

## 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 aktuellen Maximum vergleichen.

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

```
5 public static int bestimmeMaximum(int[] a) {
6     int max = a[0];
7     for (int i = 1; i < a.length; i++) {
8         if (a[i] > max) {
9             max = a[i];
10        }
11    }
12    return max;
13 }
```

Code-Beispiel auf Github ansehen:  
[src/main/java/org/bschlangaul/examen/examen\\_66115/jahr\\_2021/fruehjahr/MinimumMaximum.java](https://github.com/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.

```
15 /**
16  * Diese Methode ist nicht optimiert. Es werden  $2n - 2$  Vergleiche
17  * benötigt.
18  *
19  * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem
20  * ↪ Maximum gesucht
21  * ↪ werden soll.
22  *
23  * @return Ein Feld mit zwei Einträgen. Der erste Eintrag enthält
24  * ↪ das Minimum,
25  * ↪ der zweite Eintrag das Maximum.
26  */
27 public static int[] minMaxNaiv(int[] a) {
28     int max = a[0];
29     int min = a[0];
30     for (int i = 1; i < a.length; i++) {
31         if (a[i] > max) {
32             max = a[i];
33         }
34         if (a[i] < min) {
35             min = a[i];
36         }
37     }
38     return new int[] { min, max };
39 }
```

```

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 Eintrag 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 };

```

```

91     }
92
93     /**
94     * Diese Methode ist nach dem Teile-und-Herrsche-Prinzip optimiert.
95     ↪ Er
96     * funktioniert so ähnlich wie der Mergesort.
97     *
98     * nach <a href=
99     * "https://www.enjoyalgorithms.com/blog/find-the-minimum-and-
100     ↪ maximum-value-in-an-array">enjoyalgorithms.com</a>
101     *
102     * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem
103     ↪ Maximum
104     *          gesucht werden soll.
105     * @param l Die linke Grenze.
106     * @param r Die rechts Grenze.
107     *
108     * @return Ein Feld mit zwei Einträgen. Der erste Eintrag enthält
109     ↪ das Minimum,
110     *          der zweite Eintrag das Maximum.
111     */
112
113     int[] minMaxRekursiv(int[] a, int l, int r) {
114         int max, min;
115         if (l == r) {
116             max = a[l];
117             min = a[l];
118         } else if (l + 1 == r) {
119             if (a[l] < a[r]) {
120                 max = a[r];
121                 min = a[l];
122             } else {
123                 max = a[l];
124                 min = a[r];
125             }
126         } else {
127             int mid = l + (r - l) / 2;
128             int[] lErgebnis = minMaxRekursiv(a, l, mid);
129             int[] rErgebnis = minMaxRekursiv(a, mid + 1, r);
130             if (lErgebnis[0] > rErgebnis[0]) {
131                 max = lErgebnis[0];
132             } else {
133                 max = rErgebnis[0];
134             }
135             if (lErgebnis[1] < rErgebnis[1]) {
136                 min = lErgebnis[1];
137             } else {
138                 min = rErgebnis[1];
139             }
140         }
141         int[] ergebnis = { max, min };
142         return ergebnis;
143     }
144 }

```

Code-Beispiel auf Github ansehen:  
src/main/java/org/bschlangaul/examen/examen\_66115/jahr\_2021/fruehjahr/MinimumMaximum.java