

Aufgabe 5

Eine Folge von Zahlen ist eine *odd-ascending-even-descending*-Folge, wenn gilt:

Zunächst enthält die Folge alle Schlüssel, die *ungerade* Zahlen sind, und diese Schlüssel sind aufsteigend sortiert angeordnet. Im Anschluss daran enthält die Folge alle Schlüssel, die *gerade* Zahlen sind, und diese Schlüssel sind absteigend sortiert angeordnet.

- (a) Geben Sie die Zahlen 10, 3, 11, 20, 8, 4, 9 als *odd-ascending-even-descending*-Folge an.

3, 9, 11, 20, 10, 8, 4

- (b) Geben Sie einen Algorithmus (z. B. in Pseudocode oder Java) an, der für eine *odd-ascending-even-descending*-Folge F gegeben als Feld und einem Schlüsselwert S prüft, ob S in F vorkommt und `true` im Erfolgsfall und ansonsten `false` liefert. Dabei soll der Algorithmus im Worst-Case eine echt bessere Laufzeit als Linearzeit (in der Größe der Arrays) haben. Erläutern Sie Ihren Algorithmus und begründen Sie die Korrektheit.

Bei dem Algorithmus handelt es sich um einen leicht abgewandelten Code, der eine „klassische“ binären Suche implementiert.

```
5 public static boolean suche(int[] feld, int schlüssel) {
6     int links = 0, rechts = feld.length - 1;
7     boolean istGerade = schlüssel % 2 == 0;
8     while (links <= rechts) {
9         int mitte = links + (rechts - links) / 2;
10        if (feld[mitte] == schlüssel) {
11            return true;
12        }
13        // Verschiebe die linke Grenze nach rechts, wenn die gesuchte
14        // Zahl gerade ist und die Zahl in der Mitte größer als die
15        // gesuchte Zahl ist oder wenn die gesuchte Zahl ungerade ist
16        // und die Zahl in der Mitte kleiner.
17        if ((istGerade && feld[mitte] > schlüssel) || (!istGerade &&
18            ↪ feld[mitte] < schlüssel)) {
19            // nach rechts verschieben
20            links = mitte + 1;
21        } else {
22            // nach links verschieben
23            rechts = mitte - 1;
24        }
25    }
26    return false;
}
```

github: raw

- (c) Erläutern Sie schrittweise den Ablauf Ihres Algorithmus für die Folge 1, 5, 11, 8, 4, 2 und den Suchschlüssel 4.

Die erste Zeile der Methode `suche` initialisiert die Variable `links` mit 0 und `rechts` mit 5. Da `links` kleiner ist als `rechts`, wird die `while`-Schleife betreten und die Variable `mitte` auf 2 gesetzt. Da der gesuchte Schlüssel gerade ist und `feld[2]` 11 ist, also größer, wird in den `true`-Block der `if`-Bedingung besprungen und die Variable `links` auf 3 gesetzt.

Zu Beginn des 2. Durchlaufs der `while`-Schleife ergeben sich folgende Werte: `links`: 3 `mitte`: 4 `rechts`: 5.

In der anschließenden Bedingten Anweisung wird die `while`-Schleife verlassen und `true` zurückgegeben, da mit `feld[4]` der gewünschte Schlüssel gefunden wurde.

- (d) Analysieren Sie die Laufzeit Ihres Algorithmus für den Worst-Case, geben Sie diese in \mathcal{O} -Notation an und begründen Sie diese.

Die Laufzeit des Algorithmus ist in der Ordnung $\mathcal{O}(\log_2 n)$.

Im schlechtesten Fall muss nicht die gesamte Folge durchsucht werden. Nach dem ersten Teilen der Folge bleiben nur noch $\frac{n}{2}$ Elemente, nach dem zweiten Schritt $\frac{n}{4}$, nach dem dritten $\frac{n}{8}$ usw. Allgemein bedeutet dies, dass im i -ten Durchlauf maximal $\frac{n}{2^i}$ Elemente zu durchsuchen sind. Entsprechend werden $\log_2 n$ Schritte benötigt.

Kompletter Code

```

3  public class UngeradeGerade {
4
5      public static boolean suche(int[] feld, int schlüssel) {
6          int links = 0, rechts = feld.length - 1;
7          boolean istGerade = schlüssel % 2 == 0;
8          while (links <= rechts) {
9              int mitte = links + (rechts - links) / 2;
10             if (feld[mitte] == schlüssel) {
11                 return true;
12             }
13             // Verschiebe die linke Grenze nach rechts, wenn die gesuchte
14             // Zahl gerade ist und die Zahl in der Mitte größer als die
15             // gesuchte Zahl ist oder wenn die gesuchte Zahl ungerade ist
16             // und die Zahl in der Mitte kleiner.
17             if ((istGerade && feld[mitte] > schlüssel) || (!istGerade &&
18                 ↪ feld[mitte] < schlüssel)) {
19                 // nach rechts verschieben
20                 links = mitte + 1;
21             } else {
22                 // nach links verschieben
23                 rechts = mitte - 1;
24             }
25         }
26         return false;
27     }
28
29     public static void main(String[] args) {
30         System.out.println(suche(new int[] { 1, 5, 11, 8, 4, 2 }, 4));
31     }

```

Test-Code

```
3  import static org.junit.Assert.assertEquals;
4  import org.junit.Test;
5
6  public class UngeradeGeradeTest {
7
8      private void assertSucheUnGerade(int[] feld, int suche, boolean ergebnis)
9          ↪ {
10         assertEquals(ergebnis, UngeradeGerade.suche(feld, suche));
11     }
12
13     @Test
14     public void assertSucheUnGerade() {
15         int[] feld = new int[] { 1, 3, 5, 7, 9, 10, 8, 6, 4, 2 };
16         assertSucheUnGerade(feld, 4, true);
17         assertSucheUnGerade(feld, 11, false);
18         assertSucheUnGerade(feld, 0, false);
19         assertSucheUnGerade(feld, 3, true);
20     }
21 }
```