Aufgabe 4

Gegeben seien die Standardstrukturen Stapel (Stack) und Schlange (Queue) mit folgenden Standardoperationen:

Stapel	Schlange
`boolean isEmpty()`	`boolean isEmpty()`
`void push(int e)`	`enqueue(int e)`
`int pop()`	`int dequeue()`
`int top()`	`int head()`

Beim Stapel gibt die Operation `top()` das gleiche Element wie `pop()` zurück, bei der Schlange gibt `head()` das gleiche Element wie `dequeue()` zurück. Im Unterschied zu `pop()`, beziehungsweise `dequeue()`, wird das Element bei `top()` und `head()` nicht aus der Datenstruktur entfernt.

(a) Geben Sie in Pseudocode einen Algorithmus `sort(Stapel s)` an, der als Eingabe einen Stapel `s` mit `n` Zahlen erhält und die Zahlen in `s` sortiert. (Sie dürfen die Zahlen wahlweise entweder aufsteigend oder absteigend sortieren.) Verwenden Sie als Hilfsdatenstruktur ausschließlich eine Schlange `q`. Sie erhalten volle Punktzahl, wenn Sie außer `s` und `q` keine weiteren Variablen benutzen. Sie dürfen annehmen, dass alle Zahlen in `s` verschieden sind.

```
q := neue Schlange
while s not empty:
    q.enqueue(S.pop())
while q not empty:
    while s not empty and s.top() < q.head():
        q.enqueue(s.pop())
    s.push(q.dequeue)
```

Als Java-Code

```java
/**
 * So ähnlich wie der <a href=
 * "https://www.geeksforgeeks.org/sort-stack-using-temporary-
↪ stack/">Stapel-Sortiert-Algorithmus
 * der nur Stapel verwendet</a>, nur mit einer Warteschlange.
 *
 * @param s Der Stapel, der sortiert wird.
 */
public static void sort(Stapel s) {
  Schlange q = new Schlange();
  while (!s.isEmpty()) {
    q.enqueue(s.pop());
  }
  while (!q.isEmpty()) {
    // Sortiert aufsteigend. Für absteigend einfach das „kleiner"
    // Zeichen umdrehen.
    while (!s.isEmpty() && s.top() < q.head()) {
      q.enqueue(s.pop());
    }
    s.push(q.dequeue());
  }
}
```

Code-Beispiel auf Github ansehen:
src/main/java/org/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/Sort.java

Klasse Sort

```java
public class Sort {

  /**
   * So ähnlich wie der <a href=
   * "https://www.geeksforgeeks.org/sort-stack-using-temporary-
↪  stack/">Stapel-Sortiert-Algorithmus
   * der nur Stapel verwendet</a>, nur mit einer Warteschlange.
   *
   * @param s Der Stapel, der sortiert wird.
   */
  public static void sort(Stapel s) {
    Schlange q = new Schlange();
    while (!s.isEmpty()) {
      q.enqueue(s.pop());
    }
    while (!q.isEmpty()) {
      // Sortiert aufsteigend. Für absteigend einfach das „kleiner"
      // Zeichen umdrehen.
      while (!s.isEmpty() && s.top() < q.head()) {
        q.enqueue(s.pop());
      }
      s.push(q.dequeue());
    }
  }

  public static Stapel stapelBefüllen(int[] zahlen) {
    Stapel s = new Stapel();
    for (int i : zahlen) {
      s.push(i);
    }
    return s;
  }

  public static void zeigeStapel(Stapel s) {
    while (!s.isEmpty()) {
      System.out.print(s.pop() + " ");
    }
    System.out.println();
  }

  public static void main(String[] args) {
    Stapel s1 = stapelBefüllen(new int[] { 4, 2, 1, 5, 3 });
    sort(s1);
    zeigeStapel(s1);

    Stapel s2 = stapelBefüllen(new int[] { 1, 2, 6, 3, 9, 11, 4 });
    sort(s2);
    zeigeStapel(s2);
  }
}
```

Code-Beispiel auf Github ansehen:
src/main/java/org/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/Sort.java

Klasse Schlange

```
3  public class Schlange {
4
5    public Element head;
6
7    public Schlange() {
8      head = null;
9    }
10
11   public int head() {
12     if (head.getNext() == null) {
13       return head.getValue();
14     }
15     Element element = head;
16     Element previous = head;
17     while (element.getNext() != null) {
18       previous = element;
19       element = element.getNext();
20     }
21     element = previous.getNext();
22     return element.getValue();
23   }
24
25   /**
26    * @param value Eine Zahl, die zur Schlange hinzugefügt werden soll.
27    */
28   public void enqueue(int value) {
29     Element element = new Element(value);
30     element.setNext(head);
31     head = element;
32   }
33
34   /**
35    * @return Das Element oder null, wenn der Schlange leer ist.
36    */
37   public int dequeue() {
38     if (head.getNext() == null) {
39       int result = head.getValue();
40       head = null;
41       return result;
42     }
43     Element element = head;
44     Element previous = null;
45     while (element.getNext() != null) {
46       previous = element;
47       element = element.getNext();
48     }
49     element = previous.getNext();
50     previous.setNext(null);
51     return element.getValue();
52   }
53
54   /**
55    * @return Wahr wenn der Schlange leer ist.
56    */
57   public boolean isEmpty() {
58     return head == null;
59   }
60
61   public static void main(String[] args) {
```

```
    Schlange s = new Schlange();
    s.enqueue(1);
    s.enqueue(2);
    s.enqueue(3);
    System.out.println(s.head());
    System.out.println(s.dequeue());
    System.out.println(s.head());
    System.out.println(s.dequeue());
    System.out.println(s.head());
    System.out.println(s.dequeue());
  }

}
```

Code-Beispiel auf Github ansehen:
src/main/java/org/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/Schlange.java

Klasse Element

```
public class Element {
  public int value;

  public Element next;

  public Element() {
    this.next = null;
  }

  public Element(int value, Element element) {
    this.value = value;
    this.next = element;
  }

  public Element(int value) {
    this.value = value;
    this.next = null;
  }

  public int getValue() {
    return value;
  }

  public Element getNext() {
    return next;
  }

  public void setNext(Element element) {
    next = element;
  }

}
```

Code-Beispiel auf Github ansehen:
src/main/java/org/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/Element.java

Test-Klasse

```java
import static org.junit.Assert.*;

import org.junit.Test;

public class TestCase {

  @Test
  public void testeStapel() {
    Stapel s = new Stapel();
    s.push(1);
    s.push(2);
    s.push(3);

    assertEquals(false, s.isEmpty());

    assertEquals(3, s.top());
    assertEquals(3, s.pop());

    assertEquals(2, s.top());
    assertEquals(2, s.pop());

    assertEquals(1, s.top());
    assertEquals(1, s.pop());
    assertEquals(true, s.isEmpty());
  }

  @Test
  public void testeSchlange() {
    Schlange s = new Schlange();
    s.enqueue(1);
    s.enqueue(2);
    s.enqueue(3);

    assertEquals(false, s.isEmpty());

    assertEquals(1, s.head());
    assertEquals(1, s.dequeue());

    assertEquals(2, s.head());
    assertEquals(2, s.dequeue());

    assertEquals(3, s.head());
    assertEquals(3, s.dequeue());
    assertEquals(true, s.isEmpty());
  }

}
```

Code-Beispiel auf Github ansehen:
src/test/java/org/bschlangaul/examen/examen_66115/jahr_2015/fruehjahr/schlange/TestCase.java

(b) Analysieren Sie die Laufzeit Ihrer Methode in Abhängigkeit von n.

Zeitkomplexität: $\mathcal{O}(n^2)$, da es zwei ineinander verschachtelte `while`-Schleifen gibt, die von der Anzahl der Elemente im Stapel abhängen.