Übungsblatt 1

Beispiel 1.1 O-Notation Rechenregeln
Bestimmen/Vereinfachen Sie die Laufzeitkomplexität der folgenden Ausdrücke:

$O(10n) = ?$
$O(2n + 3) = ?$
$O(14n^3 - 3n^2 + 10) = ?$
$O(3 + 2 \cdot \sin(n)) = ?$
$O(2n^2 + 3n + 3) = ?$
$O(4 \cdot 2^n + 3n^2) = ?$
$O(4n^2 \cdot \log(n) + 2n \cdot (\log(n)))^3 = ?$
$O(17n^2 + 3n + 100) = ?$
$O(4n \cdot (-2 + 3n) \cdot (n - \log(n)) / n) = ?$

Beispiel 1.2 Pseudocode
Geben Sie einen einfachen Algorithmus mit der Laufzeitkomplexität von **$O(m \cdot n)$** in Pseudocode an.

Beispiel 1.3 Typische Komplexitäten
Bewerten Sie die folgenden Komplexitäten mit jeweils einer der Eigenschaften **hervorragend, sehr gut, akzeptabel, ungünstig** und **katastrophal**.

O(f(n))	Bezeichnung	Bewertung
$O(1)$	konstant	
$O(\log(n))$	logarithmisch	
$O(n)$	linear	
$O(n \cdot \log(n))$	quasi linear	
$O(n^2)$	quadratisch	
$O(n^3)$	kubisch	
$O(n^k)$	polynomal	
$O(2^n)$	exponentiell	
$O(n!)$	faktoriell	

Beispiel 1.4 Asymptotische Laufzeitkomplexität

Welche der folgenden Aussagen ist für eine Funktion **g(n)** in **O(f(n))** für wahr, wenn **g(n)** alle Voraussetzungen der Definition einer *O-Notation* erfüllt? Begründen Sie Ihre Wahl.

- g(n) *wächst echt stärker* als f(n)
- g(n) hat *höchstens* die gleiche Ordnung wie f(n)
- g(n) hat *mindestens* die gleiche Ordnung wie f(n)
- g(n) hat *fast genau* die gleiche Ordnung wie f(n)
- g(n) *wächst schwächer* als f(n)

Beispiel 1.5 Asymptotische Laufzeitkomplexität

Welche asymptotische Laufzeitkomplexität haben die folgenden Pseudo Code Blöcke?

a)

```
for (i = 1..n) {
    for (j = 1..5) {
        // do something simple
    }
}
```

b)

```
for (i = 1..n) {
    x[i] = 0
}
for (i = 1..n) {
    for (j = 1..n) {
        x[i] = x[i] + 1
    }
}
```

c)

```
for (i = 1..n) {
    j = m
    while (j > 1) {
        j = j / 10
    }
}
```

d)

```
i = 1
while (i <= n)
    for(j = 1..n) {
        k = n
        while(k > 1) {
            k = k / 2
        }
    }
    i = i + 1
}
```

e)

```
public static void factorial(int n) {
    int result = 1;
    for (int i=1 ; i<=n ; i++) {
        result *= i;
    }
    return result;
}
```

f)

```
public static void bSort(Array A) {
    for (n = A.size; n > 1; --n) {
        for (i = 0; i < n-1; ++i) {
            if (A[i] > A[i+1]) {
                A.swap(i, i+1)
            }
        }
    }
}
```

g)

```
for (i = 1..n) {
    for (j = 1..i) {
        // do something irrelevant
    }
}
```

Beispiel 1.6 Binäre Suche
Ermitteln Sie die Anzahl der Schritte für eine binäre erfolglose Suche nach der Zahl **17** in folgendem (unsortierten) Array. Führen Sie weiters eine umfassende Laufzeitanalyse (schlechtester Fall, bester Fall, durchschnittlicher Fall bei erfolgreicher Suche, durchschnittlicher Fall bei erfolgloser Suche) durch und geben Sie die jeweiligen Laufzeitkomplexitäten an. Etwaige Voraussetzungen, die vorab erfüllt werden müssen, sollen in der Laufzeitanalyse nicht beachtet, aber für den Lösungsweg angeführt werden.

15	18	25	5	74	68	13	20	8	64	1	28	99	77	18	2	35	80	16	42

Beispiel 1.7 Hashing
Welche Anforderungen werden an eine gute Hashfunktion gestellt? Welche Richtwerte gilt es beim Erstellen einer Hashfunktion/Hashtabelle zu beachten?

Beispiel 1.8 Hashing Kollisionsproblem
Erklären Sie kurz das sog. Kollisionsproblem, welches bei Hashing-Verfahren auftreten kann und geben Sie Lösungsmöglichkeiten + Erklärung an.

Beispiel 1.9 Hashfunktion
Bilden Sie den Hashcode durch die gewichtete Summe der Ordinalwerte (ASCII-Code) der einzelnen Zeichen für den String „FHTechnikumWien“ (ohne Sonderzeichen) mit Hilfe der Hashfunktion $h(s) = s[0] * 31^{(n-1)} + s[1] * 31^{(n-2)} \ldots + s[n-1]$

Das Ergebnis soll in einer Hashtabelle mit maximal 99991 Einträgen gespeichert werden können. Prüfen Sie, ob es sich bei 99991 um eine Primzahl handelt, passen Sie ggf. die Größe der Hashtabelle an und bestimmen Sie den Index in der Hashtabelle, wo der String abgelegt werden soll.

Beispiel 1.10 Aufwandsabschätzung
Bestimmen Sie den Aufwand für Best case/Worst case folgender Sortierverfahren nach O-Notation. Recherchieren Sie ggfs. die Funktionsweise und geben Sie weiters an, ob der jeweilige Suchalgorithmus stabil ist, oder nicht.

- Bubblesort:
- Insertionsort:
- Mergesort:
- Quicksort:

Beispiel 1.11 Rekursion
Geben Sie mindestens 2 Probleme an, die bei einer Implementierung mit Rekursion auftreten können.

Beispiel 1.12 Quicksort
Geben Sie eine rekursive Implementierung für Quicksort in Pseudo-Code an und beschreiben Sie die Funktionsweise in Stichworten.

Beispiel 1.13 Quicksort
Ist Quicksort immer schnell? Begründen Sie Ihre Antwort in 2-3 Sätzen. Wenn nein, geben Sie ein konkretes Beispiel an, wo ein anderes Sortierverfahren schneller als Quicksort ist.

Beispiel 1.14 Auswahl eines Sortierverfahrens
Gegeben ist eine Datei mit maximal n positiven ganzen Zahlen, jede kleiner als n ($n = 10^7$)
Gesucht ist eine sortierte Liste der Eingabezahlen in möglichst effizienter Laufzeit nach O-Notation.

Randbedingungen:

- sehr wenig Hauptspeicher verfügbar (wird durch andere Applikationen belegt, es kann nicht die komplette Datei in ein Integer Array geladen werden)
- alle Zahlen sind in der Datei sind voneinander unterschiedlich

Beispiel 1.15 Fraktale
Die Koch-Kurve ist eines der bekanntesten Fraktale. Sie lässt sich, wie unter https://de.wikipedia.org/wiki/Koch-Kurve beschrieben, durch einen iterativen Prozess konstruieren, wobei ihre Länge von Iteration zu Iteration steigt. Die nachstehende Abbildung zeigt die Kochkurve zu Beginn und nach den ersten drei Iterationen.

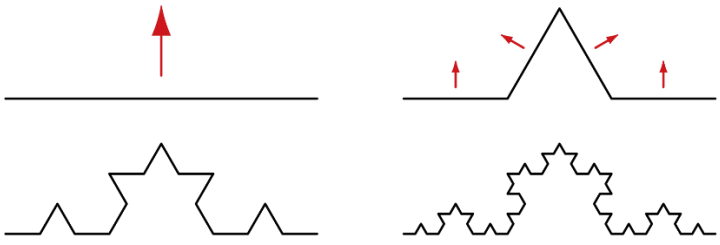

Abbildung 1 - Die ersten drei Iterationen der Koch-Kurve
(Quelle: https://de.wikipedia.org/wiki/Koch-Kurve)

Recherchieren Sie online die Formel für die Länge der Koch-Kurve. Entwickeln Sie eine rekursive Funktion in Pseudo-Code, welche die Längen der Koch-Kurve für eine gegebene Anzahl von Iterationen **n** ermittelt und ggf. am Bildschirm ausgibt, beginnend bei der ersten Iteration. Als Start-Seitenlänge ist 3 gegeben. Als Bsp. für das Ergebnis einer solchen

Kalkulation sind die berechneten Längen der ersten 3 Iterationen nachfolgend angeführt:

- Iteration 0: **3**
- Iteration 1: **4**
- Iteration 2: **5,3**
- Iteration 3: **7,11**
- usw…