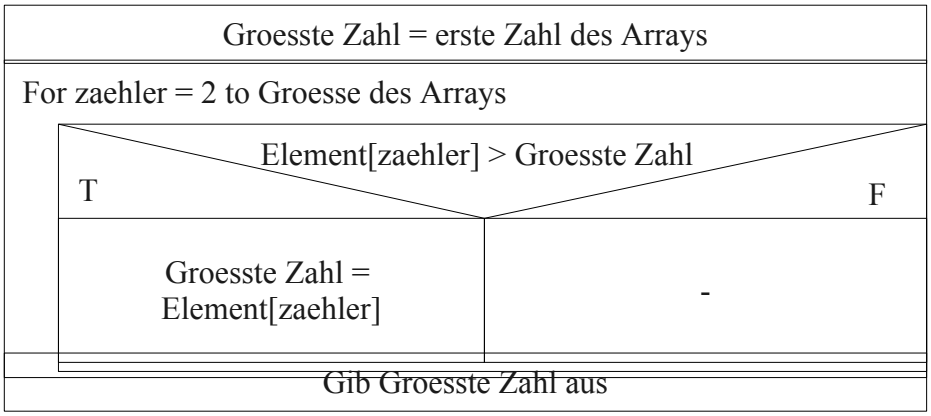
Aufgaben und Lösungen

Vorlesung Algorithmen und Datenstrukturen – 2. Semester an der DHBW Stuttgart (2011)
Dozent: Marcus Schmidt

Aufgabe 1:

Gegeben ist ein Array mit ganzen Zahlen und mindestens 2 Elementen. Entwerfe einen Algorithmus, der die größte Zahl im Array ermittelt und ausgibt. Stell den Algorithmus als Struktogramm und in Pseudo-Code dar.

Lösung Struktogramm:



Lösung Pseudo-Code:

Groesste Zahl = erste Zahl des Arrays.
for zaehler = 2 to Groesse des Arrays do:
    if Arrayelement [zaehler] > Groesste Zahl then:
        Groesste Zahl = Arrayelement [zaehler].
Gib Groesste Zahl aus.

Aufgabe 2:

Erstelle ein Javaprogramm, das eine statische rekursive Methode zur Berechnung der Fakultät besitzt. Das Programm wird mit einer positiven Ganzzahl als Parameter aufgerufen und soll die Fakultät berechnen und ausgeben. Fehlerbehandlung muss nicht implementiert werden, als Datentyp sollte long verwendet werden, Konvertierung String nach long mit: Long.parseLong(String).

Lösung:

```
public class Fakultaet {

  public static long fak(long n) {
    if (n == 0) {
      return 1;
    } else {
      return n * fak(n-1);
    }
```

```
  }

  public static void main(String[] args) {
    System.out.println(" Fakultaet von "+args[0]+":
"+fak(Long.parseLong(args[0])));
  }

}
```

Aufgabe 3:

Welche Komplexität haben jeweils die Methoden addFirst, getFirst, removeFirst, addLast, getLast, und removeLast?

Lösung:

Die Methoden addFirst, getFirst, removeFirst haben die Komplexität: O(1). Die Methoden addLast, getLast, removeLast: O(n).

Aufgabe 4:

Erstelle ein Javaprogramm SimpleListTest, das eine SimpleList erzeugt und ihr mit addFirst die Strings „Zwei“ und „Eins“ und danach mit addLast „Drei“ und „Vier“ hinzufügt. Dann soll so oft removeFirst aufgerufen und das Ergebnis auf der Konsole ausgegeben werden bis die Liste leer ist.

Lösung:

```
/**
 * Testklasse fuer eine einfach verkettete Liste.
 * @author marcus
 */
public class SimpleListTest {

  /**
   * Hauptmethode.
   * @param args Kommandozeilenparameter.
   */
  public static void main(String args[]) {

    SimpleList lst = new SimpleList();

    lst.addFirst("Zwei");
    lst.addFirst("Eins");
    lst.addLast("Drei");
    lst.addLast("Vier");

    while (!lst.isEmpty()) {
      System.out.println((String) lst.removeFirst());
    }

  }
}
```

Aufgabe 5:

Schreibe die Klassen DoubleListNode, DoubleList (implements ListIF) und DoubleListTest für eine doppelt verkettete Liste analog zur den Klassen für eine einfach verkettete Liste.

Lösung:

```
public class DoubleListNode {

  private Object obj;
  private DoubleListNode prev;
  private DoubleListNode next;

  public DoubleListNode(Object o, DoubleListNode p, DoubleListNode n) {
    obj = o;
    prev = p;
    next = n;
  }

  public void setElement(Object o) {
    obj = o;
  }

  public Object getElement() {
    return obj;
  }

  public void setNext(DoubleListNode n) {
    next = n;
  }

  public DoubleListNode getNext() {
    return next;
  }

  public void setPrevious(DoubleListNode p) {
    prev = p;
  }

  public DoubleListNode getPrevious() {
    return prev;
  }

}
```

```
import java.util.NoSuchElementException;

public class DoubleList implements ListIF {

  private DoubleListNode head = null;
  private DoubleListNode tail = null;

  public DoubleList() {
    head = new DoubleListNode(null, null, null);
    tail = new DoubleListNode(null, null, null);
    head.setNext(tail);
    tail.setPrevious(head);
  }
```

```
public void addFirst(Object o) {
  DoubleListNode l = head.getNext();
  DoubleListNode n = new DoubleListNode(o, head, l);
  l.setPrevious(n);
  head.setNext(n);
}

public void addLast(Object o) {
  DoubleListNode l = tail.getPrevious();
  DoubleListNode n = new DoubleListNode(o, l, tail);
  l.setNext(n);
  tail.setPrevious(n);
}

public Object getFirst() throws NoSuchElementException {
  if (isEmpty()) {
    throw new NoSuchElementException();
  }
  return head.getNext().getElement();
}

public Object getLast() throws NoSuchElementException {
  if (isEmpty()) {
    throw new NoSuchElementException();
  }
  return tail.getPrevious().getElement();
}

public Object removeFirst() throws NoSuchElementException {
  if (isEmpty()) {
    throw new NoSuchElementException();
  }
  DoubleListNode n = head.getNext();
  Object o = n.getElement();
  n.getNext().setPrevious(head);
  head.setNext(n.getNext());
  return o;
}

public Object removeLast() throws NoSuchElementException {
  if (isEmpty()) {
    throw new NoSuchElementException();
  }
  DoubleListNode n = tail.getPrevious();
  Object o = n.getElement();
  n.getPrevious().setNext(tail);
  tail.setPrevious(n.getPrevious());
  return o;
}

public int size() {
  int s = 0;
  DoubleListNode n = head;
  while (n.getNext() != tail) {
    s++;
    n = n.getNext();
  }
  return s;
}

public boolean isEmpty() {
  return head.getNext() == tail;
```

```
  }

}
```

```
public class DoubleListTest {

  public static void main(String args[]) {
    DoubleList lst = new DoubleList();
    lst.addFirst("Zwei");
    lst.addFirst("Eins");
    lst.addLast("Drei");
    lst.addLast("Vier");

    while (!lst.isEmpty()) {
      System.out.println((String) lst.removeFirst());
    }
  }
}
```

Aufgabe 6:

Schreibe eine Klasse SimpleListIter, analog zu SimpleListTest, die statt die Liste zu löschen diese nur mit dem Iterator durchläuft und die Elemente ausgibt.

Lösung:

```
public class SimpleListIter {

  public static void main(String args[]) {
    SimpleList lst = new SimpleList();
    lst.addFirst("Zwei");
    lst.addFirst("Eins");
    lst.addLast("Drei");
    lst.addLast("Vier");

    IteratorIF iter = lst.iterator();
    while (iter.hasNext()) {
      System.out.println((String) iter.next());
    }
  }
}
```

Aufgabe 7:

Programmiere eine Klasse ListStack, welche das Interface StackIF implementiert. Als Basis soll die Klasse SimpleList dienen. Schreibe weiterhin eine Klasse ListStackTest, welche auf dem Stack nacheinander die Strings „eins, „zwei“ und „drei“ ablegt und dann in einer Schleife solange die Elemente vom Stack nimmt und ausgibt bis der Stack leer ist.

Lösung:

```
import java.util.NoSuchElementException;
```

```java
/**
 * Implementierung eines Stackspeichers auf Basis einer Liste.
 */
public class ListStack implements StackIF {

  /** Basisliste als Attribut */
  private SimpleList list;

  /**
   * Konstruktor.
   */
  public ListStack() {
    list = new SimpleList();
  }

  /**
   * Ein Element auf den Stapel legen.
   * @param obj Element.
   */
  public void push(Object obj) {
    list.addFirst(obj);
  }

  /**
   * Ein Element vom Stapel holen.
   * @return Element.
   * @throws NoSuchElementException wenn der Stapel leer ist.
   */
  public Object pop() throws NoSuchElementException {
    return list.removeFirst();
  }

  /**
   * Das oberste Element zurueckgeben ohne es vom Stapel
   * zu nehmen.
   * @return Element.
   * @throws NoSuchElementException wenn der Stapel leer ist.
   */
  public Object top() throws NoSuchElementException {
    return list.getFirst();
  }

  /**
   * Pruefen ob der Stapel leer ist.
   * @return true wenn leer.
   */
  public boolean isEmpty() {
    return list.isEmpty();
  }
}
```

```java
/**
 * Demonstration des Stacks.
 */
public class ListStackTest {

  /**
   * Hauptmethode.
   * @param args Kommandozeilenparameter.
```

```
   */
  public static void main(String args[]) {
    ListStack stack = new ListStack();
    stack.push("eins");
    stack.push("zwei");
    stack.push("drei");

    while (!stack.isEmpty()) {
      System.out.println((String) stack.pop());
    }
  }
}
```

Aufgabe 8:

Programmiere eine Klasse ListQueue, welche das Interface QueueIF implementiert. Als Basis soll die Klasse SimpleList dienen. Schreibe weiterhin eine Klasse ListQueueTest, welche der Queue nacheinander die Strings „eins, „zwei“ und „drei“ hinzufuegt und dann in einer Schleife solange die Elemente aus der Queue nimmt und ausgibt bis die Queue leer ist.

Lösung:

```
import java.util.NoSuchElementException;

/**
 * Implementierung einer Warteschlange auf Basis einer
 * Liste.
 */
public class ListQueue implements QueueIF {

  /** Liste als Basis */
  private SimpleList list;

  /**
   * Konstruktor.
   */
  public ListQueue() {
    list = new SimpleList();
  }

  /**
   * Ein Element in die Warteschlange stellen.
   * @param obj Element.
   */
  public void enter(Object obj) {
    list.addFirst(obj);
  }

  /**
   * Ein Element aus der Warteschlange entnehmen.
   * @return Element.
   * @throws NoSuchElementException wenn Queue leer.
   */
  public Object leave() throws NoSuchElementException {
    return list.removeLast();
  }
```

```java
  /**
   * Das vorderste Element der Warteschlange zurueckliefern
   * ohne es aus der Schlange zu entnehmen.
   * @return Element.
   * @throws NoSuchElementException wenn Queue leer.
   */
  public Object front() throws NoSuchElementException {
    return list.getLast();
  }

  /**
   * Pruefen ob die Warteschlange leer ist.
   * @return true wenn leer.
   */
  public boolean isEmpty() {
    return list.isEmpty();
  }
}
```

```java
/**
 * Demoklasse fuer eine Queue.
 */
public class ListQueueTest {

  /**
   * Hauptmethode.
   * @param args Kommandozeilenparameter.
   */
  public static void main(String args[]) {
    ListQueue queue = new ListQueue();
    queue.enter("eins");
    queue.enter("zwei");
    queue.enter("drei");

    while (!queue.isEmpty()) {
      System.out.println((String) queue.leave());
    }
  }
}
```

Aufgabe 9:

Schreibe unter Einsatz der Klasse TreeSet ein Programm Lotto, das die Lottoziehung 6 aus 49 simuliert. D.h. es sollen zufällig 6 verschiedene Lottozahlen zwischen 1 und 49 ermittelt und in der richtigen Reihenfolge ausgegeben werden. Hinweis: Ziehung von Zufallszahlen über die Klasse java.util.Random. Es wird ein Objekt der Klasse instanziert und darauf die Methode nextInt(int obergrenze) aufgerufen, diese liefert int-Werte zwischen 0 und obergrenze-1 zurück.

Lösung:

```java
import java.util.Random;
import java.util.TreeSet;
import java.util.Iterator;

public class Lotto {

  public static void main(String[] args) {
```

```
    Random myRandom = new Random();
    TreeSet<Integer> set = new TreeSet<Integer>();

    while (set.size() < 6)
      set.add(new Integer(myRandom.nextInt(49)+1));

    System.out.println("Die Lottozahlen: ");

    Iterator<Integer> iter = set.iterator();
    while (iter.hasNext())
      System.out.println(iter.next());

  }
}
```

Aufgabe 10:

Schreibe eine Klasse Vokabel. Nach dem Start des Programms sollen beliebig viele Wortpaare Fremdsprache/Deutsch eingegeben werden können. Zuerst wird der Begriff in der Fremdsprache eingegeben, dann in Deutsch. Wird bei der Fremdsprache ein „#“ eingegeben, so wird die Eingabe beendet. Danach wird der Anwender gefragt, wie viele Abfragen das Programm vornehmen soll. Nun fragt das Programm den Anwender ab, in dem es zufällig einen Begriff in der Fremdsprache auswählt und dem Anwender anzeigt. Der Anwender muss daraufhin den deutschen Begriff eingeben und bekommt ein Feedback ob seine Antwort richtig oder falsch war. Bei einer falschen Antwort wird ihm außerdem der korrekte deutsche Begriff angezeigt. Das passiert nun so oft wie der Anwender gewünscht hatte. Dabei kann der gleiche Begriff durchaus mehrfach abgefragt werden. Hilfestellung - alle Schlüssel einer Map in einem Stringarray ablegen geht mit: String[] keys = map.keySet().toArray(new String[map.keySet().size()]);

Lösung:

```
import java.util.HashMap;
import java.util.Random;
import java.util.Scanner;

public class Vokabel {

  public static void main(String[] args) {
    HashMap<String, String> mapVokabel = new HashMap<String, String>();
    Scanner eingabe = new Scanner(System.in);
    Random rnd = new Random();
    int anzahl = 0;
    int korrekt = 0;
    int falsch = 0;
    String fremd;

    System.out.println("*** Vokabeln eingeben ***");
    do {
      System.out.print("Fremdsprache (# = Ende): ");
      fremd = eingabe.next();
      if (!fremd.equals("#")) {
        System.out.print("Deutsch: ");
        String deutsch = eingabe.next();
        anzahl++;
        mapVokabel.put(fremd, deutsch);
      }
    } while (!fremd.equals("#"));

    String[] keys = mapVokabel.keySet().toArray(new String[anzahl]);
    System.out.println("*** Vokabeln abfragen ***");
    System.out.print("Wieviele Abfragen? ");
    int abfragen = eingabe.nextInt();
```

```
    for (int i=0; i < abfragen; i++) {
      int nummer = rnd.nextInt(anzahl);
      String begriff = keys[nummer];
      System.out.println("Bitte uebersetzen Sie "+begriff+": ");
      String uebersetzung = eingabe.next();
      if (uebersetzung.equals(mapVokabel.get(begriff))) {
        korrekt++;
        System.out.println("Das ist richtig!");
      } else {
        falsch++;
        System.out.println("Das leider falsch! Richtig ist:
"+mapVokabel.get(begriff));
      }
    }
    System.out.println("Abgefragt: "+abfragen+" von richtig: "+korrekt+ " und
falsch: "+falsch);
  }


}
```

Aufgabe 11:

Ergänze die Klasse BinaryTree um die privaten Methoden printPreorder, printInorder, printPostorder. Diese sollen den jeweiligen rekursiven Algorithmus umsetzen und System.out.println(n.getElement()) verwenden um das Element auszugeben.

Lösung:

```
/**
 * Preorder-Algorithmus mit Ausgabe.
 */
private void printPreorder(TreeNode n) {
  if (n != nullNode) {
    System.out.println(n.getElement());
    printPreorder(n.getLeft());
    printPreorder(n.getRight());
  }
}

/**
 * Inorder-Algorithmus mit Ausgabe.
 */
private void printInorder(TreeNode n) {
  if (n != nullNode) {
    printInorder(n.getLeft());
    System.out.println(n.getElement());
    printInorder(n.getRight());
  }
}

/**
 * Postorder-Algorithmus mit Ausgabe.
 */
private void printPostorder(TreeNode n) {
  if (n != nullNode) {
    printPostorder(n.getLeft());
    printPostorder(n.getRight());
    System.out.println(n.getElement());
  }
```

```
}
```

Aufgabe 12:

Ergänze die Klasse BinaryTree um die Methode „public void buildTest()“. Diese soll „von Hand“ einen Baum aufbauen wie er in den Beispielen zum Durchlaufen von Bäumen abgebildet ist.

Lösung:

```
public void buildTest() {
  TreeNode a = new TreeNode("A");
  TreeNode b = new TreeNode("B");
  TreeNode c = new TreeNode("C");
  TreeNode d = new TreeNode("D");
  TreeNode e = new TreeNode("E");
  TreeNode f = new TreeNode("F");
  TreeNode g = new TreeNode("G");
  a.setLeft(b);
  a.setRight(c);
  b.setLeft(d);
  b.setRight(e);
  c.setLeft(f);
  c.setRight(g);
  d.setLeft(nullNode);
  d.setRight(nullNode);
  e.setLeft(nullNode);
  e.setRight(nullNode);
  f.setLeft(nullNode);
  f.setRight(nullNode);
  g.setLeft(nullNode);
  g.setRight(nullNode);
  head.setRight(a);
}
```

Aufgabe 13:

Implementiere obigen Algorithmus in einer Methode: “public TreeNode findNode(String s)“. Die Methode soll entweder den gefundenen Knoten zurückliefern oder “null” falls das Element nicht gefunden wurde. Tipp: es wird vermutlich noch eine weitere Methode nötig sein.

Lösung:

```
/**
 * Knotensuche mit rekursivem Algorithmus anstossen.
 * @param s Suchwert.
 * @return Knoten oder null wenn nicht gefunden.
 */
public TreeNode findNode(String s) {
  return findNodeRekursiv(s, head);
}

/**
 * Rekursive Suche.
 * @param s Suchwert.
 * @param n Knoten.
 * @return Knoten oder null wenn nicht gefunden.
```

```
 */
private TreeNode findNodeRekursiv(String s, TreeNode n) {
  if (n.getElement() != null && n.getElement().compareTo(s) > 0) {
    if (n.getLeft() == nullNode) {
      return null;
    } else {
      return findNodeRekursiv(s, n.getLeft());
    }
  } else if (n.getElement() == null || n.getElement().compareTo(s) < 0) {
    if (n.getRight() == nullNode) {
        return null;
      } else {
        return findNodeRekursiv(s, n.getRight());
      }
  } else {
    return n;
  }
}
```

Aufgabe 14:

Schreibe eine Methode „public boolean insert(String s)“ welche den skizzierten Algorithmus zum Einfügen eines Elements in den Suchbaum umsetzt.

Lösung:

```
/**
 * Einfuegen eines Elements in den Suchbaum.
 * @param s einzufuegendes Element.
 * @return true wenn erfolgreich eingefuegt.
 */
public boolean insert(String s) {
  TreeNode parent = head;
  TreeNode node = head.getRight();

  // Baum nach Knoten durchsuchen
  while (node != nullNode) {
    parent = node;
    if (node.getElement().compareTo(s) == 0) {
      return false; // Knoten gefunden, also nicht einfuegen!
    } else if (node.getElement().compareTo(s) > 0) {
      node = node.getLeft();
    } else {
      node = node.getRight();
    }
  }

  // Knoten nicht gefunden, d.h. neuen Knoten erzeugen und am
  // Parent-Knoten einhaengen.
  TreeNode n = new TreeNode(s);
  if (parent.getElement() != null && parent.getElement().compareTo(s) > 0) {
    parent.setLeft(n);
  } else {
    parent.setRight(n);
  }
  n.setLeft(nullNode);
  n.setRight(nullNode);
  return true;
}
```

Aufgabe 15:

Schreibe eine Klasse „BinarySearchTreeTest“ mit einer Main-Methode. In der Methode soll ein neuer Suchbaum angelegt werden und diesem mit insert in der Reihernfolge die Elemente F, C, I, A, E, G, J, D (als Strings) hinzugefügt werden. Prüfe mit einem Preorder-Lauf ob der Baum korrekt aufgebaut wurde. Preorder muss folgende Reihenfolge liefern: F – C – A – E – D – I – G – J. Versuche weiterhin über findNode die Knoten für D, J und B zu finden (der Letzte muss natürlich null liefern).

Lösung:

```
public class BinarySearchTreeTest {

  public static void main(String[] args) {
    BinarySearchTree tree = new BinarySearchTree();

    tree.insert("F");
    tree.insert("C");
    tree.insert("I");
    tree.insert("A");
    tree.insert("E");
    tree.insert("G");
    tree.insert("J");
    tree.insert("D");

    System.out.println("Preorder:");
    tree.traversePreorder();

    System.out.println("Suchen:");
    System.out.println(tree.findNode("D"));
    System.out.println(tree.findNode("J"));
    System.out.println(tree.findNode("B"));
  }

}
```

Aufgabe 16:

Implementiere die lineare Suche in der folgenden Klasse:

```
(s. Folie)
```

Lösung:

```
public static int search(int[] array, int key) {
  for (int i = 0; i < array.length; i++) {
    if (array[i] == key) {
      return i;
    }
  }
  return -1;
}
```

Aufgabe 17:

Analog zur Klasse LineareSuche soll nun eine Klasse BinaereSuche geschrieben werden die analog aufgebaut ist, aber in der search-Methode die binäre Suche implementiert. Es soll hierbei ein nicht rekursiver Ansatz gewählt werden.

Lösung:

```
public static int search(int[] array, int key) {
  int untergrenze = 0;
  int obergrenze = array.length - 1;
  while (untergrenze <= obergrenze) {
    int mitte = (untergrenze + obergrenze) / 2;
    if (array[mitte] == key) {
      return mitte;
    } else if (array[mitte] > key) {
      obergrenze = mitte - 1;
    } else {
      untergrenze = mitte + 1;
    }
  }
  return -1;
}
```

Aufgabe 18:

Implementiere den Bubblesort für ein Array von „int“ Werten. Schreibe in der Klasse „BubbleSort“ auch eine Main- Methode die den Algorithmus mit dem Array {5, 1, 8, 3, 9, 2} testet, d.h. das Array vor dem Sortieren ausgibt, sortiert und dann nochmal ausgibt.

Lösung:

```
public class BubbleSort {

  public void bubbleSort(int[] array) {
    for (int j=array.length-1; j >= 1; j--) {
      for (int i=0 ; i <= j-1; i++) {
        if (array[i] > array[i+1]) {
          int tmp = array[i+1];
          array[i+1] = array[i];
          array[i] = tmp;
        }
      }
    }
  }

  public static void main(String[] args) {
    int[] f = new int[]{5, 1, 8, 3, 9, 2};
    System.out.print("Vor dem Sortieren: ");
    for (int i = 0; i < f.length; i++) {
      System.out.print(f[i]+" ");
    }
    System.out.println("");

    BubbleSort sort = new BubbleSort();
    sort.bubbleSort(f);

    System.out.print("Nach dem Sortieren: ");
    for (int i = 0; i < f.length; i++) {
      System.out.print(f[i]+" ");
    }
    System.out.println("");
  }
```

```
}
```

Aufgabe 19:

Implementiere den SelectionSort für ein Array von „int“ Werten. Schreibe in der Klasse „SelectionSort“ auch eine Main-Methode die den Algorithmus mit dem Array {5, 1, 8, 3, 9, 2} testet, d.h. das Array vor dem Sortieren ausgibt, sortiert und dann nochmal ausgibt. Um eine bessere Modularität zu erhalten, schreibe eine eigene Methode für das Bestimmen des grössten Elements und für das Vertauschen von zwei Elementen.

Lösung:

```
public class SelectionSort {

  private void swap(int[] array, int index1, int index2) {
    int tmp = array[index1];
    array[index1] = array[index2];
    array[index2] = tmp;
  }

  private int findIndexMax(int[] array, int topIndex) {
    int index = 0;
    for (int i = 1; i <= topIndex; i++) {
      if (array[i] > array[index]) {
        index = i;
      }
    }
    return index;
  }

  public void selectionSort(int[] array) {
    int p = array.length - 1;
    while (p >= 0) {
      int g = findIndexMax(array, p);
      swap(array, p, g);
      p--;
    }
  }

  public static void main(String[] args) {
    int[] f = new int[]{5, 1, 8, 3, 9, 2};
    System.out.print("Vor dem Sortieren: ");
    for (int i = 0; i < f.length; i++) {
      System.out.print(f[i]+" ");
    }
    System.out.println("");

    SelectionSort sort = new SelectionSort();
    sort.selectionSort(f);

    System.out.print("Nach dem Sortieren: ");
    for (int i = 0; i < f.length; i++) {
      System.out.print(f[i]+" ");
    }
    System.out.println("");
  }

}
```

Aufgabe 20:

Implementiere den InsertionSort für ein Array von „int“ Werten analog zu den beiden vorherigen Sortierverfahren.

Lösung:

```
public class InsertionSort {

  public void insertionSort(int[] array) {
    for (int i = 1; i <= (array.length-1); i++) {
      if (array[i] < array[i-1]) {
        int m =  array[i];
        int j = i;
        while (j > 0) {
          if (array[j-1] >= m) {
            array[j] = array[j-1];
            j--;
          } else {
            break;
          }
          array[j] = m;
        }
      }
    }
  }

  public static void main(String[] args) {
    int[] f = new int[]{5, 1, 8, 3, 9, 2};
    System.out.print("Vor dem Sortieren: ");
    for (int i = 0; i < f.length; i++) {
      System.out.print(f[i]+" ");
    }
    System.out.println("");

    InsertionSort sort = new InsertionSort();
    sort.insertionSort(f);

    System.out.print("Nach dem Sortieren: ");
    for (int i = 0; i < f.length; i++) {
      System.out.print(f[i]+" ");
    }
    System.out.println("");
  }

}
```

Aufgabe 21:

Implementiere den QuickSort für ein Array von „int“ Werten analog zu den vorherigen Sortierverfahren.

Lösung:

```
public class QuickSort {

  private void swap(int[] array, int index1, int index2) {
    int tmp = array[index1];
```

```
    array[index1] = array[index2];
    array[index2] = tmp;
  }

  private int zerlege(int[] array, int u, int o, int p) {
    int pn = u;
    int pv = array[p];
    swap(array, p, o);
    for (int i = u; i <=  (o-1); i++) {
      if (array[i] <= pv) {
        swap(array, i, pn);
        pn++;
      }
    }
    swap(array, o, pn);
    return pn;
  }

  public void quickSort(int[] array, int u, int o) {
    if  (o > u) {
      int p = (u+o)/2;
      int pn = zerlege (array, u, o, p);
      quickSort(array, u, pn-1);
      quickSort(array, pn+1, o);
    }
  }

  public static void main(String[] args) {
    int[] f = new int[]{5, 1, 8, 3, 9, 2};
    System.out.print("Vor dem Sortieren: ");
    for (int i = 0; i < f.length; i++) {
      System.out.print(f[i]+" ");
    }
    System.out.println("");

    QuickSort sort = new QuickSort();
    sort.quickSort(f, 0, f.length-1);

    System.out.print("Nach dem Sortieren: ");
    for (int i = 0; i < f.length; i++) {
      System.out.print(f[i]+" ");
    }
    System.out.println("");
  }

}
```

Aufgabe 22:

Implementiere den MergeSort für ein Array von „int“ Werten analog zu den vorherigen Sortierverfahren.

Lösung:

```
public class MergeSort {

  public void mergeSort(int[] array, int u, int o) {
    if  (u < o) {
      int mitte = (u + o + 1) / 2;
      mergeSort(array, u, mitte-1);
```

```
      mergeSort(array, mitte, o);
      merge(array, u, mitte, o);
    }
  }

  private void merge(int[] array, int u, int mitte, int o) {
    int[] temp = new int[o-u+1];
    int j = u;
    int k = mitte;
    for (int i=0; i < temp.length; i++) {
      if ((k > o) || (j < mitte && array[j] < array[k])) {
        temp[i] = array[j];
        j++;
      } else {
        temp[i] = array[k];
        k++;
      }
    }
    for (int n = 0; n < temp.length; n++) {
      array[u + n] = temp[n];
    }
  }

  public static void main(String[] args) {
    int[] f = new int[]{5, 1, 8, 3, 9, 2};
    System.out.print("Vor dem Sortieren: ");
    for (int i = 0; i < f.length; i++) {
      System.out.print(f[i]+" ");
    }
    System.out.println("");

    MergeSort sort = new MergeSort();
    sort.mergeSort(f, 0, f.length-1);

    System.out.print("Nach dem Sortieren: ");
    for (int i = 0; i < f.length; i++) {
      System.out.print(f[i]+" ");
    }
    System.out.println("");
  }

}
```

Aufgabe 23:

Implementiere den HeapSort für ein Array von „int“ Werten analog zu den vorherigen Sortierverfahren.

Lösung:

```
public class HeapSort {

  public void heapSort(int[] array) {
    // Heap Eigenschaft herstellen
    for (int i = array.length / 2; i >= 0; i--) {
      versickere(array, i, array.length);
    }

    // Sortieren
    int m = array.length - 1;
```

```
    while (m > 0) {
      swap(array, 0, m);
      versickere(array, 0, m);
      m--;
    }
  }

  private void versickere(int[] array, int index, int last) {
    int i = index + 1; // fuer Adressberechnung Indexumsetzung notwendig
    while (2*i <= last) { // array[i] hat linkes Child
      int j = 2*i; // linkes Child
      if (j < last) { // array[i] hat auch rechtes Child
        // Kleineres der Childs ermitteln
        if (array[j-1] > array[j]) {
          j++;
        }
      }
      // Child groesser als Eltern?
      if (array[i-1] > array[j-1]) {
        swap(array, i-1, j-1);
        i = j; // versickere nun diesen Knoten
      } else {
        break; // heap erfuellt
      }
    }
  }

  private void swap(int[] array, int index1, int index2) {
    int tmp = array[index1];
    array[index1] = array[index2];
    array[index2] = tmp;
  }

  public static void main(String[] args) {
    int[] f = new int[]{5, 1, 8, 3, 9, 2};
    System.out.print("Vor dem Sortieren: ");
    for (int i = 0; i < f.length; i++) {
      System.out.print(f[i]+" ");
    }
    System.out.println("");

    HeapSort sort = new HeapSort();
    sort.heapSort(f);

    System.out.print("Nach dem Sortieren: ");
    for (int i = 0; i < f.length; i++) {
      System.out.print(f[i]+" ");
    }
    System.out.println("");
  }

}
```