A[p..r]$
    - Erster Aufruf mit $MERGESORT(A, 1, l)$
  - $MERGE(A, p, q, r)$ mischt die beiden sortierten Teilfelder $A[p..q]$ und $A[q+1..r]$

- $MERGESORT(A, p = 1, r = l)$

```
1 if p < r
2     q = ⌊(p + r)/2⌋
3     MERGESORT(A, p, q)
4     MERGESORT(A, q + 1, r)
5     MERGE(A, p, q, r)
```

Mitte des Feldes q bestimmen

$MERGESORT$ rekursiv für beide Teilfelder aufrufen

$MERGE$ führt Teilfelder zusammen

Best-Case, Worst-Case, Average-Case: $\mathcal{O}(n \log n)$

Mergesort

```
MERGE(A, p, q, r)
1     n_1 = q − p + 1
2     n_2 = r − q
3     // seien L[1..n_1 + 1], R[1..n_2 + 1] zwei neue Felder
4     for i = 1 to n_1
5           L[i] = A[p + i − 1]
6     for j = 1 to n2
7           R[j] = A[q + j]
8     L[n_1 + 1] = R[n_2 + 1] = ∞
9     i = j = 1
10    for k = p to r
11          if L[i] <= R[j]
12                A[k] = L[i]
13                i = i + 1
14          else
15                A[k] = R[j]
16                j = j + 1
```

n_1, n_2 Größe der Teilfelder

Teilfelder L, R mit Daten aus Feld A füllen

Der Reihe nach die Elemente der Teilfelder L, R miteinander vergleichen

Jeweils das kleinere in das Feld A einfügen

Aufgabe

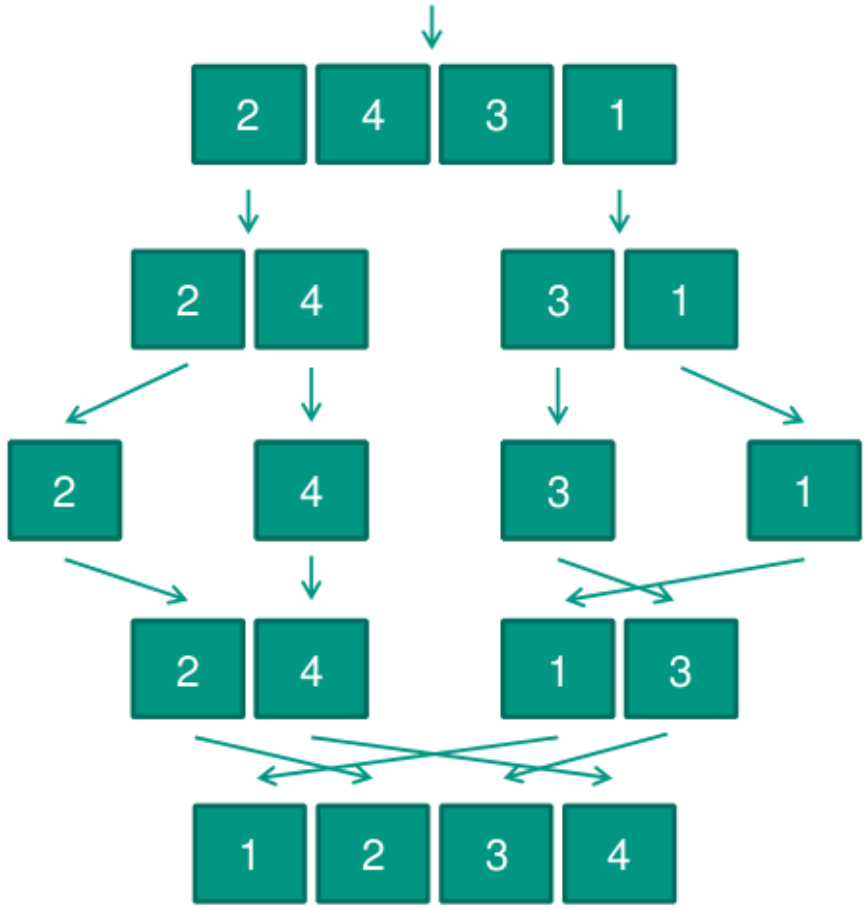
Mergesort

Sortiere die Zahlenfolge 2,4,3,1 mit Mergesort.

Mergesort

Beispiel

1. Eingabe
2. Teilen
3. Rekursiv weiter teilen
4. Mischen
5. Mischen



Quicksort

"Sortieren durch Zerlegen"

Function quickSort(s : Sequence **of** Element) : Sequence **of** Element
  if $|s| \leq 1$ **then return** s
  pick "some" $p \in s$
  $a := \langle e \in s : e < p \rangle$
  $b := \langle e \in s : e = p \rangle$
  $c := \langle e \in s : e > p \rangle$
  return concatenation of quickSort(a), b, and quickSort(c)

Best-Case, Average-Case: $\mathcal{O}(n \log n)$
Worst-Case: $\mathcal{O}(n^2)$

Heapsort

nächste Woche

Bucketsort

nächste Woche

Radixsort

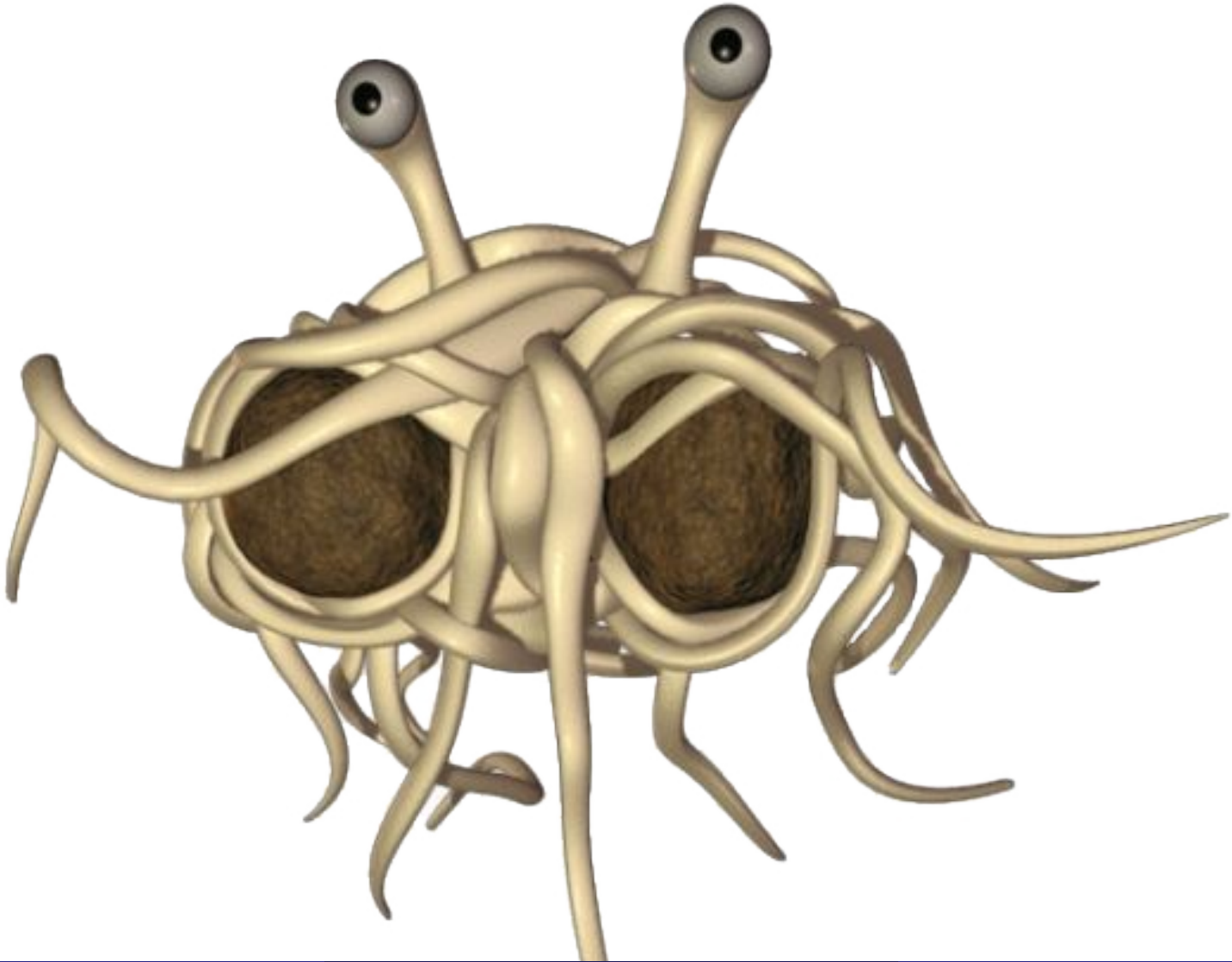
nächste Woche

Vergleich

Algorithmus	Best-Case	Worst-Case	Average-Case
Bubblesort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertionsort	$\theta(n)$	$\theta(n^2)$	$\theta(n^2)$
Selectionsort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Mergesort	$\theta(n * \lg n)$	$\theta(n * \lg n)$	$\theta(n * \lg n)$
Quicksort	$\theta(n * \lg n)$	$\theta(n^2)$	$\theta(n * \lg n)$

Welche dieser Algorithmen sind stabil? Welche sind in-place?

Aufgabe



Perzentil, Median

Definition

Gegeben: Sortierte Menge M aus n Zahlen aus $\mathbb{R}$ mit Index von 1 bis n. Das *q-Perzentil* ist das Element mit Index $\lceil q \cdot n \rceil$. Das $\frac{1}{3}$-Perzentil ist also das Element unter dem ein Drittel der Elemente der Menge liegen. Das $\frac{1}{2}$-Perzentil ist der Median.

Kreativaufgabe 

Gegeben sei ein Array mit n verschiedenen Elementen (unsortiert, aber mit Ordnung) und eine Medianfunktion, die für ein (Teil-)Array mit m Elementen den Median deterministisch in $\mathcal{O}(m)$ berechnet.

1. Schwierigkeitsstufe

Finde einen Algorithmus, der das $\frac{1}{3}$-Perzentil deterministisch in $\mathcal{O}(n)$ berechnet.

2. Schwierigkeitsstufe

Finde einen Algorithmus, der die $\frac{1}{3^{k-1}}, \frac{1}{3^{k-2}}, \ldots, \frac{1}{3}$-Perzentile deterministisch in $\mathcal{O}(n)$ berechnet. (Nicht $\mathcal{O}(nk)$!)

3. Schwierigkeitsstufe

Geht das auch inplace?

Bis zum nächsten Mal! ☺