Aufgabe : Laufzeit von Selection Sort

```
public class SelectionSort{

    int [] liste = {4,7,2,9,5,1};

     void selectionsort () {

        for (int i = 0;i < liste.length-1;i = i + 1) {

            int position_minelement;

            /* Suche des kleinsten Elements der Restliste */

            position_minelement = suche_position_minelement(i);

            /* Vertausche minelement und i-tes Element */

            tausche_element(position_minelement, i);

        }

    }

/* Hilfsmethode zur Bestimmung der Position des kleinsten Elements des uebergebenen
Feldes ab dem i-ten Feldelement */

    int suche_position_minelement (int i) {

        int position_minelement = i;

            for (int j = i;j < liste.length;j = j + 1){

                if (liste[position_minelement] > liste[j]){

                    position_minelement = j;

                }

            }

                return position_minelement;


     }

     /* Hilfsmethode zum Vertauschen zweier Elemente */

      void tausche_element (int a, int b) {

         int hilfsvariable = liste[a];

         liste[a] = liste[b];

         liste[b] = hilfsvariable;

     }


}
```

a) Veranschauliche am Beispiel [4, 7, 2, 9, 5, 1] wie der Sortieralgorithmus funktioniert. Auf Details der Methoden `suche_position_minelement` und `tausche_element` brauchst du beim Veranschaulichen nicht einzugehen.

liste: [4, 7, 2, 9, 5, 1],
index: [0, 1, 2, 3, 4, 5]

i	position_minelement	liste	Erläuterung		
		[4, 7, 2, 9, 5, 1]			
0	5 (Wert 1)	**[1, 7, 2, 9, 5, 4]**	Werte 1 und 4 getauscht		
1	2 (Wert 2)	**[1, 2, 7, 9, 5, 4]**	Werte 2 und 7 getauscht		
2	5 (Wert 4)	**[1, 2, 4, 9, 5, 7]**	Werte 4 und 7 getauscht		
3	4 (Wert 5)	**[1, 2, 4, 5, 9, 7]**	Werte 5 und 9 getauscht		
4	5 (Wert 7)	**[1, 2, 4, 5, 7, 9]**	Werte 7 und 9 getauscht		

Die Schleife kann nach Erreichen vorletzten Element abgebrochen werden, da das letzte Element dann bereits automatisch an der richtigen Position steht.

b) Wie oft wird bei einer Liste der Länge n die for-Schleife in der Methode `selectionsort` durchlaufen?
Wie viele Vergleiche sind in der Methode `suche_position_minelement` maximal notwendig?

Welches Laufzeitverhalten folgt daraus für den Sortieralgorithmus?

Die Schleife wird n – 1 mal durchlaufen. In einem Durchlauf sind dann maximal n – j Vergleiche notwendig.

Der Algortihmus hat quadratisches Laufzeitverhalten.

c) Welchen Einfluss haben best case und worst case für diesen Algorithmus?

Die Vorsortierung spielt bei diesem Algorithmus keine Rolle.

Aufgabe: Verschlüsselungsverfahren

a) Ein sehr einfaches Verschlüsselungsverfahren für Wörter des Alphabets {A,B,C, …,Z} ist der sog. Cäsar-Code.

Dabei wird eine Zufallszahl k zwischen 1 und 25 erzeugt und jeder Buchstabe eines Wortes durch den Buchstaben ersetzt, der im Alphabet k Stellen weiter ist. Erreicht man das Ende des Alphabets, geht man zyklisch zum Anfang weiter (Bsp.: k = 3; A → D, Y → B).

Schätze ab, wie lange ein Programm zur Entschlüsselung eines zehnstelligen Wortes mit Hilfe des Brute Force Verfahrens benötigt, wenn man für einen Rechenschritt 10 ns veranschlagt.

Für alle 25 Werte von k rechnet man einfach das zehnstellige Wort zurück.

Laufzeit = 25 * 10 * 10 ns = 2500 ns = 2,5 ms

b) Bei einer sog. monoalphabetischen Verschlüsselung, wird jeder Buchstabe durch einen Buchstaben eines anderen Alphabets mit gleicher Zeichenzahl ersetzt.
(z.B. A→, B→, C→, →, …)

Schätze ab, wie lange ein Programm für die Entschlüsselung des Codes mit Hilfe des Brute Force Verfahrens für das Alphabet mit 26 Buchstaben benötigt.

Für die Entschlüsselung muss man maximal alle Permutationen des Alphabets durchtesten.

Laufzeit = $26! * 10ns = 4 * 10^{18}$ s = 130 Mrd Jahre

Wie könnte die Entschlüsselungszeit für korrekte Wörter einer Sprache verkürzt werden?

Man könnte vor der Entschlüsselung eines bestimmten Textes eine Häufigkeitsanalyse für die Zeichen machen und so einige der Buchstaben vorab entschlüsseln.

Aufgabe: Maximumbestimmung

Schreibe ein Java-Methode, mit der von einem gegebenen Zahlenfeld das Maximum bestimmt werden kann.

Untersuche das Laufzeitverhalten in Abhängigkeit der Feldlänge für den best case und den worst case.

```
import java.util.Random;
public class MaximumBestimmung{
    int n;
    int[] feld;
    //Zufallsliste der Länge n erzeugen
    public void listeErzeugen(int n){
        this.n = n;
        feld = new int[n];
        Random zg = new Random();

        for (int i = 0; i < n; i++){
            feld[i] = zg.nextInt(100);
            System.out.println(feld[i] );
        }
    }

    public void maximum(){
        int max = feld[0];

        for(int i = 1; i<n; i++){
            if(feld[i]>max){
                max = feld[i];
            }
        }

        System.out.println("Maximum :" + max);
    }

}
```

Das Feld wird n -1 mal durchlaufen; bei Bedarf ist eine Wertzuweisung max=feld[i] notwendig.

best case:

Die Liste ist absteigend sortiert. Man erhält ein lineares Laufzeitverhalten.

worst case:

Die Liste ist aufsteigend sortiert; in jedem Schritt ist eine neue Wertzuweisung von max notwendig. Daraus folgt ebenfalls lineares Laufzeitverhalten allerdings etwas langsamer als im best case.