2️⃣

Blatt 2

Aufgabe 1

Beweisen Sie durch Widerspruch, dass $2^{2n} \notin \mathcal{O}(2^n)$.

$Z.Z: 2^{2n} \in \mathcal{O}(2^n) \Rightarrow \exists c > 0 \land n > n_0 : 2^{2n} \leq c * 2^n$

$2^{2n} \leq c * 2^n \quad | : 2^n$
$2^n \leq c * 1 \Rightarrow 2^{2n} \notin \mathcal{O}(2^n)$

Widerspruch! Weil es kein c geben welches diese Ungleichung erfüllt! Schließlich kann n unendlich sein, weshalb auch unendlich sein muss, wenn $2^{2n} \in \mathcal{O}(2^n)$ gelten sollte.

Aufgabe 2

Bestimme Sie die Laufzeit-Komplexität der folgenden Algorithmen in Groß-O-Notation in Abhängigkeit von n.

1.

```
1: Input: matrices a,b,c of size n x n
2: for i ← 0 to n − 1 do
3:  for j ← 0 to n − 1 do
4:    for k ← 0 to n − 1 do
5:      c[i][j] ← c[i][j] + a[i][k] + b[k][j]
6:     end for
7:   end for
8: end for
```

$\mathcal{O}(n^3)$ da es drei nested Loops gibt die immer in Abhängigkeit von n durchlaufen
$\Rightarrow n * n * n = n^3$

2.

```
1: sum ← 0
2: for i ← 1 to n do
3:  for j ← 1 to 10000 do
4:   sum ← sum + j + i
5:  end for
6: end for
```

$\mathcal{O}(n)$ die Funktion wäre $f(n) = n * 10000$

3.

```
1: Input: array a of size n
2: while swapped do
3:  swapped ← false
4:   for j ← 1 to n − 1 do
5:    if a[j − 1] > a[j] then
6:      swap(a[j − 1], a[j])
7:      swapped ← true
8:    end if
9:   end for
10: end while
```

$\mathcal{O}(n)$ wenn die Liste bereits sortiert worden ist, bei einer unsortierten liste muss mindestens einmal abgewartet werden bis das größte element ganz am Ende steht und danach muss erneut durchiteriert werden, d.h.:
$\Rightarrow \mathcal{O}(n^2)$

Aufgabe 3

- $10n$
  - $\mathcal{O}(n)$
  - $\Omega(n)$
  - $\Theta(n)$
  - $\sim (10n)$
- $66n^2 + 17n^3 - 120n$
  - $\mathcal{O}(n^3)$
  - $\Omega(n^3)$
  - $\Theta(n^3)$
  - $\sim (17n^3)$
- $5 + 8\ log_2\ n$
  - $\mathcal{O}(log_2\ n)$
  - $\Omega(log_2\ n)$
  - $\Theta(log_2\ n)$
  - $\sim (8 \cdot log_2\ n)$
- $4 \cdot 2^n + 9n^{100}$
  - $\mathcal{O}(2^n)$
  - $\Omega(2^n)$
  - $\Theta(2^n)$
  - $\sim (4 \cdot 2^n)$

Aufgabe 4

Erstellen Sie einen rekursiven Algorithmus für die folgenden drei Funktionen. Geben Sie (Pseudo)code ab.

1. $pow(a, b)$, gibt a^b zurück.

```
1: function power
2:  Input: base a and exponent b
3:  if b is not 0 then
4:    return base * (power(a, b-1))
5:  else
6:    return 1
7:  end if
8: end function
```

2. $fib(n)$, gibt n-tes Element der Fibonacci-Folge zurück.

```
1: function fib
2:  Input: n
3:  a ← 0
4:  b ← 1
5:  return function saver(n,a,b)
6: end function
7:
8: function saver
9:  Input: n, a , b
10:  if n == 0
11:    return a
12:  end if
13:  if n == 1
14:    return b
15:  end if
16:  return saver(n-1, b, a + b)
17: end function

saver(5,0,1) =
saver(4, 1, 1) = saver(3, 1, 2) = saver(2, 2, 3) =  saver(1,3,5) = saver(0,5,8)
         1               2               3               5
```

3. $isPalindrom(str)$

```
1: str ← string
2: start ← 0
3: end ← length of string - 1
4: function isPalindrom
5:   if length of str == 0||1
6:     return true
7:   else if start is not end
8:     return false
9:   else if start < end + 1
10:    return isPalindrom(str[start + 1 to end -1])
11:  end if
12: end function
```