

## Übungsblatt 3

### Aufgabe 1

Geben Sie die kleinste Klasse in Gross- $\mathcal{O}$ -Notation an, zu der die folgenden Funktionen gehören. Mit „klein“ ist hier gemeint, dass beispielsweise  $3 \cdot n^2$  in  $\mathcal{O}(n^2)$ ,  $\mathcal{O}(n^3)$  oder in  $\mathcal{O}(2^n)$  ist. In diesem Fall wäre die Antwort „ $3 \cdot n^2 \in \mathcal{O}(n^2)$ “.

Sortieren Sie die Funktionen anschliessend ihrer Grösse nach.

**Tipp:** Die nachfolgende Tabelle rechts enthält einige Klassen der Gross- $\mathcal{O}$ -Notation in aufsteigender Komplexität. Eine umfassendere Erklärung kann auf Wikipedia gefunden werden:

[https://en.wikipedia.org/wiki/Big\\_O\\_notation](https://en.wikipedia.org/wiki/Big_O_notation).

	Gross- $\mathcal{O}$ -Notation	Beispiele
(a) $100 \cdot n + 200 \cdot n + \sqrt{5 \cdot n} + \log n$	$\mathcal{O}(1)$	5, 200, $2^{100}$
(b) $2 \cdot n^3 + n^3 \cdot n^2 + n^4 + n^2 \cdot \log n$	$\mathcal{O}(\log(\log n))$	$\log(\log n) + 5$
(c) $\log(\log n) + 100 \cdot \log n + \sin(n)$	$\mathcal{O}(\log n)$	$\log n + \log^2 n + 100$
(d) $n \cdot \log n^3 + 2 \cdot n + 3 \cdot \log n$	$\mathcal{O}(\sqrt{n})$	$5 \cdot \sqrt{n} + 100$
(e) $3^{n-4}$	$\mathcal{O}(n)$	$4 \cdot n + 100$
(f) $n^{1/2} + n^{1/3}$	$\mathcal{O}(n \log n)$	$n \cdot \log n^5$
(g) $\log^2 n + 100 + 5 \cdot 20$	$\mathcal{O}(n^2)$	$n^2 + 100 \cdot n$
	$\mathcal{O}(n^7)$	$n^7 + 10 \cdot n^5 + 2 \cdot n^3$
	$\mathcal{O}(2^n)$	$2^n + n^{1000}$
	$\mathcal{O}(5^n)$	$5^n + n^{1000}$
	$\mathcal{O}(n!)$	$9^n + n!$

### Aufgabe 2

Bestimmen Sie die asymptotische Laufzeit der folgenden Funktion in Abhängigkeit von  $n$ . Diese Funktion implementiert die Multiplikation von zwei Matrizen der Grösse  $n \times n$ . Bitte beachten Sie, dass die Grösse der Eingabe dieser Funktion eigentlich  $2 \cdot n^2$  entspricht. In diesem Kontext ist es allerdings üblich, die Laufzeit in Abhängigkeit von  $n$  zu bestimmen.

```

1 def multiply(A, B):
2     # Erinnerung: A ist eine Liste von Listen.
3     # len(A) gibt die Anzahl Listen in der Liste A zurueck.
4     # Da A nach Voraussetzung quadratisch ist,
5     # entspricht n genau len(A).
6     n = len(A)
7
8     # C initialisieren
9     C = []
10    for i in range(0, n):
11        tmp = [0] * n
12        C.append(tmp)
13
14    for i in range(0, n):
15        for j in range(0, n):
16            # calculate C[i][j]
17            tmp = 0
18            for k in range(0, n):
19                tmp += A[i][k] * B[k][j]
20            C[i][j] = tmp
21    return C

```

### Aufgabe 3

Bestimmen Sie die asymptotische Laufzeit der folgenden Funktion. Die Eingabe der Funktion ist eine Liste der Länge  $n$ . Für die Elemente in der Liste kann eine konstante Grösse angenommen werden, womit  $n$  als Eingabegrösse betrachtet werden kann.

```

1 def get_largest_element(l):
2     tmp = l[0]
3     for number in l:
4         if tmp < number:
5             tmp = number
6     return tmp

```

### Aufgabe 4

Bestimmen Sie die asymptotische Laufzeit der folgenden Funktion. Die Eingabe der Funktion ist wieder eine Liste der Länge  $n$ . Für die Elemente in der Liste kann auch hier eine konstante Grösse angenommen werden, womit  $n$  wieder als Eingabegrösse betrachtet werden kann.

```

1 def descending_sort(l):
2     for i in range(0, len(l)):
3         for j in range(0, len(l)-1):
4             if l[j] < l[j+1]:
5                 tmp = l[j]
6                 l[j] = l[j+1]
7                 l[j+1] = tmp
8     return l

```

## Aufgabe 5

Kopieren Sie das untenstehende Programm in eine Sandbox in *Code-Expert* und führen Sie es aus. Haben Sie Geduld, dies kann einige Zeit dauern. Stellen Sie eine Theorie auf, wieso diese Funktion so langsam ist.

Bei dieser Aufgabe geht es nicht um die Gross- $\mathcal{O}$ -Notation, sondern um die tatsächliche Zeit, die das Programm für die konkrete gegebene Eingabe braucht.

**Tipp 1:** `expensive_function(l)` erhält eine Liste, quadriert alle Zahlen darin und addiert sie dann zusammen. Anschliessend prüft die Funktion, ob das Resultat eine Primzahl ist. Falls das Resultat eine Primzahl ist, wird es zurückgegeben, sonst gibt die Funktion `-1` zurück.

**Tipp 2:** `func(l)` nimmt eine Liste `l` und zählt wie oft das erste Element addiert werden kann, bis das Resultat grösser ist als der Rückgabewert von `expensive_function(l)`.

```
1 l = [2, 5, 6, 7, 10, 200, 17, 1]
2
3 def expensive_function(l):
4     # Quadriert alle Zahlen in l und addiert sie zusammen
5     square_sum = 0
6     for i in l:
7         square_sum += i * i
8     # Findet heraus, ob diese Zahl eine Primzahl ist
9     is_prime = True
10    for i in range(2, square_sum):
11        if square_sum % i == 0:
12            is_prime = False
13    # Falls square_sum eine Primzahl ist, gib sie zurueck
14    if is_prime:
15        return square_sum
16    # Sonst gib 10000 zurueck
17    else:
18        return 10000
19
20 def func(l):
21     s = l[0]
22     i = 0
23     while s < expensive_function(l):
24         s += l[0]
25         i += 1
26     print(i)
27
28 func(l)
```

## Aufgabe 6

Vergleichen Sie das Programm aus dieser Aufgabe mit dem aus [Aufgabe 5](#). Was ist anders? Wie könnte sich diese Veränderung auf die Laufzeit des Programms auswirken?

Kopieren Sie das untenstehende Programm in eine Sandbox in *Code-Expert* und führen Sie es aus.

```
1 l = [2, 5, 6, 7, 10, 200, 17, 1]
2
3 def expensive_function(l):
4     # Quadriert alle Zahlen in l und addiert sie zusammen
5     square_sum = 0
6     for i in l:
7         square_sum += i * i
8     # Findet heraus, ob diese Zahl ein Primzahl ist
9     is_prime = True
10    for i in range(2, square_sum):
11        if square_sum % i == 0:
12            is_prime = False
13    # Falls square_sum eine Primzahl ist, gib sie zurueck
14    if is_prime:
15        return square_sum
16    # Sonst gib 10000 zurueck
17    else:
18        return 10000
19
20 def fast_func(l):
21     limit = expensive_function(l)
22     s = l[0]
23     i = 0
24     while s < limit:
25         s += l[0]
26         i += 1
27     print(i)
28
29 fast_func(l)
```

## Aufgabe 7

Die Funktion `print_ind_div_by_3(D)` soll eine Matrix `D` erhalten und alle Elemente von `D` ausgeben, bei welchen der 2. Index durch 3 teilbar ist, es werden also alle `D[i][j]` ausgegeben, für die `j` durch 3 teilbar ist.

```
1 D = [ [1, 2, 3, 4, 5],
2       [6, 7, 8, 9, 10],
3       [1, 2, 3, 4, 5],
4       [6, 7, 8, 9, 10],
5       [1, 2, 3, 4, 5],
6       [6, 7, 8, 9, 10] ]
7
8 def print_ind_div_by_3(D):
9     for row in D:
10        for j in row:
```

```
11         if j % 3 == 0:
12             print(row[j], end = " ")
13         print()
14
15 print_ind_div_by_3(D)
```

Leider ist dabei ein Fehler passiert. Finden Sie den Fehler und korrigieren Sie ihn, so dass die Funktion `print_ind_div_by_3(D)` das Folgende ausgibt.

```
1 1 4
2 6 9
3 1 4
4 6 9
5 1 4
6 6 9
```