1.

Was versteht man unter einer rekursiven Methode/Funktion? (4 Punkte)

nur direkt!

Eine Methode / Funktion, die sich selbst aufruft.

war nicht gefragt

Dabei muss beachtet werden, dass sie sich nie mit den gleichen Parametern aufruft und eine Abbruchbedingung hat.

2/4

2. (20 Punkte)

Die Fakultätsfunktion ist durch

$$n! \stackrel{\text{def}}{=} \begin{cases} 1 & \text{, falls } n \leq 1 \\ n \cdot (n-1)! & \text{sonst} \end{cases}$$

gegeben.

(a) (10 Punkte)

Implementieren Sie eine rekursive Java-Methode, die die Fakultätsfunktion für positive `int`-Werte berechnet.

```
public static int rekFakultaet (int x){
    if (x <= 1) return 1;
    return x*rekFakultaet ((x-1));
}
```

10/10

(b) (10 Punkte)

Implementieren Sie eine nichtrekursive Java-Methode, die die Fakultätsfunktion für positive `int`-Werte berechnet.

```
public static int fakultaet(int x){
    int ergebnis = 1;
    while (x >= 1){
        ergebnis = ergebnis * x;
        x = x - 1;
    }
    return ergebnis;
}
```

10/10

3. (12 Punkte)

Geben Sie eine Funktion in Pseudocode an die das folgende Problem löst:
Eingabe: Ein sortiertes `int`-Array `a` sowie ein `int`-Wert `x`
Ausgabe: `true`, falls `x` in `a` enthalten ist und `false` sonst.
Achten Sie bei Ihren Algorithmus auf Zeiteffizienz! Wie schnell ist Ihr Algorithmus in Abhängigkeit der Arraylänge n (*O*-Notation)?

```
function contains (int-Array a, int x){
    for (int i=0; i<a.length; i++){
        if (a[i]== x) return true
    }
    return false;
}
```

⇒ sehr langsam $O(n)$

Binäre Suche, rekursiv mit 2 Fkt.

```
function contains(a, x){
 return binSearch(0, Länge von a-1, x)
}
function binSearch (li, re, a, x){
  int mitte = (li+re)/2
  if (a[li] == x){
        return true;
  } else {return false;}
  binSearch (li, mitte, a, x);
  binSearch (mitte+1, re, a, x);
}
```

⇒ $O(\log n)$ 12/12

4. (10 Punkte)

Es sei die folgende Methode gegeben, die (abgesehen von Rundungsfehlern) die Varianz der in einem `int`-Array gespeicherten Werte errechnet, wenn man davon ausgeht, dass das Auftreten der Werte gleichverteilt ist:

```
public static int varianz(int[] a) {
    int x = 0;                    // 1 Takt
    int n = a.length;             // 2 Takte
    for (int i = 0; i < n; i++)   x = x + a[i]; // 3 Takte
    int ew = x / n;               // 2 Takte
    x = 0;                        // 1 Takt
    for (int i = 0; i < n; i++) {
        int y = (ew - a[i]);      // 3 Takte
        x = x + (y * y);          // 3 Takte
    }
    return ew / n;                // 1 Takt
}
```

Ⓐ 1 Takt
Ⓑ 1 Takt } je Durchlauf
Ⓒ 2 Takte }

Ⓐ 1 Takt Ⓑ 1 Takt je Durchlauf Ⓒ 2 Takte je Durchlauf

Bestimmen Sie die genaue Laufzeit (in Takten) der Methode in Abhängigkeit der Länge n des Arrays `a`. Geben Sie dazu für jede Zeile des Quellcodes die jeweilige Taktzahl an. Notieren Sie zusätzlich die Laufzeit eventuell vorkommender Schleifen. Vergessen Sie nicht die Gesamtlaufzeit der Methode anzugeben.

$3 + 1 + (n+1)\cdot 1 + n\cdot 2 + 3\cdot n + 3 + 1 + (n+1)\cdot 1 + n\cdot 2 + n\cdot 6 + 1$

$= 9 + 15n + 2(n+1) = \underline{15n + 11 \text{ Takte}}$

10/10

5.

Die folgenden beiden Methoden implementieren einen Sortieralgorithmus. (14 Punkte)

```
public static void sort(int[] a) {
    sortRek(a, 1);
}


public static void sortRek(int[] a, int li) {
    int n = a.length;
    if (li < n) {
        int j = li - 1;
        while ((j >= 0) && (a[j] > a[j + 1])) {
            int z = a[j + 1];
            a[j + 1] = a[j];
            a[j] = z;
            j--;
        }
        sortRek(a, li + 1);
    }
}
```

(a) (2 Punkte)

Um welchen Sortieralgorithmus handelt es sich?

Bubble ~~Insertion~~ Sort

0/2

(b) (2 Punkte)

Welche Laufzeit hat der Algorithmus in Abhängigkeit der Arraylänge n?

Im Worst-Case eine Laufzeit von $O(n^2)$.

2/2

(c) (10 Punkte)

Implementieren Sie eine nichtrekursive Variante des Algorithmus (Java oder Pseudocode).

function bubbleSort(a){ Eingabe: int-Array a mit Länge n

int n = n-1

solange n>0 tue das folgende{

für alle i zwischen 0 und n-1 tue das folgende(

ist a[i] > [a[i+1], so vertausche beide

}

n=n-1

}

3 3

10/10

6.

Gegeben sei die folgende Liste mit Aussagen. Bitte kreuzen Sie an, welche der Aussagen wahr sind, und welche falsch. (13 Punkte)

Beachten Sie: Für jede richtig angekreuzte Antwort erhalten Sie einen Punkt, für jede falsch angekreuzte Antwort bekommen Sie einen Punkt abgezogen. Insgesamt können Sie auf diese Aufgabe aber nicht weniger als 0 Punkte bekommen.

wahr	falsch	Aussage
☒	☐	$O(\sqrt{n}) \neq O(\log n)$
☐	☒	Jeder vergleichsbasierte Sortieralgorithmus benötigt höchstens Zeit $O(n \cdot \log n)$.
☐	☒	Jeder Graph ist auch ein Baum.
☒	☐	Für *jede* rekursive Methode gibt es eine nichtrekursive, die das selbe berechnet.
☐	☒	$O(\log_2 n) \neq O(\log_3 n)$
☐	☒	Eine Hashfunktion $h : G \to K$ ist immer dann gegeben, wenn $\|G\| < \|K\|$.
☐	☐	Für jede nichtrekursive Methode gibt es eine rekursive, die das selbe berechnet.
☐	☒	Wenn eine Kollision beim Einfügen in eine Hashtabelle auftritt, wird normalerweise ein Re-Hashing durchgeführt.
☐	☒	Es ist einer Adjazenzmatrix direkt anzusehen, ob sie einen gerichteten oder einen ungerichteten Graphen codiert.
☐	☒	$O(3 \cdot n^2) = O(\frac{1}{3} \cdot n^3)$
☒	☐	Es ist möglich die Struktur eines Binärbaums (ohne Knoteninhalte) in einer Adjazenzmatrix zu codieren.
☐	☒	$O(2^n) = O(2^{2n})$
☐	☒	$O(1,001^n) \neq O(2^n)$

10/13

$u^n = \ldots \left(2^{\log k}\right)^n = 2^{\log k \cdot n} \in \ldots 2^{O(n)} \ldots \Rightarrow O(2^n)$

Lipschitz

ch	Aussage
	$O(\sqrt{n}) \neq O(\log n)$
	Jeder vergleichsbasierte Sortieralgorithmus benötigt höchstens Zeit $O(n \cdot \log n)$.
	Jeder Graph ist auch ein Baum.
	Für jede rekursive Methode gibt es eine nichtrekursive, die das selbe berechnet.
	$O(\log_3 n) \neq O(\log_7 n)$
	Eine Hashfunktion $h : G \to K$ ist immer dann gegeben, wenn $\|G\| < \|K\|$.
	Für jede nichtrekursive Methode gibt es eine rekursive, die das selbe berechnet.
	Wenn eine Kollision beim Einfügen in eine Hashtabelle auftritt, wird normalerweise ein Re-Hashing durchgeführt.
	Es ist einer Adjazenzmatrix direkt anzusehen, ob sie einen gerichteten oder einen ungerichteten Graphen codiert.
	$O(3 \cdot n^3) = O(\frac{1}{2} \cdot n^3)$
	Es ist möglich die Struktur eines Binärbaums (ohne Knoteninhalte) in einer Adjazenzmatrix zu codieren.
	$O(2^n) = O(2^{n-1})$.
	$O(1,0001^n) \neq O(2^n)$

7.

Geben Sie eine nichtrekursive(!) Methode mit dem Kopf (20 Punkte)

```
void quicksort(int[] a)
```

an, die das gegebene Array `a` mit Hilfe des Quicksort-Algorithmus sortiert. Als Hilfsmittel dürfen Sie die Klasse `LinkedList` benutzen (sonst keine weiteren externen Klassen). Sie dürfen weitere Methoden implementieren und benutzen aber diese dürfen alle auch nicht rekursiv sein.

Falls Sie die Methoden der Klasse `LinkedList` nicht mehr so im Kopf haben, hier einige dieser Methoden aus der API der Klasse:

`LinkedList()` (Konstruktor) Constructs an empty list.	`boolean add(Object e)` Appends the specified element to the end of this list.
`void addFirst(Object e)` Inserts the specified element at the beginning of this list.	`int size()` Returns the number of elements in this list.
`Object get(int index)` Returns the element at the specified position in this list.	`Object pop()` Pops an element from the stack represented by this list.
`void push(Object e)` Pushes an element onto the stack represented by this list.	`Object set(int index, Object element)` Replaces the element at the specified position in this list with the specified element.

```
void quicksort(int[] a){
    int links = 0;
    int rechts = a.length-1;
    int pivot = a[0];
    while (links < rechts){
        if (a[links+1] <= pivot){
            int z = a[links+1];
            a[links+1] = a[links];
            a[links] = z;
            links++;
        } else {
            int z = a[links+1];
            a[links+1] = a[rechts];
            a[rechts] = z;
            rechts--;
        }
    } // 1. Pivot-Element an der richtigen
      Stelle. Eine Liste
      mit allen kleineren
      Elementen, eine
      mit allen
      größeren.
```

LinkedList kl
LinkedList gr

kl.add(z);
gr.add(z);

Aber was jetzt?

3/20

7.

Geben Sie eine nichtrekursive(!) Methode mit dem Kopf (20 Punkte)

`void quicksort(int[] a)`

an, die das gegebene Array `a` mit Hilfe des Quicksort-Algorithmus sortiert. Als Hilfsmittel dürfen Sie die Klasse `LinkedList` benutzen (sonst keine weiteren externen Klassen). Sie dürfen weitere Methoden implementieren und benutzen aber diese dürfen alle auch nicht rekursiv sein.

Falls Sie die Methoden der Klasse `LinkedList` nicht mehr so im Kopf haben, hier einige dieser Methoden aus der API der Klasse:

`LinkedList()` (Konstruktor) Constructs an empty list.	`boolean add(Object e)` Appends the specified element to the end of this list.
`void addFirst(Object e)` Inserts the specified element at the beginning of this list.	`int size()` Returns the number of elements in this list.
`Object get(int index)` Returns the element at the specified position in this list.	`Object pop()` Pops an element from the stack represented by this list.
`void push(Object e)` Pushes an element onto the stack represented by this list.	`Object set(int index, Object element)` Replaces the element at the specified position in this list with the specified element.

```
void quicksort (int[] a) {
    list = neue LinkedList
    list.add (0)
    list.add (a.length)
    while (list.size()>0) {
      int li = list.pop()
      int re = list.pop()
      int rep = re
      int lip = li
      int pivot = a[li]
      while li < re {
        if a[li+1] <= pivot {
            vertausche a[li] und a[li+1]
            li++
        }else{
            vertausche a[li+1] und a[re]
            re--
        }
      }
      if (li > lip+1) {
          list.add(lip)
          list.add(li-1) }
      if (li < rep-1) {
          list.add(li+1)
          list.add(rep) }
}
```

20
~~18~~/20