Aufgabe 2

(a) Argumentieren Sie, warum man das Maximum von n Zahlen nicht mit weniger als $n - 1$ Vergleichen bestimmen kann.

> Wenn die n Zahlen in einem unsortierten Zustand vorliegen, müssen wir alle Zahlen betrachten, um das Maximum zu finden. Wir brauchen dazu $n - 1$ und nicht n Vergleiche, da wir die erste Zahl zu Beginn des Algorithmus als Maximum definieren und anschließend die verbleibenden Zahlen $n - 1$ mit dem aktuelle Maximum vergleichen.

(b) Geben Sie einen Algorithmus im Pseudocode an, der das Maximum eines Feldes der Länge n mit genau $n - 1$ Vergleichen bestimmt.

```java
  public static int bestimmeMaximum(int[] a) {
    int max = a[0];
    for (int i = 1; i < a.length; i++) {
      if (a[i] > max) {
        max = a[i];
      }
    }
    return max;
  }
```

Code-Beispiel auf Github ansehen:
src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java

(c) Wenn man das Minimum und das Maximum von n Zahlen bestimmen will, dann kann das natürlich mit $2n - 2$ Vergleichen erfolgen. Zeigen Sie, dass man bei jedem beliebigen Feld mit deutlich weniger Vergleichen auskommt, wenn man die beiden Werte statt in zwei separaten Durchläufen in einem Durchlauf geschickt bestimmt.

```java
  /**
   * Diese Methode ist nicht optimiert. Es werden 2n - 2 Vergleiche benötigt.
   *
   * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum gesucht
   *          werden soll.
   *
   * @return Ein Feld mit zwei Einträgen. Der erste Einträg enthält das Minimum,
   *         der zweite Eintrag das Maximum.
   */
  public static int[] minMaxNaiv(int[] a) {
    int max = a[0];
    int min = a[0];
    for (int i = 1; i < a.length; i++) {
      if (a[i] > max) {
        max = a[i];
      }
      if (a[i] < min) {
        max = a[i];
      }
    }
```

```
35      return new int[] { min, max };
36    }
37
38    /**
39     * Diese Methode ist optimiert. Es werden immer zwei Zahlen paarweise
40     * betrachtet. Die Anzahl der Vergleiche reduziert sich auf 3n/2 + 2 bzw.
41     * 3(n-1)/2 + 4 bei einer ungeraden Anzahl an Zahlen im Feld.
42     *
43     * nach <a href=
44     * "https://www.techiedelight.com/find-minimum-maximum-element-array-using-minimum-comparisons/">techiedelight.com</a>
45     *
46     * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum gesucht
47     *          werden soll.
48     *
49     * @return Ein Feld mit zwei Einträgen. Der erste Einträg enthält das Minimum,
50     *         der zweite Eintrag das Maximum.
51     */
52    public static int[] minMaxIterativPaarweise(int[] a) {
53      int max = Integer.MIN_VALUE, min = Integer.MAX_VALUE;
54      int n = a.length;
55
56      boolean istUngerade = (n & 1) == 1;
57      if (istUngerade) {
58        n--;
59      }
60
61      for (int i = 0; i < n; i = i + 2) {
62        int maximum, minimum;
63
64        if (a[i] > a[i + 1]) {
65          minimum = a[i + 1];
66          maximum = a[i];
67        } else {
68          minimum = a[i];
69          maximum = a[i + 1];
70        }
71
72        if (maximum > max) {
73          max = maximum;
74        }
75
76        if (minimum < min) {
77          min = minimum;
78        }
79      }
80
81      if (istUngerade) {
82        if (a[n] > max) {
83          max = a[n];
84        }
85
86        if (a[n] < min) {
87          min = a[n];
88        }
89      }
90      return new int[] { min, max };
```

```java
  }

  /**
   * Diese Methode ist nach dem Teile-und-Herrsche-Prinzip optimiert. Er
   * funktioniert so ähnlich wie der Mergesort.
   *
   * nach <a href=
   * "https://www.enjoyalgorithms.com/blog/find-the-minimum-and-maximum-value-in-an-array">enjoyalgorithms.com</a>
   *
   * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum
   *          gesucht werden soll.
   * @param l Die linke Grenze.
   * @param r Die rechts Grenze.
   *
   * @return Ein Feld mit zwei Einträgen. Der erste Einträg enthält das Minimum,
   *         der zweite Eintrag das Maximum.
   */
  int[] minMaxRekursiv(int[] a, int l, int r) {
    int max, min;
    if (l == r) {
      max = a[l];
      min = a[l];
    } else if (l + 1 == r) {
      if (a[l] < a[r]) {
        max = a[r];
        min = a[l];
      } else {
        max = a[l];
        min = a[r];
      }
    } else {
      int mid = l + (r - l) / 2;
      int[] lErgebnis = minMaxRekursiv(a, l, mid);
      int[] rErgebnis = minMaxRekursiv(a, mid + 1, r);
      if (lErgebnis[0] > rErgebnis[0]) {
        max = lErgebnis[0];
      } else {
        max = rErgebnis[0];
      }
      if (lErgebnis[1] < rErgebnis[1]) {
        min = lErgebnis[1];
      } else {
        min = rErgebnis[1];
      }
    }
    int[] ergebnis = { max, min };
    return ergebnis;
  }

}
```

Code-Beispiel auf Github ansehen:
src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java