a) Ein Beweis war eig. nur bei A2 nötig, aber schön!

A_1: Gegeben: $T_1(n) = 2^n + 7$

Dann $T_1(n) \in O(2^n)$, denn $C = 19$ und $n_0 = 1$

$$\begin{aligned} T_1(n) &= 2^n + 7 \\ &\leq 2^n + 7^n \\ &\leq 19^n \\ &\leq C \cdot a^n \quad \text{für } n \geq n_0 \end{aligned}$$

A_2: Gegeben: $2n^3 + 7n + 5$

Dann $T_2(n) \in O(n^3)$, denn $C = 14$ und $n_0 = 1$

$$\begin{aligned} T_2(n) &= 2n^3 + 7n + 5 \\ &\leq 2n^3 + 7n^3 + 5n^3 \\ &\leq 14n^3 \\ &\leq C \cdot n^3 \quad \text{für } n \geq n_0 \end{aligned}$$

A_3: Gegeben: $T_3(n) = 14n^2 + 300$

Dann $T_3(n) \in O(n^2)$, denn $C = 314$ und $n_0 = 1$

$$\begin{aligned} T_3(n) &= 14n^2 + 300 \\ &\leq 14n^2 + 300n^2 \\ &\leq 314n^2 \\ &\leq C \cdot n^2 \quad \text{für } n \geq n_0 \end{aligned}$$

b) A_2: Gegeben: $2n^3 + 7n + 5$

Dann $T_2(n) \in O(n^3)$, denn $c = 14$ und $n_0 = 1$

$$T_2(n) = 2n^3 + 7n + 5$$
$$\leq 2n^3 + 7n^3 + 5n^3$$
$$\leq 14n^3$$
$$\leq c \cdot n^3 \quad \text{für } n \geq n_0$$

c) Var: $T_2(n) = 2n^3 + 7n + 5$, $n > 0$

Behauptung: $T_2(n) \notin O(n^2)$

Beweis: per Gegenbeispiel:

$$T_2(n) = 2n^3 + 7n + 5$$
$$\leq 2n^2 + 7n^2 + 5n^2$$
$$\leq 14n^2$$
$$\leq c \cdot n^2 \quad \text{für } n \geq n_0$$

dann ist $c = 14$

dann gilt

$T_2(n) = O(n^2)$

$\Leftrightarrow T_2(2) \leq c \cdot 2^2$

$\Leftrightarrow 35 \leq 14 \cdot 2^2$

$\Leftrightarrow 35 \leq 56$ ↯

Damit ist nur gezeigt, dass T2 für c = 14 nicht in O(n^2) liegt - theoretisch könnte die Bedingung für irgendein anderes c aber erfüllt sein. Daher muss man entweder zeigen, dass
a) Es generell keine natürliche Zahl c und kein n0 gibt, wo gilt T2(n) <= c * n^2 f. a. n >= n0 oder
b) T2(n) ∈Ω(n^3) => T2 kann nicht in O(n^2) liegen

d) 1 Rechenschritt = 1ns

$n = 100$

$A_1: T_1(100) = 2^{100} + 7$

$= 1{,}26765 \cdot 10^{30}$

$\Rightarrow 1{,}26765 \cdot 10^{30}$ ns

$A_2: T_2(100) = 2 \cdot 100^3 + 7 \cdot 100 + 5$

$= 2.000.705$

$\Rightarrow 2.000.705$ ns

$A_3: T_3(100) = 14 \cdot 100^2 + 300$

$= 140300$

$\Rightarrow 140300$ ns

e) $T_1(10) = 2^{10} + 7 = 1031$

$T_2(10) = 2 \cdot 10^3 + 7 \cdot 10 + 5 = 2075$

$T_3(10) = 14 \cdot 10^2 + 300 = 1700$

Für den praktischen Einsatz eignet sich der Algorithmus A_1 bei einer Eingabegröße ≤ 10 am besten, da die Kosten bei dieser Eingabegröße am geringsten sind. Erst bei größeren Eingabegrößen ist A_1 deutlicher teurer als die anderen, da das Wachstumsverhalten exponentiell ist. Bei größeren Eingaben sind die Algorithmen

A_2 und A_3 mit einem quadratischen bzw. kubischen Wachstumsverhalten günstiger, doch bei kleinen Eingabegrößen ≤ 10 teurer als beim exponentiellen.

1,5/4

Übung 04

Aufgabe 2)

Ich vermute, dass die kleinste O-Klasse für $T(n)$ bei $O(2^n)$ liegt, da in der inneren for-Schleife bis Math.pow(2, i), also 2^i iteriert wird.

1 -> in der inneren Schleife wird in jedem Durchlauf eine Multiplikation ausgeführt

$$T(n) = \sum_{i=0}^{n} \sum_{i=0}^{2^n} x \cdot 2 \quad , \quad n > 0 .$$

Das bedeutet je öfter durchlaufen wird bzw. je größer n ist, desto unterscheidbarer wird der Algorithmus in eine kleinere O-Klasse passen.

$T(n) \in O(2^n)$, denn $c = 4x$ und $n_0 = 1$

$$\begin{aligned} T(n) &= 2^n \cdot x \cdot 2 \\ &\leq 2^n \cdot x^n \cdot 2^n \\ &\leq 4x^n \quad , \text{ für } n \geq n_0 \end{aligned}$$

Vor.: $T(n) = 2^n \cdot x \cdot 2 \quad , n > 0$

Beh.: $T(n) \in O(2^n)$,

$T(n) \notin O(n^3)$

Beweis: Sei $n = 1000$ siehe Aufgabe 1

dann gilt

$$2^{1000} \cdot x \cdot 2 > 1000^3 \cdot x \cdot 2$$

$$\Leftrightarrow \sim 10^{300} \cdot x \cdot 2 > 1000\,000\,000 \cdot x \cdot 2$$

Somit kann $T(n)$ kein Element von $O(n^3)$ sein, da es bei hoher Anzahl an Durchläufen nicht passt

□

Aufgabe 3 0,5/3

```java
package uebung04;


public class Uebung043 {

	public static void main(String[] args) {
		int[] array1 = { 1, 3, 4, 5, 6, 9 };
		int[] array2 = { 2, 3, 5, 7, 8, 9, 10 };
		int length = array1.length + array2.length;
		int[] resultarray = new int[length];



		int newindex = 0;

		for(int k = 0; k < array1.length; k++) {
			resultarray[newindex] = array1[k];
			newindex++;
		}

		for(int j = 0; j < array2.length; j++) {
			resultarray[newindex] = array2[j];
			newindex++;
		}

		selectionsort(resultarray);

		for( int values: resultarray) {
				System.out.print(values + " ");
		}

	}

	public static int[] selectionsort(int[] resultarray) {


			for (int i = 0; i < resultarray.length; i++) {
				int minIndex = i;

				for(int j = i + 1; j < resultarray.length; j++) {
					if (resultarray[j] < resultarray[minIndex]) {
						minIndex = j;
					}

				}
			swap(resultarray, i, minIndex);
			}


			return resultarray;

		}



	private static void swap(int[] array1, int index1, int index2) {
		int tmp = array1[index1];
		array1[index1] = array1[index2];
		array1[index2] = tmp;
	}


}
```

richtige Länge +0,5P

das Array sollte schon beim Zusammenfügen sortiert werden
- und nicht erst zusammengefügt und danach sortiert werden
-> Aufgabenstellung nicht erfüllt

```java
package uebung04;

public class Uebung044 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		int[][] sudoku = new int [][] {
			{ 0 , 9 , 0 , 0 , 0 , 0 , 1 , 1 , 0 } ,
			{ 8 , 0 , 4 , 0 , 2 , 0 , 3 , 0 , 7 } ,
			{ 0 , 6 , 0 , 9 , 0 , 7 , 0 , 2 , 0 } ,
			{ 0 , 0 , 5 , 0 , 3 , 0 , 1 , 0 , 0 } ,
			{ 0 , 7 , 0 , 5 , 0 , 1 , 0 , 3 , 0 } ,
			{ 0 , 0 , 3 , 0 , 9 , 0 , 8 , 0 , 0 } ,
			{ 0 , 2 , 0 , 8 , 0 , 5 , 0 , 6 , 0 } ,
			{ 1 , 0 , 7 , 0 , 6 , 0 , 4 , 0 , 9 } ,
			{ 0 , 3 , 0 , 0 , 0 , 0 , 0 , 8 , 0 }

		};

		//Prüfung jede Ziffer pro Zeile
		//Durlauf Zeile 0-8
		for (int i = 0; i <= 8; i++) {
			//Durchlauf Index 0-8
			for (int j = 0; j <= 8; j++) {
				//Vergleich index j mit index k 0-8
				for (int k=0; k <= 8; k++) {

				// Spalte ungelich Null
				if (sudoku[i][j] != 0) {

					// Gleiche Zahl
					if(sudoku[i][j] == sudoku[i][k]) {
```

```
                    if( j != k) {
                    System.out.println("Dublicate number '" + sudoku[i][j] + "' in row " + i);
                    }
                    }

               }



          }

          }

     }

     //Prüfung jede Ziffer pro Spalte
               //Durlauf index 0-8
               for (int l = 0; l <= 8; l++) {
                    //Durchlauf  Reihe 0-8
                    for (int m = 0; m <= 8; m++) {
                         //Vergleich index m mit index n 0-8
                         for (int n=0; n <= 8; n++) {

                         // Wert ungelich Null
                         if (sudoku[m][l] != 0) {

                              // Gleiche Zahl
                              if(sudoku[m][l] == sudoku[n][l]) {
                                   if (m != n) {
                                   System.out.println("Dublicate number '" + sudoku[m][l] + "'
                                   in column " + l + ".");
                                   }
```

```
                                   }


                              }


                         }

                         }

                    }
int square = 20;
     //index 0-2
     for( int o = 0; o <= 8; o++) {
          //Zeile 0-2
          for( int p = 0; p <= 8; p++){
               //Vergleich mit index 0-2
               for(int q = 0; q <= 8; q++ ) {
                    //Prüfung ob Wert belegt
                    if (sudoku[p][o] != 0 ) {
                         //Prüfen auf Gleichheit
                         if(sudoku[p][o] == sudoku[p][q]) {
                              if(o != q) {

                                   //Prüfung welches square
                                   if (o < 3 && p < 3) {
                                        square = 0;
                                   }

                                   else if (o < 6 && p < 3) {
                                        square = 1;
```

```
            }

            else if (o < 9 && p < 3) {
                 square = 2;
            }

            else if (o < 3 && p < 6) {
                 square = 3;
            }

            else if (o < 6 && p < 6) {
                 square = 4;
            }

            else if (o < 9 && p < 6) {
                 square = 5;
            }

            else if (o < 3 && p < 9) {
                 square = 6;
            }

            else if (o < 6 && p < 9) {
                 square = 7;
            }

            else if (o < 9 && p < 9) {
                 square = 8;
            }
            System.out.println("Dublicate number '" + sudoku[p][o] +
            "' in square " + square + ".");
      }
```

```
                                }
                            }
                        }
                    }

            }


    }
    }
```