Informatik II

Übung 3

Andreas Bärtschi, Andreea Ciuprina, Felix Friedrich, Patrick Gruntz,
Hermann Lehner, Max Rossmannek, Chris Wendler

FS 2018

Heutiges Programm

1. Feedback letzte Übung
2. Wiederholung Theorie
3. Zur aktuellen Übung

Eier werfen

- Strategie für beliebig viele Eier?

Eier werfen

- Strategie für beliebig viele Eier?
  - Binäre Suche, höchstens $\log_2 n$ Versuche.

Eier werfen

- Strategie für beliebig viele Eier?
  - Binäre Suche, höchstens $\log_2 n$ Versuche.
- Strategie mit nur einem Ei?

Eier werfen

- Strategie für beliebig viele Eier?
  - Binäre Suche, höchstens $\log_2 n$ Versuche.
- Strategie mit nur einem Ei?
  - Von unten anfangen. n Versuche.

Eier werfen

- Strategie für beliebig viele Eier?
  - Binäre Suche, höchstens $\log_2 n$ Versuche.
- Strategie mit nur einem Ei?
  - Von unten anfangen. n Versuche.
- Strategie mit zwei Eiern:

Eier werfen

- Strategie für beliebig viele Eier?
  - Binäre Suche, höchstens $\log_2 n$ Versuche.
- Strategie mit nur einem Ei?
  - Von unten anfangen. n Versuche.
- Strategie mit zwei Eiern:
  - Versuche, mit s Versuchen auszukommen.
  - Kleiner werdende Intevalle
  - $s + (s-1) + (s-2) + \cdots + 2 + 1 = \sum_{i=1}^{n} i = \frac{s(s+1)}{2} \geq 100$. Daher $s = 14$.

Eier werfen

- Strategie für beliebig viele Eier?
  - Binäre Suche, höchstens $\log_2 n$ Versuche.
- Strategie mit nur einem Ei?
  - Von unten anfangen. n Versuche.
- Strategie mit zwei Eiern:
  - Versuche, mit s Versuchen auszukommen.
  - Kleiner werdende Intevalle
  - $s + (s-1) + (s-2) + \cdots + 2 + 1 = \sum_{i=1}^{n} i = \frac{s(s+1)}{2} \geq 100$. Daher $s = 14$.
  - $\sqrt{n}$

Hottest Path

```
int current = 0;
List<Integer> route = new ArrayList<Integer>();
route.add(0);
while (!food[current]) { // termination criterion
  float max = -1;
  int next = -1;
  for (int j = 0; j < edges.length; ++j) {
    if (edges[current][j] != 0 && max < popularity[current][j]) {
      max = popularity[current][j];
      next = j;
    }
  }
  route.add(next);
  current = next;
}
```

Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort				

Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$		

Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl				

Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$		

Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Einfügen				

Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Einfügen	$\Theta(n \log n)$	$\Theta(n \log n)$		

Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Einfügen	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort				

Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Einfügen	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$\Theta(n \log n)$	$\Theta(n^2)$		

Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Einfügen	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n^2)$
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$		

Sortieren und Laufzeiten

Algorithmus	Vergleiche		Vertauschungen	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Auswahl	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Einfügen	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n^2)$
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$

Algorithmus Quicksort($A[l, \ldots, r]$

Input : Array A der Länge n. $1 \leq l \leq r \leq n$.
Output : Array A, sortiert zwischen l und r.

if $l < r$ **then**
    Wähle Pivot $p \in A[l, \ldots, r]$
    $k \leftarrow$ Partition$(A[l, \ldots, r], p)$
    Quicksort$(A[l, \ldots, k-1])$
    Quicksort$(A[k+1, \ldots, r])$

Algorithmus Rekursives 2-Wege Mergesort(A, l, r)

Input : Array A der Länge n. $1 \leq l \leq r \leq n$
Output : Array $A[l, \ldots, r]$ sortiert.

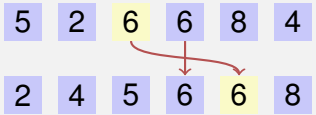
```
if l < r then
    m ← ⌊(l + r)/2⌋          // Mittlere Position
    Mergesort(A, l, m)       // Sortiere vordere Hälfte
    Mergesort(A, m + 1, r)   // Sortiere hintere Hälfte
    Merge(A, l, m, r)        // Verschmelzen der Teilfolgen
```

Algorithmus NaturalMergesort(A)

Input : Array A der Länge $n > 0$
Output : Array A sortiert
repeat
  $r \leftarrow 0$
  while $r < n$ **do**
    $l \leftarrow r + 1$
    $m \leftarrow l$; **while** $m < n$ **and** $A[m+1] \geq A[m]$ **do** $m \leftarrow m + 1$
    if $m < n$ **then**
      $r \leftarrow m + 1$; **while** $r < n$ **and** $A[r+1] \geq A[r]$ **do** $r \leftarrow r + 1$
      Merge(A, l, m, r);
    else
      $r \leftarrow n$
until $l = 1$

Stabile und in-situ-Sortieralgorithmen

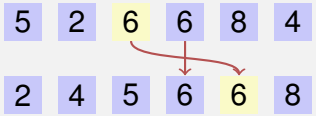
- Stabile Sortieralgorithmen ändern die relative Position von zwei gleichen Elementen nicht.



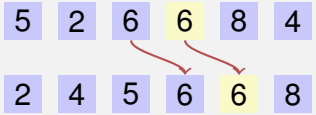
nicht stabil

Stabile und in-situ-Sortieralgorithmen

- Stabile Sortieralgorithmen ändern die relative Position von zwei gleichen Elementen nicht.



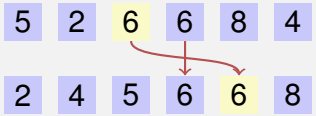
nicht stabil



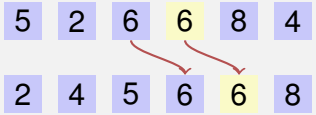
stabil

Stabile und in-situ-Sortieralgorithmen

- Stabile Sortieralgorithmen ändern die relative Position von zwei gleichen Elementen nicht.



nicht stabil



stabil

- In-situ-Algorithmen brauchen nur konstant viel zusätzlichen Speicher.

Zur Bonusübung

Fragen oder Anregungen?