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.

```java
public static boolean suche(int[] feld, int schlüssel) {
  int links = 0, rechts = feld.length - 1;
  boolean istGerade = schlüssel % 2 == 0;
  while (links <= rechts) {
    int mitte = links + (rechts - links) / 2;
    if (feld[mitte] == schlüssel) {
      return true;
    }
    // Verschiebe die linke Grenze nach rechts, wenn die gesuchte
    // Zahl gerade ist und die Zahl in der Mitte größer als die
    // gesuchte Zahl ist oder wenn die gesuchte Zahl ungerade ist
    // und die Zahl in der Mitte kleiner.
    if ((istGerade && feld[mitte] > schlüssel) || (!istGerade &&
    ↪  feld[mitte] < schlüssel)) {
      // nach rechts verschieben
      links = mitte + 1;
    } else {
      // nach links verschieben
      rechts = mitte - 1;
    }
  }
  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` aus 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.

text

Kompletter Code

```java
public class UngeradeGerade {

  public static boolean suche(int[] feld, int schlüssel) {
    int links = 0, rechts = feld.length - 1;
    boolean istGerade = schlüssel % 2 == 0;
    while (links <= rechts) {
      int mitte = links + (rechts - links) / 2;
      if (feld[mitte] == schlüssel) {
        return true;
      }
      // Verschiebe die linke Grenze nach rechts, wenn die gesuchte
      // Zahl gerade ist und die Zahl in der Mitte größer als die
      // gesuchte Zahl ist oder wenn die gesuchte Zahl ungerade ist
      // und die Zahl in der Mitte kleiner.
      if ((istGerade && feld[mitte] > schlüssel) || (!istGerade &&
       ↪  feld[mitte] < schlüssel)) {
        // nach rechts verschieben
        links = mitte + 1;
      } else {
        // nach links verschieben
        rechts = mitte - 1;
      }
    }
    return false;
  }

  public static void main(String[] args) {
    System.out.println(suche(new int[] { 1, 5, 11, 8, 4, 2 }, 4));
  }
}
```

github: raw

Test

```java
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class UngeradeGeradeTest {

  private void assertSucheUnGerade(int[] feld, int suche, boolean ergebnis)
   ↪  {
    assertEquals(ergebnis, UngeradeGerade.suche(feld, suche));
```

```
10      }
11
12      @Test
13      public void assertSucheUnGerade() {
14        int[] feld = new int[] { 1, 3, 5, 7, 9, 10, 8, 6, 4, 2 };
15        assertSucheUnGerade(feld, 4, true);
16        assertSucheUnGerade(feld, 11, false);
17        assertSucheUnGerade(feld, 0, false);
18        assertSucheUnGerade(feld, 3, true);
19      }
20
21    }
```