.

Aufgabe 7.1

Gegeben ist folgende rekursiv definierte Funktion $g : \mathbb{N} \to \mathbb{N}$ mit

$$g(n) = \begin{cases} 0 & \text{falls } n = 0, \\ 1 & \text{falls } n = 1, \\ 2 \cdot g(n/2) & \text{falls } n > 1 \text{ und } n \text{ gerade}, \\ g(n+1) + 1 & \text{falls } n > 1 \text{ und } n \text{ ungerade} \end{cases}$$

(a) Füllen Sie die folgende Tabelle aus:

n	0	1	2	3	4	5	6	7	8	9	10
$g(n)$											

(2 Punkte)

(b) Schreiben Sie eine Klasse `MyRecursiveFunction.java`, die eine rekursive Methode

```
public static int g(int n)
```

enthält, die die oben definierte Funktion $g(n)$ berechnet. Außerdem soll die Klasse eine `main`-Methode enthalten, von der aus die Methode `g` aufgerufen wird. (3 Punkte)

Aufgabe 7.2

Im Material zur Übung ist noch einmal die aus der Vorlesung bekannte Implementierung einer verketteten Liste in der Klasse `MyLinkedList.java` enthalten. Schreiben Sie eine Klasse `MyListMerger.java`, welche eine Methode

```
public static MyLinkedList mergeLists(MyLinkedList l1, MyLinkedList l2)
```

enthält. Diese Methode bekommt zwei verkettete Listen übergeben, die jeweils eine aufsteigend sortierte Folge von ganzen Zahlen enthalten. Diese beiden Folgen sollen zu einer aufsteigend sortierten Folge gemischt werden, die auch wieder in einer verketteten Liste gespeichert werden soll. Diese Liste soll der Rückgabewert der Methode sein. (5 Punkte) Um welche potentiellen Probleme, die zu Fehlermeldungen während der Laufzeit führen, müsste man sich eigentlich auch noch Gedanken machen? (1+Punkt)

Aufgabe 7.3

Bringen Sie die folgenden sieben Funktionen in eine Reihenfolge $f_1, f_2, \ldots, f_7$, sodass $f_i \in O(f_{i+1})$ für alle $i \in \{1, 2, \ldots, 6\}$ gilt.

$$n^2, \quad \sqrt[5]{n}, \quad \log_3 n, \quad (\sqrt{2})^n, \quad \sqrt[7]{n}, \quad \log_2 n, \quad n^{\sqrt{2}}$$

(2 Punkte)

Rückseite beachten!

Aufgabe 7.4

Gegeben sind zwei aufsteigend sortierte Arrays `A` und `B`. Alle Elemente sind paarweise verschiedene ganze Zahlen. Die Anzahl der Elemente in `A` sei n und die Anzahl der Elemente in `B` sei m. Die Arrays `A` und `B` sollen gemischt werden. Prof. Sain-Tist möchte dafür unbedingt die binäre Suche verwenden. Deshalb schlägt er vor, wie folgt vorzugehen:

(1) Für jedes Element `A[i]` wird mittels binärer Suche die Anzahl $a(i)$ der Elemente in `B` ermittelt, die kleiner als `A[i]` sind.

(2) Analog wird dann für jedes Element `B[j]` mittels binärer Suche die Anzahl $b(j)$ der Elemente in `A` ermittel werden, die kleiner als `B[j]` sind.

(3) Ein neues Array `C` der Länge $n + m$ wird angelegt.

(4) Das Element `A[i]` wird in `C` an die Stelle $a(i) + i$ geschrieben.

(5) Das Element `B[j]` wird in `C` an die Stelle $b(j) + j$ geschrieben.

Funktioniert das so überhaupt? Welche Laufzeit hat dieser Algorithmus in Abhängigkeit von n und m (O-Notation)? (4 Punkte)