Aufgabenblatt 3 - O-Notation

Aufgabe 1: Komplexitätsklassen

Füllen Sie die folgende Tabelle mit den unterschiedlichen Funktionswerten für die angegebenen Werte für n aus,
mindestens so weit Ihr Taschenrechner reicht.

N	log_2 (n)	√n	n* log_2 (n)	n^2	n^3	2^n	3^n
16	4	4	64	256	4096	65536	43046721
64	6	8	384	4096	262144	1,84*10^19	3,43*10^30
256	8	16	2048	65536	16777216	1.15*10^77	1.39*10^257
1024	10	32	10240	1048576	1073741824	1,797*10^308	3,733*10^512
2^20	20	1024	20.971.520	1,09*10^12	1,15*10^15	∞	∞

Welche Schlussfolgerungen können Sie aus den Werten in der Tabelle auf die Laufzeit von Algorithmen ziehen?
Die Laufzeit des Algorithmus ist eine Kombination aus linearem und logarithmischen Aufwand und steigt etwas stärker als linear mit der Eingabegröße an. Moderateraufwand

Algorithmen mit logarithmischem Zeitaufwand (O(log n)) sind auch für große *n* effizient.

Algorithmen mit quadratischem Zeitaufwand (O(n^2)) sind für moderate *n* noch anwendbar, aber sie werden schnell unpraktikabel, wenn *n* wächst.

Algorithmen mit kubischem Zeitaufwand (O(n^3)) sind bereits für relativ kleine *n* ineffizient.

Exponentielle Algorithmen (O(2^n)) und (O(3^n)) sind extrem ineffizient und praktisch nicht anwendbar, sogar für kleine n.

Aufgabe 2: O-Notation

Zeigen Sie, dass die folgenden Aussagen wahr sind oder widerlegen Sie sie. Begründen Sie Ihre Entscheidung und geben Sie
bei wahren Aussagen ein geeignetes c und n0 an.

https://share.goodnotes.com/s/6nYE4EdnefcNgm0Z3eDaBH

1. $27 \in O(1)$
2. $\frac{n(n-1)}{2} \in O(n^2)$
3. $max(n^3, 10n^2) \in O(n^2)$
4. $logn \in \Omega(n)$
5. $2n + 4 \in \Theta(n)$

1.

27 ist eine Konstante daher war
c = 27
n = 1

2. $\frac{n(n-1)}{2} \in O(n^2)$

$\frac{n(n-1)}{2} = \frac{n^2-n}{2}$ | *2
$n^2 - n \leq 2 * c * a^2$ | c= 1/2

$n^2 - n \leq n^2$ gilt wenn $n \geq 1$ und c = 1/2

Wahr da c und n_0 positiv sind

3. $max(n^3, 10n^2) \leq c * n^2$

$max(n^3, 10n^2) \leq 10n^2$ bis n = 10
Danach gilt es nicht mehr

max() nimmt immer das größere

Also trifft nicht zu

4. $\log n \in \Omega(n)$

$n \geq n_0$
$\log n \geq c \cdot n$
nicht wahr da n stärker wächst als log n

N	$\log_2 n$
1	0
2	1
3	1,58496
4	2
4000	11,9658

5. $2n + 4 \in \theta(n)$

$O(n)$
$2n + 4 \leq c \cdot n$ | -4
$2n \leq c \cdot n - 4$ | c = 3
$2n \leq 3 \cdot n - 4$

$n \geq 4$
$c = 3$

Wahr

$\Omega(n)$
$2n + 4 \geq c \cdot n$ | c = 1
$2n + 4 \geq 1 \cdot n$

$c = 1$
$n \geq 1$

Wahr
$\theta(n)$ = Wahr

Aufgabe 3: Codeanalyse

Bestimmen Sie zunächst die Anzahl elementarer Rechenschritte wie Vergleiche, Zuweisungen und arithmetischer Operationen
für die folgenden Codestücke in Abhängigkeit von der Anzahl n „beteiligter“ Elemente. Die zu betrachtenden
Elementarschritte sind jeweils angegeben. Geben Sie dann den Aufwand in O-Notation an.

1. Vertauschen zweier Feldelemente: (Anzahl der Zuweisungen?)

```
class CodeAnalysis {
   static void exchange(int[] array, int index1, int index2) {
       array[index1] = array[index1] + array[index2];
       array[index2] = array[index1] - array[index2];
       array[index1] = array[index1] - array[index2];
   }
}
```

Zuweisungen = 3
Anzahl elementarer Rechenschritte = 6

$O(1)$, da nicht von der große des Arrays beeinflusst

2. Suche des Minimums in einem Array: (Anzahl der Vergleiche?)

```
class CodeAnalysis {
   static int minimum(int[] array) {
      int minimum = array[0];
      int index = 1;
      while (index < array.length) {
         if (array[index] < minimum) {
            minimum = array[index];
         }
         index++;
      }
      return minimum;
   }
}
```

Zuweisungen = 3
Vergleiche = 2n
Anzahl elementarer Rechenschritte = 4+2n

$O(n)$, da hier hier von der Arraygröße linear abhängt

3. Suche eines Wertes in einem sortierten Array: (Anzahl der Vergleiche?)

```
class CodeAnalysis {
   static int indexOfValue(int[] array, int value) {
      int from = 0;
      int to = array.length - 1;
      int middle;
      while (from <= to) {
         middle = (from + to) / 2;
         if (value < array[middle]) {
            to = middle - 1;
         } else if (value > array[middle]) {
            from = middle + 1;
         } else {
            return middle;
         }
      }
      return -1;
   }
}
```

Zuweisungen = 5
Vergleiche = 3 * log n
Anzahl elementarer Rechenschritte = 13

$O(logn)$ da es immer wieder halbiert wird

Aufgabe 4: O-Notation

Die folgende Funktion implementiert die Suche eines Wertes in einem sortierten Array. Bestimmen Sie zunächst die rekursive Aufwandsfunktion für die Anzahl der erforderlichen Vergleiche und bestimmen Sie dann die Komplexität der Funktion in O-Notation.

```
class CodeAnalysis {
   static int indexOfRecursive(int[] array, int value, int from, int to) {
      int middle;
      if (from <= to) {
         middle = (from + to) / 2;
         if (value < array[middle]) {
            return indexOfRecursive(array, value, from, middle - 1);
         } else if (value > array[middle]) {
            return indexOfRecursive(array, value, middle + 1, to);
         } else {
            return middle;
         }
      }
      return -1;
   }
}
```

$$t_n = at(\frac{n}{b}) + f(n)$$

$$t_n = 1t(\frac{n}{2}) + 1$$

a = 1
b = 2
f(n) = 1

$O(logn)$, da auch hier wieder halbiert wird

N	t(n)	Log n
1	1	0
2	2	1
4	3	2
8	4	3
16	5	4
32	6	5
64	7	6