Prof. Dr.-Ing. Tim Güneysu
B. Sc. Jakob Feldtkeller
B. Sc. Markus Krausz
M. Sc. Georg Land
M. Sc. Jan Richter-Brockmann

Ruhr-Universität Bochum
Elektrotechnik/Informationstechnik
Lehrstuhl Security Engineering

Sommersemester 2020

Informatik 2 - Algorithmen und Datenstrukturen

Übungsblatt 1

Abgabe bis spätestens Mittwoch, 13. Mai 2020, 23:59 Uhr via *moodle* — Σ 100 Punkte

---

Aufgabe 1: Theorie: Obere Schranken (12 Punkte)

Ordnet den folgenden Funktionen jeweils die kleinstmögliche obere Schranke für ihr Wachstum zu. Verwendet dazu die $\mathcal{O}$-Notation.

a) $f(n) = n + 1$
b) $f(n) = (1 + 1/n)(1 + 2/n)$
c) $f(n) = 2n^3 - 15n^2 + n$
d) $f(n) = \lg(2n)/\lg(n)$
e) $f(n) = \sqrt{4^{2n}}$
f) $f(n) = 3^5 n + n^n$

Aufgabe 2: Theorie: Asymptotisches Verhalten (32 Punkte)

Gebt an, welche der Ausdrücke $f(n) \in \mathcal{O}(g(n))$, $f(n) \in \Omega(g(n))$ und $f(n) \in \Theta(g(n))$ mit $n \in \mathbb{N}$ für die folgenden Paare von Funktionen f und g gelten. Begründet eure Antworten unter Anwendung der Definitionen für $\mathcal{O}, \Omega, \Theta$ aus der Vorlesung und gebt den ganzen Rechenweg an.

a) $f(n) = 3^n + n^3$; $g(n) = 3^{n+2}$
b) $f(n) = n \log n + n^2$; $g(n) = (\sqrt{n})^3$
c) $f(n) = \sqrt[4]{2^{8n+4}} \cdot 2^{-n}$; $g(n) = \frac{1}{n^{-5}}$
d) $f(n) = \frac{\log 2n}{\log n}$; $g(n) = (1 + \frac{1}{n}) \cdot (2 + \frac{2}{n}) - (\frac{4}{n} + \frac{2}{n^2})$
e) $f(n) = n \cdot n^2$; $g(n) = 31n^2 + 45n + 21$
f) $f(n) = 2^n + 0.75^{3n}$; $g(n) = 2^{2n}$
g) $f(n) = \log(n!)$; $g(n) = \log(n^n)$
h) $f(n) = \sqrt{n}$; $g(n) = \log n$

Aufgabe 3: Theorie: Laufzeitanalyse (15 Punkte)

Analysiert den unten angegebenen Algorithmus. Gebt bei Berechnungen stets den vollständigen Lösungsweg an.

a) Was tut der Algorithmus? Beschreibt kurz die Funktion, die Ein- und Ausgabeparameter, sowie den groben Berechnungsablauf.
b) Bestimmt die *Worst-, Best- und Average-Case Laufzeit* unter Verwendung der $\mathcal{O}$-Notation. Bestimmt zunächst die Laufzeit jeder einzelnen Anweisung. Berechnet daraus sukzessive die Gesamtlaufzeit der Funktion. Einfache arithmetische Funktionen und Speicherzugriffe haben konstante Laufzeit.

```
   input  : integer array a of length n > 1, integer x
   output : integer y
 1 function f(a, x)
 2 begin
 3     y := 0;
 4     for i = 0 to n − 1 do
 5         tmp = 1;
 6         for k = 1 to i do
 7             tmp = tmp ∗ x;
 8         end
 9         y = y + a[i] ∗ tmp;
10     end
11     return y
12 end
```

Algorithmus 1 : Funktion f(a, x)

Aufgabe 4: Praxis: Calculator (41 Punkte)

Um eure Programmierkenntnisse wieder aufzuwärmen ist eure erste Aufgabe einen Rechner zu programmieren. Genauer gesagt die Funktionen: Dekrementieren, Addieren, Subtrahieren, Multiplizieren, Dividieren und Potenzieren.
Die Herausforderung besteht daraus, dass ihr dafür *nicht* die üblichen Operatoren (+,-,*,/, usw.) nutzen dürft. Sondern nur Funktionen, die ihr weiter oben im Code schon definiert habt und die Grundlagen: Inkrementieren und Negieren. Das heißt zum Beispiel für die Funktion Multiplikation dürft ihr eure Funktion zur Addition benutzen. Rekursion und Fallunterscheidungen sind in einigen Fällen sehr hilfreich.
Im Moodle findet ihr eine Vorlage für die Programmieraufgabe die ihr zwingend nutzen müsst. Euren Code schreibt ihr nur in die `calculator.cpp` Datei in den dafür vorgesehenen Bereichen. Euer Code muss die mitgelieferte Testbench bestehen! Abgaben welche die Testbench nicht bestehen können wir leider nicht bewerten.
Als Lösung bitte nur eure `calculator.cpp` hochladen.

(Die mit dem ATC Game kompatiblen Aufgaben beginnen mit dem nächsten Aufgabenblatt.)