a)
```
public static void rev1(int n)
{
    System.out.print(n % 10);
    if (n > 9)
        rev1(n / 10);
}
```

b)
```
public static int rev2(int n)
{
    if (n <= 9)
        return n;

    int logn = (int)Math.log10(n);
    int zehnHochLogn = (int)Math.pow(10, logn);

    return (n % 10) * zehnHochLogn + rev2(n / 10);
}
```

c) Programmieren Sie eine iterative Methode, die dieselbe Ausgabe wie `rev1` bzw. dieselbe Rückgabe wie `rev2` erzeugt.

d) Programmieren Sie die folgenden Methoden der Klasse LinkedList REKURSIVE

```
public int size() {
    if (isEmpty()) {
        return 0;
    } else {
        int counter = 0;
        Node ptr = head;
        while (ptr != null) {
            ptr = ptr.next;
            counter++;
        }
        return counter;
    }
}
```

```
public void add(int newValue) {
    Node newNode = new Node(newValue, null);
    if (isEmpty()) {
        head = last = newNode;
    } else {
        Node ptr = head;
        while (ptr.next != null) {
            ptr = ptr.next;
        }
        ptr.next = newNode;
        last = newNode;
    }
}
```

```java
public void add(int index, int element) {
    Node newNode = new Node(element, null);
    if (index < 0) {
        System.out.println("Invalid Index!");
    } else if (index == 0) {
        if (isEmpty()) {
            add(element);
        } else {
            newNode.next = head;
            head = newNode;
        }
    } else if (index >= size() - 1) {
        add(element);
    } else {
        int i = 0;
        Node ptr = head;
        while (i != index - 1) {
            ptr = ptr.next;
            i++;
        }
        newNode.next = ptr.next;
        ptr.next = newNode;
    }
}
```

```java
private String toStringHelper() {
    String erg = "[";
    Node ptr = head;
    while (ptr.next != null) {
        erg += ptr.value + ", ";
        ptr = ptr.next;
    }
    erg += ptr.value + "]";
    return erg;
}
```

```java
public boolean contains(int value) {
    Node ptr = head;
    while (ptr != null) {
        if (ptr.value == value) {
            return true;
        }
        ptr = ptr.next;
    }
    return false;
}
```

Bestimmen Sie für die nachstehenden Methoden folgende Informationen:

- Berechnen Sie die Anzahl der Aufrufe von `tuwas()` für $n=3$
- Bestimmen Sie die Anzahl der Aufrufe von `tuwas()` als Funktion von n
- Bestimmen Sie die asymptotische Zeitkomplexität (O-Notation) unter der Annahme, dass die asymptotische Zeitkomplexität von `tuwas()` $O(1)$ ist

```
public int funk1(int n)
{
    tuwas();

    if (n <= 1)
    {
        return n;
    }
    else
    {
        return n + funk1(n - 1);
    }
}
```

```
public void proz2(int n)
{
    tuwas();

    if (n > 0)
    {
        proz2(n - 1);
        proz2(n - 1);
    }
}
```

Die Ulam-Funktion für natürliche Zahlen ist wie folgt definiert:

$$\text{Ulam}(n) = \begin{cases} 1 & \text{für } n = 1 \\ \text{Ulam}\left(\frac{n}{2}\right) & \text{für } n \text{ gerade } \wedge\ n > 1 \\ \text{Ulam}(3 * n + 1) & \text{für } n \text{ ungerade } \wedge\ n > 1 \end{cases}$$

a) Berechnen Sie folgende Funktionswerte auf dem Papier, und geben Sie alle Zwischenschritte an, die für die Berechnung jeweils notwendig sind.

$\text{Ulam}(2) =$

$\text{Ulam}(3) =$

Beispiel: $\text{Ulam}(18) = \text{Ulam}(9) = \ldots$

b) Schreiben Sie eine rekursive und eine iterative Funktion in Java, die die Werte der Ulam-Funktion berechnet.

Die Folge der Fibonacci-Zahlen ist rekursiv definiert durch:

1. $\text{fib}(0) = 0, \text{fib}(1) = 1$
2. $\text{fib}(n) = \text{fib}(n\text{-}1) + \text{fib}(n\text{-}2)$ für $n \geq 2$

Die Fibonacci-Zahlen ergeben somit die Folge $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$

a) Schreiben Sie je eine Java-Funktion, die die n-te Fibonacci-Zahl rekursiv bzw. iterativ berechnet, ohne den Programmcode aus Aufgabe 5 zu verwenden. Wie ist das Laufzeitverhalten beider Varianten?

b) Schreiben Sie außerdem ein Hauptprogramm, das mit Hilfe der beiden Funktionen jeweils die ersten 50 Fibonacci-Zahlen berechnet und die Rechenzeit für beide Vorgehensweisen vergleicht.