**Ludwig-Maximilians-Universität München**
**Institut für Informatik**
PD Dr. Matthias Renz
Dr. Erich Schubert, Daniel Kaltenthaler

München, 28.4.2015

# Algorithmen und Datenstrukturen

SS 2015

## Übungsblatt 3: O-Notation I und Arrays

Besprechung: 11.5.–15.5.

Abgabe aller mit **Punkten** versehenen Aufgaben bis 10.5. 13:25

### Aufgabe 3-1 O-Notation (Fibonaccizahlen) — Tutoraufgabe

Die Fibonaccifolge ist definiert als: $f_0 = 1$, $f_1 = 1$, $f_n = f_{n-1} + f_{n-2}$ (für $n > 1$)

(a) Geben Sie einen rekursiven Algorithmus zur Berechnung der $n$-ten Fibonaccizahl in Java-Code an und schätzen Sie dessen Laufzeit mittels O-Notation ab.

(b) Überlegen Sie, wie dieser Algorithmus effizienter umgesetzt werden kann. Geben Sie auch für diese Umsetzung eine Laufzeitabschätzung in O-Notation an.

### Aufgabe 3-2 O-Notation I — Tutoraufgabe

Zeigen oder widerlegen Sie:

(a) $4^n = O(3^n)$

(b) $3n^3 + 30n + 300 = O(n^3)$ und $O(n^3) = O(n^3 + 1)$

(c) $\sin^2 n = O(1)$

(d) Sei $p(n)$ ein Polynom in $n$ vom Grad $k \geq 1$. Dann gilt: $O(\log(p(n))) = O(\log n)$

(e) Es gilt für jedes positive $k$:

$$\sum_{i=0}^{n} i^k = O(n^{k+1})$$

(f) $n^{3/2} + n \log_2 n = O(n^{3/2})$

(g) $2^{(2^n)} = O(n^{(2^n)})$

(h) Wenn $f(n) = O(s(n))$ und $g(n) = O(r(n))$ gilt, dann gilt auch $f(n) - g(n) = O(s(n) - r(n))$

### Aufgabe 3-3 O-Notation II — 5 Punkte

Geben Sie an und begründen Sie kurz, ob die folgenden Aussagen wahr oder falsch sind.

(a) $n^2 + 2n + 3 = \Omega(n^2)$

(b) $3n^2 + 2n + 1 = \Theta(n^2)$

(c) $\sqrt{n+1} = O(\sqrt{n})$

(d) $2^{n+1} = O(2^n)$

(e) $x^n = O(n)$

**Aufgabe 3-4 Arraybasierte Listen** **Tutorenaufgabe**

In der Vorlesung wurde die Datenstruktur eines Arrays besprochen.

Wir möchten im Folgenden die Komplexität der Datenstruktur verstehen. Dazu implementieren wir eine dynamische Liste ähnlich der verketteten Liste der vorherigen Übungsblätter, die in einem Array gespeichert wird. In der Praxis finden Sie eine derartige Datenstruktur in Java unter dem Namen `ArrayList`!

Für Komplexitätsbetrachtungen nehmen Sie `maxsize`$=\infty$ an.

- Ergänzen Sie die Methoden `size`, `append` und `prepend`:

```java
/** Array-basierte Liste für Daten vom Typ Object */
public class ObjectArray {
  /** Datenspeicher */
  Object[] array;

  /** Anzahl der belegten Einträge */
  int belegt;

  /** Konstruktor für eine leere Liste
   *
   * @param maxsize Maximale Größe
   */
  public ObjectArray(int maxsize) {
    array = new Object[maxsize];
    belegt = 0;
  };

  /** Direktzugriff auf ein Element */
  public Object get(int position) {
    return array[position];
  }

  /** Länge der Liste. */
  public int size() {
    // TODO: Ergänzen
  }

  /** Element am Ende einfügen */
  public void append(Object daten) {
    if (belegt == array.length) {
      throw new ArrayIndexOutOfBoundsException("Liste ist voll.");
    }
    // TODO: Ergänzen
  }

  /** Element am Anfang einfügen */
  public void prepend(Object daten) {
    if (belegt == array.length) {
      throw new ArrayIndexOutOfBoundsException("Liste ist voll.");
    }
    // TODO: Ergänzen
  }
}
```

- Welche Komplexität haben die Methoden `size()`, `append()`, `prepend()`, `get()`?
- Vergleichen Sie die Komplexität dieser Liste mit der verketteten Liste der vorherigen Übungsblätter.

**Aufgabe 3-5 Arraybasierte Listen II** **8 Punkte**

Ergänzen Sie die Liste aus der vorherigen Aufgabe wie folgt:

- Implementieren Sie wiederum das Interface `Iterable<Object>`:

```
import java.util.*;

/** Array-basierte Liste für Daten vom Typ Object */
public class ObjectArrayPlus extends ObjectArray implements Iterable<Object> {
  /** Konstruktor für eine leere Liste
   *
   * @param maxsize Maximale Größe
   */
  public ObjectArrayPlus(int maxsize) {
    super(maxsize);
  };

  /** Einen Iterator für die Liste erzeugen */
  public Iterator<Object> iterator() {
    return new Iter();
  }

  /** Implementierung eines Iterators */
  public class Iter implements Iterator<Object> {
    /** Aktuelle Position */
    int position;

    /** Konstruktur */
    public Iter() {
      position = 0;
    }

    /** Zum nächsten Element gehen */
    public Object next() {
      // TODO: Ergänzen
    }

    /** Test, ob die Liste ein nächstes Element hat */
    public boolean hasNext() {
      // TODO: Ergänzen
    }

    /** Entfernen nicht erlaubt. */
    public void remove() {
      throw new UnsupportedOperationException();
    }
  }
}
```

- Fügen Sie eine Methode `remove(int idx)` zur Klasse `ObjectArrayPlus` hinzu, mit der der Nutzer das Element an der `i` ten Position entfernen kann.
(Die Methode `remove()` des `Iterator`-Interfaces müssen Sie nicht implementieren!)

```
/** Array-basierte Liste für Daten vom Typ Object */
public class ObjectArrayPlus extends ObjectArray implements Iterable<Object> {
  // Wie bisher ...

  /** Element aus Liste entfernen */
  public void remove(int idx) {
    // TODO: Ergänzen
  }
}
```

- Welche Komplexität in der Größe der Liste $n$ (= `belegt`) hat der Aufruf `remove(0)`?
(Für Komplexitätsbetrachtungen nehmen Sie weiterhin `maxsize`$=\infty$ an.)

- Passen Sie die `main` Methode aus der bekannten `Liste`-Klasse auf diese Datenstruktur an, ergänzen Sie diese um einen Test für die neue `remove()` Methode. Testen Sie ihre Implementierung.