Algorithmen 1

Ziele

- Verständnis der wichtigsten Algorithmen
- Verständnis asymptotischer Laufzeiten
- **Fähigkeit einen guten Algorithmus zu entwerfen**

Organisatorisches

Kontaktdaten

- Florian Weber
- uagws@...
- GPG: 16EF 32D8 211F 14C2 49A6 FBE1 44B5 40D5 **164F 062D**
- Mailingliste: algo1_tut@lists.kit.edu
- github.com/FlorianJW/tutorium_algo1
- Punkte unter algo14.florianjw.de/<pseudonym>.html
- SSL: ausgestellt für *.florianjw.de und florianjw.de; SHA1-Fingerprint: A6:67:1F:7A:CF:AC:41:85:52:3D:5C:69:DE:CE:F0:BA:09:C9:07:A3

Ablauf

- Übungsblätter zu zweit oder alleine
- Handgeschrieben!
- Programmieraufgaben in Java (bäh!) mit Praktomat
- Abschreiben -> schlecht gelaunter Tutor -> härtere Korrektur

Bonusmaterial

- C++ (11 oder 14, machmal experimentel)
- Potentiell neuste Versionen von clang/GCC benötigt

- Blick in die C++-stdlibs
- Performance in der echten Welt
- Steigerung der Code-Qualität durch Algorithmen

Pseudocode

```
fn find(container: Sequence, pred: Predicate): auto
    for element in container:
        if predicate(element):
            return element
    throw not_found_exception
```

- high-level
- relativ hohe Freiheit
- trotzdem: Genaue Spezifikation was wie funktioniert!
- Sehr gerne auch prozedural oder funktional statt OOP (aber nur wo angemessen)
- Mangelnde Dokumentation: Punktabzüge

Maschinenmodel

- Speicher der Größe 2^n, mit Wortbreite n Bits
- Wahlfreier Zugriff in $O(1)$
- Einfache arithmetische Operationen über Wörtern in $O(1)$
- Keine Aussagen über Multithreading
- Toll für theoretische Informatik und sehr intuitiv, aber nicht sehr realistisch

Multiplikationsalgorithmen

Übersicht

- Schulmethode
- Rekursive-Methode
- Karatsuba-Ofman

Laufzeiten

Theorie

- konstante Faktoren sind egal
- Summanden mit langsamarem Wachstum sind egal
- nicht-konstante Faktoren in Algo wichtig

- $O(f(x)) = \{g(x) | \exists c, n_0 : \forall n > n_0 : g(n) \leq c \cdot f(n)\}$
- $\Omega(f(x)) = \{g(x) | \exists c, n_0 : \forall n > n_0 : g(n) \geq c \cdot f(n)\}$
- $\Theta(f(x)) =$
  $\{g(x) | \exists c_0, c_1, n_0 : \forall n > n_0 : c_0 \cdot f(n) \leq g(n) \leq c_1 \cdot f(n)\}$

Aufgabe 1

```
fn find(container: Sequence, pred: Predicate): auto
    for element in container:
        if predicate(element):
            return element
    throw not_found_exception
```

Aufgabe 2

```
fn fold(container: Sequence,
        fun:       Function<U(T,U)>,
        init:      auto
       ): auto
    var value := init;
    for element in container:
        value := fun(element, value)
    return value
```

Aufgabe 3

```
fn fib(n: uint): uint
    if n < 2:
        return 1
    else return fib(n-1) + fib(n-2)
```

Aufgabe 4

```
fn fib(n: uint):uint
    n0 := 1
    n1 := 0
    n2 := 0

    for i in range(0, n):
        n2 := n1
        n1 := n0
        n0 := n1 + n2

    return n0
```

Aufgabe 5

```
fn fib(n: uint): uint
    if n < 2:
        return 1
    static var cache: hash_map<uint, uint>
    if cache.contains(n):
        return cache.get(n)
    value := fib(n-1) + fib(n-2)
    cache.insert(n, value)
    return value
```

- Hashmaps sind Tabellen mit (amortisiertem) Lese- und Einfügezugriff in $O(1)$ (Details später in der Vorlesung)