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 class UngeradeGerade {

  /**
   * Eine klassische Binäre Suche, die in einem sortierten Feld eine
   * Zahl findet. Kann so nicht auf die geforderte
   * ungerade-aufsteigend-gerade-absteigend Folge angewendet werden.
   *
   * @param feld Ein aufsteigend sortiertes Feld.
   * @param suche Eine Zahl nach der im Feld gesucht werden soll.
   *
   * @return Wahr, wenn die Zahl im Feld gefunden wurde.
   */
  public static boolean sucheBinär(int[] feld, int suche) {
    int links = 0, rechts = feld.length - 1;
    while (links <= rechts) {
      int mitte = links + (rechts - links) / 2;
      if (feld[mitte] == suche)
        return true;
      if (feld[mitte] < suche)
        links = mitte + 1;
      else
        rechts = mitte - 1;
    }
    return false;
  }

  public static boolean sucheBinärUngeradeGerade(int[] feld, int
  ↪  suche) {
    int links = 0, rechts = feld.length - 1;
    boolean istGerade = suche % 2 == 0;
    while (links <= rechts) {
      int mitte = links + (rechts - links) / 2;
      if (feld[mitte] == suche)
        return true;
      if (
        // 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.
        (istGerade && feld[mitte] > suche)
        ||
        (!istGerade && feld[mitte] < suche)
      )
        // nach rechts verschieben
        links = mitte + 1;
      else
        // nach links verschieben
        rechts = mitte - 1;
    }
    return false;
  }

  /**
   * Suche die Index-Nummer der letzten ungeraden Zahl.
   *
   * @param feld
   * @return Die Index-Nummer der letzten ungeraden Zahl. Ist im Eingabe-Feld
   *         keine ungerade Zahl vorhanden, dann wird -1 ausgegeben.
   */
  public static int sucheBinärLetzteUngerade(int[] feld) {
    int links = 0, rechts = feld.length - 1;
    int mitte = 0;
    while (links <= rechts) {
      mitte = links + (rechts - links) / 2;
      // ist Ungerade
      if (feld[mitte] % 2 != 0)
        links = mitte + 1;
      else
        rechts = mitte - 1;
    }
    if (rechts < links)
      return rechts;
    return mitte;
  }

  public static void main(String[] args) {
    // System.out.println(binarySearch(new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 },
    // 1));
    System.out.println(sucheBinärLetzteUngerade(new int[] { 1, 3, 5, 7, 9, 10, 8, 6, 4, 2 }));

  }
}
```

github: raw

```
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.sucheBinärUngeradeGerade(feld, suche));
  }
```

```
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    private void assertLetzteUngerade(int[] feld, int index) {
22      assertEquals(index,
        ↪  UngeradeGerade.sucheBinärLetzteUngerade(feld));
23    }
24
25    @Test
26    public void letzteUngerade() {
27      assertLetzteUngerade(new int[] { 1, 3, 5, 7, 9, 10, 8, 6, 4, 2 },
        ↪  4);
28      assertLetzteUngerade(new int[] { 1, 3, 5, 7, 9 }, 4);
29      assertLetzteUngerade(new int[] { 10, 8, 6, 4, 2 }, -1);
30      assertLetzteUngerade(new int[] { 1, 10, 8, 6, 4, 2 }, 0);
31      assertLetzteUngerade(new int[] { 1, 49, 10, 8, 6, 4, 2 }, 1);
32      assertLetzteUngerade(new int[] { }, -1);
33    }
34
35  }
```

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

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