Wiederholung: Boolesche Algebra und Digitale Schaltungen

https://kalle2k.lima-city.de/computerscience/Informatik_12/2020-21/Technische_Informatik/

Boolesche Terme und Schaltalgebra
https://kalle2k.lima-city.de/computerscience/Informatik_12/2020-21/Technische_Informatik/Boolesche_Terme.pdf

Boolesche Terme und Schaltalgebra (mit Beispielen)
https://kalle2k.lima-city.de/computerscience/Informatik_12/2021-22/Technische_Informatik/Boolesche_Terme_und_Schaltalgebra.pdf

Addierer für Dualzahlen
https://kalle2k.lima-city.de/computerscience/Informatik_12/2020-21/Technische_Informatik/Addierer_fuer_Dualzahlen.pdf

Minimale ALU
https://kalle2k.lima-city.de/computerscience/Informatik_12/2020-21/Technische_Informatik/Minimale_ALU.pdf

Die „höheren Rechenarten" Multiplikation, Division und Potenzierung als iterativ formulierte Algorithmen (hier: in Python), die sich auf die in der CPU implementierten Grundoperationen Addition und Subtraktion stützen:
https://kalle2k.lima-city.de/computerscience/Informatik_12/2020-21/Technische_Informatik/Grundrechenarten_GUI.pdf

Digitalschaltungen DigitalSimulator
https://kalle2k.lima-city.de/computerscience/Informatik_12/2020-21/Technische_Informatik/DSIM/

Aufgabenblatt Nr. 9 (08.06.2021)
https://kalle2k.lima-city.de/computerscience/Informatik_12/2020-21/Aufgabenblaetter/Aufgabenblatt_Nr9_inf12.pdf

Wiederholung: Sorting and Searching

SelectionSort

https://kalle2k.lima-city.de/computerscience/Informatik_12/2020-21/SelectionSort/Sortierverfahren_Direkte_Auswahl_29-01-2021.pdf

https://kalle2k.lima-city.de/computerscience/Informatik_12/2020-21/SelectionSort/sorting_by_direct_selection_varianten.pdf

https://kalle2k.lima-city.de/computerscience/Informatik_12/2020-21/SelectionSort/Aufwand_SelectionSort.pdf

MergeSort

https://kalle2k.lima-city.de/computerscience/Informatik_12/2020-21/MergeSort/mergesort_16-04-2021.pdf

MergeSort Übungsblatt:
https://kalle2k.lima-city.de/computerscience/Informatik_12/2021-22/MergeSort/MergeSort_Arbeitsblatt.doc

BinarySearch

https://kalle2k.lima-city.de/computerscience/Informatik_13/2021-2022/BinarySearch/BinaereSuche.pdf

Vergleichende Laufzeitmessungen bei SelectionSort, MergeSort und BinarySearch:

https://kalle2k.lima-city.de/computerscience/Informatik_13/2021-2022/BinarySearch/MergeSort_BinarySearch.py.txt

https://kalle2k.lima-city.de/computerscience/Informatik_13/2021-2022/BinarySearch/SelectionSort_BinarySearch.py.txt

Prinzipien zur Formulierung eines Algorithmus

Imperativer Ansatz

Der Quellcode (formuliert in einer Programmiersprache, z. B. Pascal, Java oder Python) besteht aus einer Folge von ausführbaren Anweisungen, die in der vorgegebenen Reihenfolge nacheinander abgearbeitet werden.

Wesentliche Kontrollstruktur: **Iteration** (realisiert als for- oder while-Schleife)

Funktionaler Ansatz

Die Formulierung des Quellcodes orientiert sich der inneren, in der Regel mathematischen Struktur eines Algorithmus.

Wesentliche Kontrollstruktur: **Rekursion**

Definition:
*Eine Prozedur (Teilprogramm, Subroutine) oder eine Funktion heißt **rekursiv**, wenn deren Anweisungsblock mindestens einen Aufruf von sich selbst enthält.*

Bei beiden Ansätzen ist durch eine Abbruchbedingung sicherzustellen, daß der Algorithmus terminiert, also nach endlich vielen Schritten beendet wird und zu einem Ergebnis führt.

Aufgaben

1. **Fakultätsfunktion** (engl.: factorial)

   Wir ordnen jeder natürlichen Zahl n, $n \geq 0$, die Zahl n! (lies: n-Fakultät) zu:

   $0! = 1$
   $n! = 1 \cdot 2 \cdot \ldots \cdot n$ falls $n > 0$

   a) Formuliere einen in Python geschriebenen iterativen Algorithmus, der nach Eingabe einer natürlichen Zahl n, $n \geq 0$, n! berechnet und ausgibt.

   b) Definiere die Funktion $n \rightarrow$ fact(n) rekursiv und erstelle ein Python-Programm mit rekursivem Funktionsaufruf.

   c) Beurteile die Komplexität des rekursiv formulierten Algorithmus, indem man die Anzahl z(n) der Aufrufe der rekursiven Funktion fact in Abhängigkeit von n bestimmt.

2. **Fibonacci-Folge**

   Für $n \in \{0, 1, 2, 3, \ldots\}$ läßt sich die Fibonacci-Folge rekursiv definieren:

   Rekursionsanfang: **f(0) = 0**
   f(1) = 1

   Rekursionsvorschrift: **f(n) = f(n–1) + f(n–2)** falls $n > 1$

   (In Worten: für n > 1 erhält man das n-te Folgenglied als Summe der beiden vorangehenden Folgenglieder.)

   a) Schreibe und teste ein Python-Programm mit rekursivem Funktionsaufruf, welches nach Eingabe von **n** den Wert **f(n)** ausgibt (oder: alle Werte f(0), . . . , f(n)); implementiere auch eine Variable **z**, welche die Anzahl der Funktionsaufrufe ermittelt.

b) Zeige: Für die Anzahl **z(n)** der Funktionsaufrufe gilt

Rekursionsanfang: **z(0) = z(1) = 1**

Rekursionsvorschrift: **z(n) = 1 + z(n–1) + z(n–2)** falls n > 1

Hinweis:
Erstelle für f(2), f(3), f(4) jeweils ein Baumdiagramm, so wie es für die Aufrufe der Funktion sort in dem paper ***mergesort_16-04-2021.pdf*** *gemacht wurde.*

c) Für **f(n)** gilt die Abschätzung:

$$\mathbf{\tfrac{1}{2} \cdot (\sqrt{2})^n \;<\; f(n) \;<\; \tfrac{1}{2} \cdot 2^n} \qquad \text{falls } n > 2$$

Schließe hieraus, daß die Folge **z(n)** exponentiell mit **n** wächst; folglich ist die rekursive Berechnung der Fibonacci-Folge von exponentieller Komplexität.

Die in c) mitgeteilte Ungleichung ergibt sich aus folgenden Überlegungen:

Behauptung: Die Folge $\{f(n)\}_{n=0}^{\infty}$ ist streng monoton wachsend für n > 1.

Beweis: **f(n+1) – f(n) = f(n-1) > 0** falls n > 1

Behauptung: $\mathbf{f(n) < 2^{n-1} = \tfrac{1}{2} \cdot 2^n}$ falls n > 1

Beweis:
$f(n) = f(n-1) + f(n-2)$ $< 2 \cdot f(n-1)$ wegen der Monotonie
$< 2 \cdot 2 \cdot f(n-2) = 2^2 \cdot f(n-2)$
$< 2^3 \cdot f(n-3)$
. . . . . .
$< 2^{n-1} \cdot f(n-(n-1)) = 2^{n-1} \cdot f(1) = 2^{n-1}$

Behauptung: $\mathbf{f(n) > \tfrac{1}{2} \cdot (\sqrt{2})^n}$ falls n > 2

Beweis: o.B.d.A. sei n gerade mit n = 2·m, m > 1

$f(2m) = f(2m-1) + f(2m-2)$
$> 2 \cdot f(2m-2) = 2^1 \cdot f(2(m-1))$ wegen der Monotonie
$> 2 \cdot 2 \cdot f(2m-4) = 2^2 \cdot f(2(m-2))$
$> 2^3 \cdot f(2(m-3))$
. . . . . .
$> 2^{m-1} \cdot f(2(m-(m-1))) = 2^{m-1} \cdot f(2) = 2^{m-1}$

mit m = n/2 folgt:

$f(n) > 2^{n/2-1} = \tfrac{1}{2} \cdot 2^{n/2} = \tfrac{1}{2} \cdot (\sqrt{2})^n$

Folglich erhalten wir für **f(n)** die Abschätzung

$$\mathbf{\tfrac{1}{2} \cdot (\sqrt{2})^n \;<\; f(n) \;<\; \tfrac{1}{2} \cdot 2^n} \qquad \text{falls } n > 2$$

Die Fibonacci-Folge wächst exponentiell mit n.

Das exponentielle Wachstum läßt sich auch an der für die Fibonacci-Folge geltenden Formel von Moivre-Binet ablesen:

f(n) = $$\frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Für große Werte von n kann man den Subtrahend gegenüber dem Minuend vernachlässigen.

3. Komplexität des Algorithmus **SelectionSort** (SelectionSort ist ein imperativ formulierter Algorithmus)

Um den Aufwand bei **SelectionSort** zu ermitteln, betrachten wir denjenigen Programmteil, der das Sortieren ausführt:

```
j = 0
while j <= n-2:
    i = j + 1
    min = a[j]
    while i < n:
        if a[i] < min:
           min = a[i]
           a[i] = a[j]
           a[j] = min
        i = i + 1
    j = j + 1
```

Wir fassen die Anweisungen aus dem Schleifenrumpf der inneren Schleife dieses Programmauszugs gedanklich zum Anweisungsblock **A** zusammen
(markiere Block **A** im obenstehenden Programmtext).
Um den Aufwand zu ermitteln, ein aus **n** Komponenten bestehendes array zu sortieren, fragen wir, wie oft Block **A** in Abhängigkeit von **n** abgearbeitet wird.

a) Vervollständige die Einträge in folgender Tabelle, wobei z(j) angibt, wie oft Block **A** in Abhängigkeit von j abgearbeitet wird.

Index j	**Index i**	**z(j)**
j = 0	≤ i ≤	
j = 1	≤ i ≤	
j = 2	≤ i ≤	
....	....	....
j = n-3	≤ i ≤	
j = n-2	≤ i ≤	

b) Die Gesamtzahl **z** der Abarbeitungen von Block **A** ergibt sich als

z = z(0) + z(1) + z(2) + z(3) + ... + z(n-3) + z(n-2)

Vereinfache diese Summe und zeige so, daß **z** quadratisch mit **n** wächst!

Hinweis: Für die Summe der ersten n natürlichen Zahlen gilt bekanntlich:
1 + 2 + ... + n = ½ · n · (n + 1)

4. Komplexität des Algorithmus **MergeSort** (MergeSort ist ein funktional formulierter Algorithmus)

Siehe **mergesort_16-04-2021.pdf** (S. 2 ff)