Algorithmen und Datenstrukturen 1

Serie 2

1. a) Ich gehe davon aus das n unbekannt ist.[1]
   Ich setze zwei Zeiger auf den Listenanfang. Den ersten Zeiger erhöhe ich mit jedem Schleifendurchlauf, den zweiten nur bei jedem zweiten Mal. Wenn der erste Zeiger das Listenende erreicht hat, steht der zweite Zeiger auf dem mittleren Element. Ich brauche dafür $n + \frac{n}{2}$ Schritte.

   b) Ich setze einen Zeiger auf den Listenanfang, einen zweiten auf das Listenende. Bei jedem Schleifendurchlauf wird der erste Zeiger auf das nächste Element gesetzt, der zweite auf das vorherige. Sobald die beiden Zeiger auf das selbe Element verweisen, ist das Mittlere gefunden. Es werden hierfür nur n Schritte benötigt.

2. a) Die Höhe der Liste ist $1 + \lceil \log_3(n) \rceil = 1 + \lceil \log_3(243) \rceil = 6$.
   Somit hat die Skiptliste eine Zeigeranzahl von:

$$\begin{aligned} z_n &= 243 + 1 + \left\lceil \frac{243}{3^1} \right\rceil + 1 + \left\lceil \frac{243}{3^2} \right\rceil + 1 + \left\lceil \frac{243}{3^3} \right\rceil + 1 + \left\lceil \frac{243}{3^4} \right\rceil + 1 + \left\lceil \frac{243}{3^5} \right\rceil + 1 \\ &= 243 + 81 + 27 + 9 + 3 + 1 + 6 \\ &= 370 \end{aligned}$$

   Die perfekte Skipliste hätte eine Höhe von 8 und eine Zeigeranzahl von $z_n = 488$.

   b) Die Komplexität der Suche auf der perfekten Skipliste ist $O(\log_2(n))$. Demnach ist sie für die modifizierte Skipliste $O(\log_3(n))$.

[1] falls n bekannt sein sollte, kann man das Problem schneller lösen indem man die Liste einfach iterativ bis zum $\frac{n}{2}$-ten Element durchläuft.

3. Hier die Quelltexte:

a)
```
/*****************************************************************
 * ADS1 Serie 02 Aufgabe 3.a)
 * Implementierung einer Verketteten Liste
 *
 * Arne Brutschy, 8964813, Uni Leipzig
 *****************************************************************/
public class VerketteteListe {
  ListenElement  head;
  ListenElement  tail;

  public VerketteteListe() {
    this.head= new ListenElement();
    this.tail= new ListenElement();
    this.head.setNext(tail);
    this.tail.setNext(head);
  }

  public VerketteteListe(VerketteteListe Init) {
    this.head= new ListenElement();
    this.tail= new ListenElement();

    ListenElement i= Init.head.getNext();
    ListenElement j= this.head;
    while (!i.equals(Init.tail)) {
      j.setNext(new ListenElement(i.getData()));
      j= j.getNext();
      i= i.getNext();
    }
    j.setNext(this.tail);
    this.tail.setNext(j);

    return this;
  }

  public VerketteteListe Insert(Object data) {
    ListenElement neues= new ListenElement(data);

    neues.setNext(this.head.getNext());
    this.head.setNext(neues);
    return this;
  }

  public VerketteteListe Delete(Object data) {
    ListenElement  i= this.head.getNext();
    ListenElement  last= this.head;

    while (!i.equals(this.tail)) {
      if (i.getData().equals(Object)) {
```

```
        last.setNext(i.getNext());
        return this;
      }

      last= i;
      i= i.getNext();
    }
    return this;
  }

  public boolean Search(Object data) {
    ListenElement  i= this.head.getNext();
    ListenElement  last= this.head;

    while (!i.equals(this.tail)) {
      if (i.getData().equals(Object))
        return true;

      last= i;
      i= i.getNext();
    }
    return false;
  }

  public boolean Empty() {
    return this.tail.getNext().equals(this.head);
  }
}
```

```
/*****************************************************************
 * ADS1 Serie 02 Aufgabe 3.
 * Implementierung eines Elements einer Liste
 *
 * Arne Brutschy, 8964813, Uni Leipzig
 *****************************************************************/
class ListenElement {
  private ListenElement next;
  private Object        data;


  public void Listenelement() {
    this.data= null;
    this.next= null;
  }

  public void Listenelement(Object data) {
    this.data= data;
    this.next= null;
  }

  public void setData(Object data) {
    this.data= data;
  }

  public Object getData() {
    return this.data;
  }

  public void setNext(ListenElement next) {
    this.next= next;
  }

  public ListenElement getNext() {
    return this.next;
  }
}
```

b)
```
/*****************************************************************
 * ADS1 Serie 02 Aufgabe 3.b)
 * Implementierung einer Queue
 *
 * Arne Brutschy, 8964813, Uni Leipzig
 *****************************************************************/
public class Queue extends Liste {

  public Queue() {
    super();
  }

  public Queue(Queue Init) {
    super(Init);
  }

  public Queue Enqueue(Object Element) {
    this.tail.getNext().setNext(new ListenElement(Element))
    this.tail.setNext(this.tail.getNext().getNext());
    this.tail.getNext().setNext(this.tail);
    return this;
  }

  public Queue Dequeue() {
    this.head.setNext(this.head.getNext().getNext());
    return this;
  }

  public Object Top() {
    return this.head.getNext().getData();
  }
}
```

c)
```
/*****************************************************************
 * ADS1 Serie 02 Aufgabe 3.c)
 * Implementierung eines Stacks, erweitert v. Liste
 *
 * Arne Brutschy, 8964813, Uni Leipzig
 *****************************************************************/
public class Stack extends Liste {

  public Stack() {
    super();
  }

  public Stack(Stack Init) {
    super(Init);
  }

  public Stack Push(Object Element) {
    return super.Insert(Element);
  }

  public Stack Pop() {
    this.head.setNext(this.head.getNext().getNext())
    return this;
  }

  public Object Top() {
    return this.head.getNext().getData();
  }
}
```