Aufgabensammlung zum Buch „Algorithmen und Datenstrukturen“ (Kapitel 1)

1 Aufgaben aus dem Buch

Zu folgenden Aufgaben, die direkt aus dem Buch entnommen sind, gibt es an der Universität Freiburg am Lehrstuhl Ottmann Musterlösungen. In der Version mit Lösungen sind diese angegeben. Hinter der fortlaufenden Aufgabennummer steht in Klammern die Nummer der Aufgabe im Buch.

Aufgabe 1 (Aufgabe 1.5):

(Bundeswettbewerb Informatik 1985)

Ein großes Wirtschaftsmagazin will seinen Lesern eine Analyse der Börsenentwicklung der letzten fünf Jahre präsentieren. Dazu sollen unter anderem die Kurse der wichtigsten Aktien in diesem Zeitraum untersucht werden. Für jede Aktie soll nachträglich ein bester Einkaufstag festgestellt werden.

Dabei wird angenommen, dass ein Kapitalanleger jede Aktie höchstens einmal eingekauft hätte, und zwar in einer beliebigen Stückzahl, und dass er zum Ende des betrachteten Zeitraums alle Stücke wieder verkauft hat. Der beste Einkaufstag für eine Aktie wäre dann derjenige gewesen, der zu einem eingesetzten Betrag den höchsten Gewinn geliefert hätte (Steuern, Gebühren und alternative Anlagemöglichkeiten sollen außer Betracht bleiben).

Das Wirtschaftsmagazin hat von einem Börsendienst Informationen über die Notierungen jeder Aktie für alle Börsentage der letzten fünf Jahre gekauft. Für jede Aktie erhält es eine Zahlenfolge. Die erste Zahl ist der Kurs der Aktie am ersten Börsentag und jede folgende Zahl gibt die absolute Kursveränderung gegenüber dem Vortag an, in der Reihenfolge der Börsentage. Der Kurs, der sich für einen gewissen Tag ergibt, gilt für alle Käufe und Verkäufe dieses Tages.

Unterstützen Sie die Kursanalyse durch Schreiben eines Programms, das für eine Aktie aus der gegebenen Zahlenfolge nachträglich einen besten Einkaufstag, einen besten Verkaufstag und den dabei höchsten erzielbaren Gewinn (in Prozent vom eingesetzten Betrag) ermittelt. Da das Programm sehr lange Zahlenfolgen bearbeiten muss, ist es außerordentlich wichtig, dass die Laufzeit bei zunehmender Zahlenfolgenlänge nicht stärker als nötig wächst.

Beispiel: Die Eingabe

„127.5 -0.5 2 -1 1 3.5 -13 7 -2 -6 -9 -21 -17 -5 0.5 4 -7 -12 2.5 -3 2“

liefert die Ausgabe:

„Ein bester Einkaufstag wäre der 14. Börsentag gewesen, ein dazugehöriger Verkaufstag der 16. Börsentag. Der so realisierbare Gewinn wäre 6.7669 % vom eingesetzten Betrag gewesen.“

Aufgabe 2 (Aufgabe 1.6):

Gegeben sei ein lineares Feld `a[1..n]` positiver reeller Zahlen. Gegeben seien außerdem eine Funktion g die als Werte 0 oder 1 liefert, und die folgende Funktion *gtest*:

```
int gtest(int li, int re) {
  if (li > re) return 0;
  int m = (li+re)/2;
  return gtest(li,m-1)+g(a[m])+gtest(m+1,re);
}
```

a) Beschreiben Sie, welches Resultat die Funktion beim Aufruf $gtest(1, n)$ für ein gegebenes Feld a liefert.

b) Ermitteln Sie größenordnungsmäßig die Anzahl der Additionen bei der Ausführung eines Aufrufs von $gtest(li, re)$ im schlimmsten Fall, in Abhängigkeit von $|\, re - li \,|$, mit Hilfe einer Rekursionsformel.

c) Geben Sie in Java ein alternatives (iteratives) Verfahren zur Ermittlung des Funktionswertes $gtest$ an; verwenden Sie denselben Funktionskopf.

Aufgabe 3 (Aufgabe 1.14):

Erstellen Sie zwei Pseudo-Code Programme zur iterativen Berechnung der folgenden verallgemeinerten Binomialkoeffizienten.

$$\begin{array}{ccccccccc} & & & & 1 & & & & \\ & & & 1 & & 2 & & & \\ & & 1 & & 3 & & 4 & & \\ & 1 & & 4 & & 7 & & 8 & \\ 1 & & 5 & & 11 & & 15 & & 16 \end{array}$$

Das allgemeine Bildungsgesetz lautet $\binom{d}{0} = 1,\ \binom{d}{d} = 2^d$ und $\binom{d}{h} = \binom{d-1}{h} + \binom{d-1}{h-1}$.

a) Das erste Programm verwende den internen Stapel durch rekursive Berechnung.

b) Das zweite Programm verwende ein Berechnungsschema, das mehrfache Berechnung von gleichen Teilresultaten vermeidet.

Aufgabe 4 (Aufgabe 1.17):

Geben Sie ein Verfahren an, das mit Hilfe eines Stapels eine Folge von Buchstaben einliest und in umgekehrter Reihenfolge wieder ausgibt. Die Länge der Folge ist unbekannt, das Ende der Eingabe durch einen Punkt markiert. Das Verfahren soll nur auf die für Stapel üblichen Operationen, aber nicht auf eine konkrete Implementation Bezug nehmen.

2 Ähnlich Aufgaben

Bei den folgenden Aufgaben handelt es sich um Aufgaben die an der ETH Zürich, am Institut für Theoretische Informatik und an der Universität Freiburg im Institut für Informatik in diversen Vorlesungen gestellt wurden. Inhaltlich sind diese Aufgaben mit dem behandelten Stoff im Buch verwandt. Zu allen Aufgaben gibt es Musterlösungen, die allerdings nur in der Version mit Lösungen enthalten sind.

Aufgabe 5:

Die Fibonacci-Zahlen sind für $n \in \{0, 1, 2 \ldots\}$ wie folgt rekursiv definiert:

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2) \qquad \text{für } n \geq 2. \end{aligned}$$

Zeigen Sie z.B. durch vollständige Induktion:

1. Für $n \geq 6$ ist $F(n) \geq 2^{n/2}$.

2. Für $n \geq 0$ gilt $F(0) + \cdots + F(n) = F(n+2) - 1$.
3. Für $n \geq 1$ ist $F^2(n) = F(n-1)F(n+1) + (-1)^{n+1}$.

Aufgabe 6:

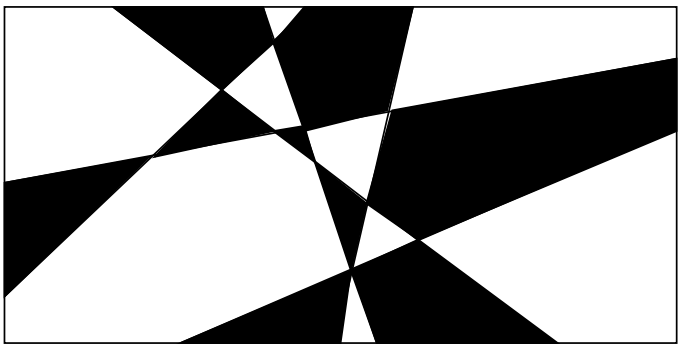
1. Sei F(n) die n-te Fibonacci-Zahl. Zeigen Sie: $F(n) \geq (\sqrt{2})^n, \forall n \in \mathbb{N}$ mit $n \geq 6$
2. Gegeben sei die folgende Java-Methode:

```
int fib (int n)
{
  if (n==0 || n==1) return n;
  else return fib(n-1)+fib(n-2);
}
```

   Geben Sie für $fib(5)$ alle Methodenaufrufe mit den zugehörigen Parameterwerten in der Aufrufreihenfolge an.
3. Schätzen Sie die Zahl der rekursiven Aufrufe von fib in Abhängigkeit von n ab.

Aufgabe 7:

Zeigen Sie, daß, wie im Beispiel der Abbildung, jede Aufteilung der Ebene durch n verschiedene Linien mit *schwarz* und *weiß* eingefärbt werden kann, so daß keine zwei Regionen gleicher Farbe eine Kante teilen. Hinweis: Auch hier läßt sich das Prinzip der vollständigen Induktion im Beweis nutzen.



Aufgabe 8:

Beweisen Sie durch vollständige Induktion:

(a) $2n < 2^n$ für $n \geq 3$.

(b) $\sum_{i=1}^{n} i^3 = \left(\sum_{i=1}^{n} i \right)^2$.

Aufgabe 9:

Professor Hurtig hat folgendermaßen bewiesen, dass alle Menschen die gleiche Haarfarbe haben:

Aussage: Gegeben eine beliebige Menge von n Personen, $n \geq 1$. Dann haben alle diese Personen die gleiche Haarfarbe.

Beweis durch Induktion:
Induktionsanfang: $n = 1$. Eine einzelne Person hat offensichtlich die gleiche Haarfarbe wie sie selbst.
Induktionsschluss: Sei n fest und sei die Aussage für alle k, $1 \leq k \leq n$, schon bewiesen. Sei M eine Menge von $n+1$ Personen und $A \in M$ eine beliebige Person aus dieser Menge. Betrachtet man die Menge $M \setminus \{A\}$, so haben nach Induktionsvoraussetzung alle Personen in dieser Menge die gleiche Haarfarbe. Wählt man eine andere Person $B \in M$ aus, so haben ebenfalls nach Induktionsvoraussetzung alle Personen in der Menge $M \setminus \{B\}$ die gleiche Haarfarbe. Somit haben A und B die gleiche Haarfarbe, und damit alle Personen aus M.

Was hat Professor Hurtig hier falsch gemacht?

Aufgabe 10:

Definieren Sie die Begriffe *terminierend*, *determiniert* und *deterministisch*.

Aufgabe 11:

1. Zeigen Sie durch die Wahl geeigneter Konstanten c und n_0, daß $(n+1)^2$ in der Klasse $O(n^2)$ liegt.
2. Es gilt die folgende Beziehung: Falls der Grenzwert von $f(n)/g(n)$ für n gegen unendlich durch eine Konstante c beschränkt ist (wir schreiben $\lim_{n\to\infty} f(n)/g(n) \leq c$), so gilt $f(n) \in O(g(n))$. Zeigen Sie unter Verwendung dieses Sachverhaltes abermals $(n+1)^2 \in O(n^2)$.
3. Die Regel von L'Hospital besagt: Falls der Grenzwert $f'(n)/g'(n)$ für n gegen unendlich existiert (f' ist die Ableitung von f), so ist er gleich dem Grenzwert von $f(n)/g(n)$. Zeigen Sie mit Hilfe dieser Regel letztmalig $(n+1)^2 \in O(n^2)$.
4. Sei $P(n)$ ein Polynom vom Grad k. Zeigen Sie: $O(\log P(n)) = O(\log n)$

Aufgabe 12:

1. Zeigen Sie: $3n^4 - n^2 = O(n^4)$.
2. Zeigen Sie: $2^n = o(n!)$ (Hinweis $n! \geq (n/2)^{n/2}$).
3. Zeigen Sie: Für alle k ist $n^k = o(2^n)$.
4. Zeigen Sie: Es seien p_1 und p_2 Polynome vom Grad d_1 bzw. d_2, wobei die Koeffizienten von n^{d_1} und n^{d_2} positiv sind. Dann gilt:
    1. $d_1 = d_2 \Rightarrow p_1 = \Theta(p_2)$,
    2. $d_1 < d_2 \Rightarrow p_1 = o(p_2)$,
    3. $d_1 > d_2 \Rightarrow p_1 = \omega(p_2)$.
5. Zeigen Sie: Für alle $k > 0$ und $\epsilon > 0$ gilt $\log^k n = o(n^\epsilon)$

6. Zeigen Sie: $2^{n/2} = o(2^n)$.
7. Finden Sie eine Funktion, in denen beide Definitionen von Ω aus der Vorlesung voneinander abweichen.
8. Für monoton steigende Funktionen $f \not\equiv 0$ sind beide Definitionen für *Groß-O* aus der Vorlesung äquivalent.

Aufgabe 13:

Beweisen Sie die Additionsregel $O(f)+O(g) = O(\max\{f,g\})$ und die Multiplikationsregel $O(f)\cdot O(g) = O(f \cdot g)$. Worin besteht der praktische Nutzen dieser Regeln?

Aufgabe 14:

Wir betrachten Funktionen von den natürlichen Zahlen in die nicht-negativen Zahlen.

1. Welche Funktionen liegen in $\Theta(1)$?
2. Wann gilt $f(n) \in O(\lfloor f(n) \rfloor)$?
3. Seien a, b, c reell mit $1 < b, 1 \leq a, c$. Zeigen Sie:
$$\log_b(an + c) \in \Theta(\log_2 n)$$
4. Seien $f_1, f_2, \ldots$ Funktionen in $O(g)$. Gilt dann
$$h(n) = \sum_{i=1}^{n} f_i(n) \in O(ng(n))?$$
5. Sei $P(n)$ ein Polynom vom Grad k. Zeigen Sie: $O(\log P(n)) = O(\log n)$.
6. Zeigen Sie: $(\log n)^3 \in O(\sqrt[3]{n})$ und allgemein $(\log n)^k \in O(n^\epsilon)$ für $k, \epsilon > 0$.

Aufgabe 15:

Gegeben seien die Funktionen

$f_1(n) = 2n + 5\log n$
$f_2(n) = 3n \cdot \log n + 4\sqrt{n}$
$f_3(n) = 7n^2 \cdot \sqrt{n}$

und

$g_1(n) = \sqrt{n}$
$g_2(n) = n$
$g_3(n) = \text{nlog}\, n$
$g_4(n) = n^2$

$f_i, g_j : \mathbb{N} \longrightarrow \mathbb{R}$. Geben Sie die Paare (i, j) an, für die $f_i(n) = O(g_j(n))$ gilt.

Aufgabe 16:

1. Zeigen Sie für $f(n) = O(s(n))$ und $g(n) = O(r(n))$:
   - $f(n) + g(n) = O(s(n) + r(n))$
   - $f(n) \cdot g(n) = O(s(n) \cdot r(n))$
2. Geben Sie jeweils ein Gegenbeispiel zu folgenden Behauptungen an:
   Ist $f(n) = O(s(n))$ und $g(n) = O(r(n))$, dann gilt:
   - $f(n) - g(n) = O(s(n) - r(n))$
   - $\frac{f(n)}{g(n)} = O(\frac{s(n)}{r(n)})$
3. Zeigen Sie: $a_k n^k + \cdots + a_1 n + 1 = O(n^k), \qquad a_i, n, k \in \mathbb{N}, a_i \geq 0, n > 0, k \geq 0$
4. Geben Sie zwei monoton wachsende Funktionen f(n) und g(n) an, so daß $f(n) \neq O(g(n))$ und $g(n) \neq O(f(n))$.

Aufgabe 17:

Seien a, b, c natürliche Zahlen und $n \in \{c^0, c^1, c^2, \ldots\}$.

1. Falls $T(n) = \begin{cases} b, & \text{für } n = 1 \\ aT(n/c) + bn, & \text{für } n > 1 \end{cases}$
   so folgt
   $$T(n) = \begin{cases} O(n), & \text{falls } a < c \\ O(n \log n), & \text{falls } a = c \\ O(n^{\log_c a}), & \text{falls } a > c \end{cases}$$
2. Falls $T(n) = \begin{cases} b, & \text{für } n = 1 \\ aT(n/c) + bn^2, & \text{für } n > 1 \end{cases}$
   so folgt
   $$T(n) = \begin{cases} O(n^2), & \text{falls } a < c^2 \\ O(n^2 \log n), & \text{falls } a = c^2 \\ O(n^{\log_c a}), & \text{falls } a > c^2 \end{cases}$$

Hinweis: Setzen Sie $r = a/c$ bzw. $r = a/c^2$ und nutzen Sie, die Darstellung $T(n) = bn \cdot \sum_{i=0}^{\log_c n} r^i$ bzw. $T(n) = bn^2 \cdot \sum_{i=0}^{\log_c n} r^i$. Diese Darstellungen kann man jeweils durch eine einfache Induktion bestätigen. Unterscheiden Sie die Fälle $r < 1$, $r = 1$ und $r > 1$.

Aufgabe 18:

Lösen Sie die folgenden drei Rekursionsgleichungen auf und bestimmen Sie die Größenordnung von $T(n)$:

1. $T(n) = 8T(n/3) + n^2$
2. $T(n) = 9T(n/3) + n^2$
3. $T(n) = 10T(n/3) + n^2$

Aufgabe 19:

Finden Sie für die folgenden Rekursionsgleichungen die Größenordnungen für $T(n)$:

1. $T(n) = 2T(\sqrt{n}) + \log n$.

2. $T(n) = n + T(n/4) + T(3n/4)$.

Abstrahieren Sie von den Anfangswerten und der Ganzzahligkeit der Argumente in T.

Aufgabe 20:

In der Schule haben sie eine Methode zur Multiplikationen von Matrizen in $R^{n \times n}$ kennengelernt. Wir nehmen an, daß n eine 2er Potenz ist.

1. Wie viele Multiplikationen und Additionen sind dazu notwendig? Geben Sie jeweils die Größenordnung in der O-Notation an.

2. Strassen hat 1969 einen sehr eleganten Ansatz zur gefunden, die Laufzeit der Schulmethode zu unterbieten. Er benutzt Divide-and-Conquer und zerteilt die Matrix jeweils in vier $n/2 \times n/2$ Blöcke. Seien diese für die Matrix A mit $A_{11}, A_{12}, A_{21}, A_{22}$ bezeichnet. Zur Berechnung von $C = AB$ erzeugt er die sieben Hilfsterme $m_1 = (A_{12} - A_{22})(B_{21} + B_{22})$, $m_2 = (A_{11} + A_{22})(B_{11} + B_{22})$, $m_3 = (A_{11} - A_{21})(B_{11} + B_{12})$, $m_4 = (A_{11} + A_{12})B_{22}$, $m_5 = A_{11}(B_{12} - B_{22})$, $m_6 = A_{22}(B_{21} - B_{11})$, $m_7 = (A_{21} + A_{22})B_{11}$ und fügt sie wie folgt zusammen: $C_{11} = m_1 + m_2 - m_4 + m_6$, $C_{12} = m_4 + m_5$, $C_{21} = m_6 + m_7$, $C_{22} = m_2 - m_3 + m_5 - m_7$.

   Berechnen Sie die Größenordnung der Anzahl von Elementaroperationen (Additionen und Multiplikationen).

Aufgabe 21:

Ben Bitdiddle hat sich ein neues Multiplikationsverfahren für zwei natürliche Zahlen ausgedacht, das er uns an dem Beispiel der Zahlen 9 und 14 gerne vorführt:

9	14
~~4~~	~~28~~
~~2~~	~~56~~
1	112
total	= 126

Er erklärt sein Verfahren wie folgt:

> *Die linke Seite wird ohne Rest halbiert und die rechte Seite verdoppelt. Dabei werden die Zeilen mit gerader linker Seite gestrichen und die nicht gestrichenen rechten Seiten aufaddiert.*

1. Beschreiben Sie das Verfahren durch Angabe eines Java Programmes `mult`.

2. Beweisen Sie die Korrektheit des Ansatzes.

3. Wieviel Iterationen benötigt Ben Bitdiddles Verfahren? Ist es besser als das Verfahren der Schulmethode?

Aufgabe 22:

Innerhalb eines zufällig mit den n Ganzzahlelementen $a[0], \ldots, a[n-1]$ gefüllten Arrays a soll das minimale und maximale Element gefunden werden. Sie können n als gerade voraussetzen.

1. Zeigen Sie, daß $n - 1$ Vergleiche notwendig sind, um eines der beiden Elemente zu finden und ein solches Verfahren existiert.

2. Zeigen sie, daß ein einfaches Scan-Verfahren mit den Variablen `minsofar` und `maxsofar` existiert, das höchstens $3n/2 - 2$ Vergleiche benötigt.

3. (Herausforderung) Zeigen Sie, daß mindestens $3n/2 - 2$ Vergleiche zur Lösung des Problems notwendig sind.

Aufgabe 23:

Innerhalb eines zufällig mit Ganzzahlelementen (Schlüsseln) `a[0]`,...,`a[n-1]` gefüllten Arrays `a` soll das minimale und maximale Element gefunden werden. Schreiben Sie ein Java Programm `MinMax`, das zur gleichzeitigen Suche beider Elemente möglichst wenige (Schlüssel-) Vergleiche durchführt. Ihr Programm sollte die triviale Anzahl von $2(n-1)$ Vergleichen wesentlich unterbieten. Nutzen Sie eine Variable `call`, die protokolliert, wie häufig ein Schlüsselvergleich `less` bzw. `greater` durchgeführt wurde.

1. Schreiben Sie das Programm ohne Divide-and-Conquer.

2. Schreiben Sie das Programm mit Divide-and-Conquer.

Zur Vereinfachung können Sie sich auf $n = 2^k$ für ein ganzzahliges k beschränken.

Aufgabe 24:

Für den Inhalt A eines unregelmäßigen n-Ecks in der Ebene mit den (entgegen dem Uhrzeigersinn gelesenen) Endpunkten $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n) = (x_0, y_0)$ gilt:

$$A = \frac{1}{2} \sum_{k=0}^{n-1} (x_k y_{k+1} - x_{k+1} y_k)$$

Dabei ist $\sum_{k=0}^{n} a_k = a_0 + a_1 + \cdots + a_n$.

1. Schreiben Sie ein Java Programm `Polygon`, das die Koordinaten der vier Punkte in der Reihenfolge x_0,y_0,x_1,y_1,...,x_3,y_3 einliest und den Flächeninhalt dieses Vierecks berechnet. Nutzen Sie dabei die Arrays `x` und `y` vom Typ `double` (Initialisierung z.B. von `x` durch den Aufruf `double[] x = new double[5]` und Einlesen z.B. von `x[0]` durch den Aufruf `x[0] = Double.valueOf(args[0]).doubleValue()`). Führen Sie die Berechnung des Flächeninhalts `A` in der Schleife `for(int i=0;i<4;i++)` durch.

2. Erweitern Sie Ihr Programm von Vier- auf n-Ecke. Dabei sei n der erste Aufrufparameter.

Aufgabe 25:

In dieser Aufgabe sollen Matrixoperationen implementiert werden. Zur Erinnerung: Matrizen sind Tabellen bestehend aus Zahlen. Die Addition $A + B$ und die Multiplikation AB für zwei Matrizen $A = (a_{ij})$ und $B = (b_{ij})$ sind wie folgt definiert: $C = A + B$ mit $c_{ij} = a_{ij} + b_{ij}$ und $C = AB$ mit $c_{ij} = \sum_k a_{ik} b_{kj}$. Im ersten Fall müssen die Dimensionen von A und B übereinstimmen, im zweiten Fall muß die Spaltenanzahl von A gleich der Zeilenanzahl von B sein. Ein Beispiel:

Für $A = \begin{pmatrix} 1 & 3 \\ 0 & 4 \\ -2 & 5 \end{pmatrix}$ und $B = \begin{pmatrix} 3 & 1 \\ 0 & 2 \end{pmatrix}$ ist $B + B = \begin{pmatrix} 6 & 2 \\ 0 & 4 \end{pmatrix}$ und $AB = \begin{pmatrix} 3 & 7 \\ 0 & 8 \\ -6 & 8 \end{pmatrix}$.

Ergänzen Sie die folgende Klasse `Matrix` einer ganzzahligen Matrix an den angedeuteten Stellen und testen Sie die Klasse mit dem angegebenen Beispiel. Hinweis: Zur Dimensionsabfrage des zwei-dimensionalen Arrays M können Sie die Variablen `M.length` und `M[0].length` nutzen.

```
public class Matrix {
  public int M[][];
  Matrix (int z, int s) { M = new int[z][s]; }
  public Matrix add (Matrix A) ...
  public Matrix mult (Matrix A) ... }
```

Wenn Sie möchten, können Sie eine Matrix und die darauf agierenden Matrixoperationen anstatt für Ganzzahlen für Körperelemente und damit auch für Bruchzahlen verwirklichen.

Aufgabe 26:

1. Schreiben Sie in Java eine rekusive Funktion **public long** bk(**long** n, **long** k), die den Binomialkoeffizienten $\binom{n}{k}$ auf folgende Weise berechnet:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{falls } 0 < k < n \\ 1 & \text{falls } k = 0 \text{ oder } k = n \\ 0 & \text{falls } k > n \end{cases}$$

2. Eine weitere Möglichkeit, den Binomialkoeffizienten rekursiv zu bestimmen, ist folgende:

$$\binom{n}{k} = \begin{cases} \frac{n}{k} \cdot \binom{n-1}{k-1} & \text{falls } 0 < k < n \\ 1 & \text{falls } k = 0 \text{ oder } k = n \\ 0 & \text{falls } k > n \end{cases}$$

   Bestimmen Sie sowohl für die obige Rekursionsformel als auch für die Rekursionsformel in Aufgabenteil a) die Anzahl der rekursiven Aufrufe, die nötig sind, um $\binom{4}{2}$ zu berechnen.

3. Welche der Formeln berechnet $\binom{n}{k}$ effizienter? Begründen Sie Ihre Antwort.

Aufgabe 27:

In einer einfach verketteten Liste von Ganzzahlen soll am Listenanfang `Kopf` Elemente eingefügt und entfernt werden können. Vervollständigen Sie die folgende Java Spezifikation

```
class Element {
  Element succ; int i;
  Element(Element succ, int i) {
    this.succ = succ; this.i = i;
  }
}
public class Liste {
  Element Kopf;
  Liste() { Kopf = null; }
  public void Einfuegen(int i) ...
  public void Entfernen() ...
  public String toString() ...
  public static void main(String args[]) {
```

```
    Liste L = new Liste();
    for(int i=0;i<10;i++) {
      L.Einfuegen(i); System.out.println(L);
    }
    for(int i=0;i<5;i++)  {
      L.Entfernen();  System.out.println(L);
    }
  }
}
```

Aufgabe 28:

Schreiben Sie ein Java-Programm, daß ganze Zahlen in einer verketteten Liste speichert. Neue Werte werden dabei stets hinten an die Liste angehängt. Desweiteren soll das Programm die Möglichkeit bieten, die Liste beginnend mit dem ersten Element auszugeben und die Liste umzudrehen. Das Umdrehen der Liste soll ohne Verwendung einer zweiten Liste oder zusätzliche **new**-Aufrufe erfolgen. Die Kommandos werden dem Programm beim Aufruf als Folge von Zeichen über die Kommandozeilen-Parameter übergeben. Ganze Zahlen in dieser Folge müssen an die Liste angehängt werden. Beim Buchstaben `t` bzw. `p` soll die aktuelle Liste umgedreht bzw. ausgegeben werden.

Beispiel:

```
computer@~ > java TestIntList 2 p 1 3 p 2 p t t p t p
2
2 1 3
2 1 3 2
2 1 3 2
2 3 1 2
```

Aufgabe 29:

Ein Ring ist eine Datenstruktur, die, wie der Name schon sagt, Elemente in einem Zyklus miteinander verbindet. Ringe können in zwei Teile aufgeteilt und auch miteinander verschmolzen werden. Der Ring besitzt einen Einstiegspunkt `first` an dem (Ganzzahl-) Elemente eingefügt werden können. Implementieren Sie den wie folgt spezifizierten Datentyp `Ring` als doppelt verkettete Liste in Java.

Erzeugen Sie im Testlauf einen Ring mit den Elementen 1..8, spalten Sie die Elemente 3..6 in einen neuen Ring ab und fügen Sie den zweiten Ring, beginnend mit seinem zweiten Element, hinter das erste Element des ersten Rings ein.

```
class Item {  Item prev, next;  public int value;  }
public class Ring {
  private Item first;
  public Ring();
   // Konstruktor fuer leeren Ring
  public boolean isEmpty();
   // Abfrage auf leeren Ring
  public boolean isFirst(Item l);
   // Anfrage auf erstes Element
  public Item getFirst();
   // Ausgabe erstes Element
```

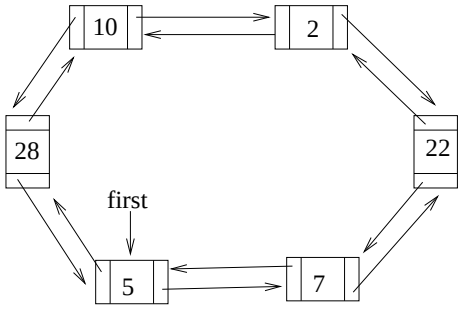
```
  public Item getNext(Item l);
   // Ausgabe nachfolgendes Element
  public void append(int val);
   // fuegt Element mit Wert val an
  public Ring split(Item s, Item t);
   // trennt den Teilring [s..t] vom aktuellen Ring
   // ab und liefert ihn zurueck
  public void merge(Item insertPos, Ring r,
                    Item beginWith);
   // fuegt den Ring r ab der Position beginWith
   // in den aktuellen Ring ein, und zwar hinter
   // der Position insertPos
  public String toString();
}
```

Aufgabe 30:

Ein Ring ist eine Datenstruktur, die Elemente in einem Zyklus miteinander verbindet. Ringe können in zwei Teile aufgeteilt und auch miteinander verschmolzen werden. Die Elemente in dem Ring sind als ganze Zahlen in einer doppelt verketteten Liste in Java wie folgt verwirklicht.

```
class Item {
  Item prev, next;
  int value;
}

class Ring {
  Item first;
  ...
}
```



Der Ring besitzt einen Einstiegspunkt `first` vom Typ `Item`. Geben Sie die Java-Implementationen der folgenden Funktionen an.

1. `public void insert(int val)`: Erzeugt ein Element mit Wert `val` und fügt es *vor* die Stelle `first` ein.
2. `public Ring split(Item s, Item t)`: Trennt einen Teilring von Position `s` zur Position `t` vom aktuellen Ring ab und und gibt ihn zurück.
3. `public void merge(Item i, Ring r, Item b)`: Fügt den Ring `r` ab Position `b` hinter der Position `i` in den aktuellen Ring ein.

In allen Aufgabenteilen können Sie voraussetzen, daß alle beteiligten Ringe vor und nach den Operationen mehr als ein Element enthalten. Außerdem brauchen Sie keine Fehlerbehandlung in den Operationen vorzusehen.

Aufgabe 31:

Wir beweisen den folgenden Satz konstruktiv durch die Angabe eines Programmes: Ist der größte gemeinsame Teiler von a und b, kurz $ggT(a, b)$, mit $a, b \in \mathbb{N}$ gleich d, so gibt es ein x und ein y mit $ax + by = d$.

Dazu schauen wir uns den *euklidischen Algorithmus* $ggT(a,b) = ggT(b, a \bmod b)$ mit $ggT(a,0) = a$ genauer an. Zu a und b gibt es ein Paar $(x', y') = (1,0)$ mit $a = x' \cdot a + y' \cdot b$ und ein Paar $(x'', y'') = (0,1)$ mit $b = x'' \cdot a + y'' \cdot b$. Für $a \bmod b = a - qb$ mit $q = a$ div b findet sich die Darstellung $(x' - qx'')a + (y' - qy'')b$.

1. Schreiben Sie aufbauend auf diesen Sachverhalt ein Java Programm GGT zur Berechnung von x und y. Dabei sollen Sie x und y als statische Klassenvariablen festgelegen und eine rekursive Prozedur `ggt` mit den sechs Parametern `a`, `xa`, `ya`, `b`, `xb` und `yb` implementieren (`xa` steht für x', `ya` steht für y', `xb` steht für x'' und `yb` für y''). Ist der Parameter `b` gleich Null, so werden `x` und `y` auf `xa` bzw. `ya` festgelegt und `a` zurückgegeben. Der Aufruf ist `ggt(a,1,0,b,0,1)`. Hinweise: Der Java-Ausdruck `a%b` berechnet a mod b und der Ausdruck `xa-(a/b)*xb` berechnet $x' - (a \text{ div } b)x''$.

   Testen Sie Ihr Programm an dem Beispiel $a = 168$ und $b = 62$.
   (Die Ausgabe sollte `2=-7*168+19*62` ergeben).

2. Ergänzen Sie das Java Programm durch die Berechnung des kleinsten gemeinsamen Vielfachen kgV. Hinweis: $kgV(a,b) = ab/ggT(a,b)$.

Aufgabe 32:

Die Quadratwurzel einer positiven reelen Zahl a läßt sich näherungsweise über die folgende Iterationsformel bestimmen: $x_{n+1} = (x_n + a/x_n)/2$. Schreiben sie ein Java Programm, das die Wurzel der Eingabe berechnet. Dabei gilt ein Iterationswert als gut genug, falls er von dem nachfolgenden Wert um nicht mehr als 0.00001 abweicht.

Aufgabe 33:

Gegeben seien folgende Definitionen:

$$f(n) = g(n,0)$$

und

$$\begin{aligned} g(n,i) &= 0, \text{ falls } n \leq 0 \text{ gilt, und} \\ g(n,i) &= g(n-1-2i, i+1) + 1, \text{ sonst.} \end{aligned}$$

Schreiben sie ein Java-Programm, das für jedes übergebene Argument n aus den natürlichen Zahlen den Wert $f(n)$ berechnet und ausgibt! Können Sie in Worten beschreiben, welche Funktion durch f berechnet wird?

Aufgabe 34:

In *Rekursien* treffen sich die drei Einwohner Herr Ackermann, Herr Fibonacci und Frau Ulam. Bald ist eine heftige Diskussion über die Schönheit der von Ihnen erdachten Funktionen entbrannt. Herr Ackermann betont das außergewöhnliche Wachstum seiner Funktion, Herr Fibonacci unterstreicht den inneren Zusammenhang der Funktionswerte, während Frau Ulam sich an den vielen Wegen zur Eins erfreut...

1. In der Zwischenzeit sollen Sie zeigen, wie leicht sich alle drei Funktionen in Java implementieren lassen. Berechnen Sie für Herrn Ackermann $a(3,5)$, für Herrn Fibonacci $F(25)$ und für Frau Ulam *ulam*(71). Als Hinweis soll Ihnen die Implementation der Ackermannfunktion dienen:

```
public class Acker {
  static int a(int x, int y) {
    System.out.println("a("+x+","+y+") =");
    if (x==0) return y+1;
    if (y==0) return a(x-1,1);
    return a(x-1,a(x,y-1)); }
  public static void main(String args[]) {
    int x = Integer.parseInt(args[0]);
    int y = Integer.parseInt(args[1]);
    System.out.println(a(x,y)); }}
```

2. Erweitern Sie Ihre Programme so, daß zusätzlich jeweils die Anzahl der rekursiven Aufrufe ausgegeben wird.

Aufgabe 35:

Gehen Sie für diese Aufgabe von folgendem *Verhalten* handelnder Personen auf einer *Bühne* aus:

- Erhält der große Zauberer *Merlin* die Nachricht `auftritt(`n, k`)` mit $n, k \in \mathbb{N}$, $n > 0$ und $0 < k \leq n$, dann *erweckt* er n *Feen* auf *Avalon* in den *Positionen* $1, \ldots, n$. Anschließend schickt er eine `gruss()`-Nachricht an die Fee auf Position k.
- Erhält Merlin die Nachricht `berichte()`, dann teilt er mit, wie oft er selbst seit seinem Auftritt `gruss()`-Nachrichten erhielt.
- Jede Fee kennt ihre Position p auf Avalon.
- Eine Fee reagiert nur dann auf eine Nachricht, wenn sie *erweckt* ist.
- Erhält eine *erweckte* Fee eine `gruss()`-Nachricht, ist sie sofort *nicht mehr erweckt*, ehe sie aber ganz untätig wird, verschickt sie noch einige `gruss()`-Nachrichten: Eine an Merlin, eine an die Fee auf Position $3p + 1$, falls diese existiert und p die eigene Position ist, und eine an die Fee auf Position $p/2$, wenn die eigene Position p eine gerade Zahl ist.

Schreiben Sie ein Java-Programm `Buehne.java`, welches einen Auftritt von Merlin für eine als Argument übergebene Anzahl n von Feen und eine ebenfalls als Argument übergebene Position k für die erste von Merlin zu grüßende Fee initiiert. Anschließend soll Merlin berichten, wie oft er selbst gegrüßt wurde. Beispiel für verschiedene Programmläufe:

```
> java Buehne 200 92
Ich bin Merlin und erhielt 65 gruss-Nachrichten.
> java Buehne 200 93
Ich bin Merlin und erhielt 1 gruss-Nachricht.
> java Buehne 200 94
Ich bin Merlin und erhielt 4 gruss-Nachrichten.
```

Aufgabe 36:

Man kann die Implementierung einer Queue (Schlange) für beliebige Objekte auf eine schon vorhandene Implementierung der entsprechenden Stack-Klasse zurückführen, auch wenn das nicht sehr effizient ist. Die begonnene Java-Implementierung von Queue sei

```
class Queue {
  Stack s, z;
  Queue () { s = new Stack (); z = new Stack (); }
  empty () { return s.empty (); }
  ...
}
```

Ergänzen Sie die Implementierung der noch fehlenden Methode `enqueue(Object o)` und `Object dequeue()`.

Aufgabe 37:

Eine Schlange Q stellt die Operationen *pophead(Q)* und *pushtail(Q,x)* zur Verfügung, die ein Element am Anfang der Schlange entnehmen bzw. am Ende anfügen. Zeigen Sie, daß sich dieser Datentyp mit zwei Stapeln S und S' so implementieren läßt, daß beide Operationen konstante amortisierte Kosten haben.

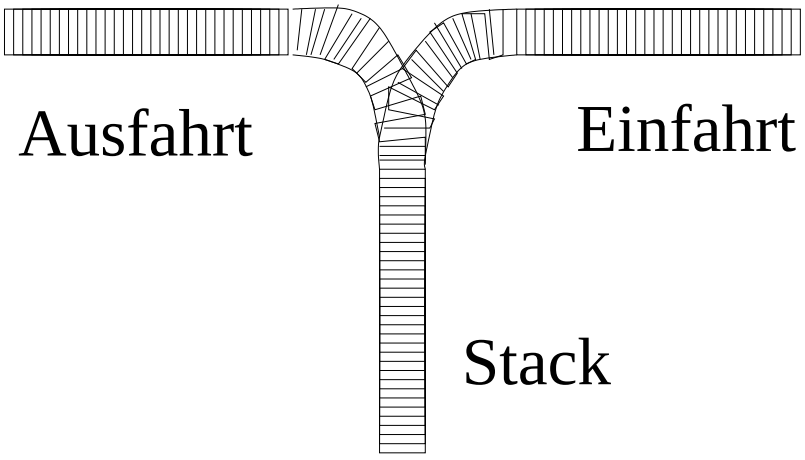
Aufgabe 38:

Implementieren Sie in Java eine Klasse `Stack`, die einen Stapel realisiert. Stellen Sie dafür in der Klasse folgende Methoden zur Verfügung:

public void init():	Initialisiert den Stapel. Der Stapel ist danach leer.
public void push(int i):	Legt `i` auf den Stapel.
public void pop():	Nimmt oberstes Element vom Stapel.
public int top():	Liefert oberstes Element des Stapels zurück.
public boolean isEmpty():	Liefert `true`, wenn der Stapel leer ist und sonst `false`.
public void print():	Gibt den Stapel mit dem obersten Element beginnend aus.

Aufgabe 39:

Einen Stack kann man sich als Rangiergleis mit Prellbock vorstellen (siehe Skizze). Die Operation `push` entspricht dem Einfahren eines Waggons von rechts auf das Rangiergleis und die Operation `pop` dem Ausfahren eines Waggons vom Rangiergleis auf das linke Gleis. Durch Mischen der `push`– und `pop`–Operationen kann nun die Reihenfolge der Wagen $1, \ldots, n$ permutiert werden. So erzeugt die Folge `push`, `pop`, `push`, `push`, `pop`, `push`, `push`, `pop`, `pop`, `pop` die Permutation 1,3,5,4,2 der Zahlen 1,2,3,4,5.



Programmieren Sie einen Algorithmus `RangierTest`, der für eine gegebene Permutation $i_1, \ldots, i_n$ der Zahlen $1, \ldots, n$ entscheidet, ob diese durch das obige Verfahren erzeugt werden kann. Es bietet sich an, den Stapel `Stack` aus der Bibliothek `java.util.Stack` zu benutzen. Die wichtigen Operationen sind `isEmpty()`, `peek()`, `pop()` und `push(Object)`.

Aufgabe 40:

Arithmetische Ausdrücke lassen sich sowohl in der Infix-, in der Prefix-, als auch in der Postfixdarstellung beschreiben. So ist `(12-3) * -4` in der Infixdarstellung `(* (- 12 3) (- 4))` in der Prefixdarstellung und `12 3 - 4 - *` in der Postfixdarstellung. Zur Unterscheidung mit dem binären Minusoperator wählen wir zur Vereinfachung die Tilde als ein unäres Minuszeichen.

Wir betrachten die Postfixdarstellung, die durch einen Punkt abgeschlossen wird. Sie kommt ohne Klammern aus und kann durch den Einsatz eines Stapels verarbeitet werden. Von links nach rechts vorranschreitend werden jeweils die Operanden auf den Stapel gelegt. Wird ein Operator erreicht, so werden die obersten Elemente vom Stapel genommen (pop) und das Ergebnis wieder hinaufgelegt (push).

Schreiben Sie eine Java Klasse, die einen Postfix Ausdruck auswertet. Testen Sie Ihr Programm mit dem obigen Ausdruck.

Aufgabe 41:

Das folgende Java-Programmfragment definiert eine Schlange, die Integer-Werte in einer Liste speichert:

```
class SillyQueue
{
  ...
  public boolean isEmpty() {...}
  public void enqueue(int i) {...}
  public int dequeue() {...}
  public void flip() {...}
  ...
}
```

Die Methode *isEmpty* liefert *true*, wenn die Schlange leer ist und *false* sonst. Die Methode *enqueue* hängt das Element am Ende (*tail*) der Schlange an. Die Methode *dequeue* entfernt ein Element am Anfang (*head*) der Schlange und liefert es zurück. Die Methode *flip* vertauscht Anfang und Ende der Schlange:

Schreiben Sie eine Java-Klasse als Ableitung der Klasse *SillyQueue*, die die Datenstruktur *Stapel* realisiert. Implementieren Sie die üblichen Stapel-Methoden *void push(int)*, *void pop()*, *int top()* und *void init()* mit Hilfe der Methoden der Klasse *SillyQueue*.

Zur Erinnerung: Die Methode *push* legt ein Element auf den Stapel drauf, die Methode *pop* entfernt das oberste Element vom Stapel, die Methode *top* liefert das oberste Element vom Stapel zurück, ohne es zu entfernen, und die Methode *init* versetzt den Stapel in den Anfangszustand zurück - der Stapel ist dann leer.

Aufgabe 42:

Gegeben sei folgende abstrakte Klasse einer Flächenform.

```
public abstract class Form {
  public abstract double Flaeche();
  public abstract double Umfang();
  public abstract String toString();
}
```

Programmieren Sie für die folgende generische Testprozedur die abgeleiteten Klassen `Rechteck` und `Kreis`.

```
public class FormTest {
  public static void main (String[] args) {
    Form[] f = new Form[2];
    f[0] = new Kreis(3);
    f[1] = new Rechteck(3,4);
    for(int i=0;i<f.length;i++) {
      System.out.println(f[i]);
    }
  }
}
```

Aufgabe 43:

Ein Körper ist eine algebraische Struktur, die axiomatisch festgelegt ist und eine additive und multiplikative Verknüpfung besitzt. Beispiele sind die rationalen, die reellen und die komplexen Zahlen und der Restklassenring Z/pZ zur Primzahl p.

Implementieren Sie aufbauend auf die folgende abstrakte Klasse die abgeleitete Klasse `Bruchzahl`, die die angegebenen Operationen für gekürzte Brüche verwirklicht. Nutzen Sie hierbei die Berechnung des ggT's nach Euklid. Der Zähler soll eine ganze, der Nenner eine natürliche Zahl sein.

```
public abstract class Koerper {
  public abstract Koerper Null();
  public abstract Koerper Eins();
  public abstract Koerper addInverses();
  public abstract Koerper multInverses();
  public abstract boolean isEqual(Koerper q);
  public abstract Koerper add(Koerper q);
  public abstract Koerper mult(Koerper q);
  public abstract String toString();
}
```

Testen Sie ihr Programm in einer Klasse mit verschiedenen Bruchaufgaben, z.B. mit (2/6 - 4/8)/6.

Aufgabe 44:

Das Spiel *Nim* ist ein beliebter Zeitvertreib. Gegeben sind dabei m Stapel von n_i Hölzern, $1 \leq i \leq m$. Abwechselnd werden von den beiden Spielern von nur einem Stapel beliebig viele Hölzer entnommen. Gewonnen ist das Spiel, falls man das letzte Hölzchen wegnehmen kann. Als Beispiel betrachten wir $m = 4$ und die Eingabe $[n_1, n_2, n_3, n_4] = [7, 5, 3, 1]$.

Die Strategie ist es, immer eine Gewinnposition zu erreichen, in der man bei einem beliebigen Zug des Gegners wieder eine Gewinnposition erreichen kann. Alle anderen Positionen sind Verlustpositionen. Nach

einigen Spielversuchen stellt sich heraus, daß neben $[0,0,0,0]$ einfache Gewinnpositionen durch $[1,1,0,0]$, $[2,2,0,0]$, $[3,3,0,0]$ oder auch durch $[3,2,1,0]$ gegeben sind.

1. Schreiben Sie ein Java-Programm, das eine 4-dimensionale Tabelle `T` füllt und bestimmt, ob die gegebene Anfangsstellung $[7,4,3,1]$ eine Gewinn- oder eine Verlustposition ist. Hier dürfen Sie vier ineinander verschachtelte Schleifen verwenden.

2. Es gilt der folgende Satz: Eine Position ist genau dann eine Gewinnposition, wenn man die Binärdarstellungen der Werte n_i untereinander schreibt und zählt, ob die Anzahl der Einsen in jeder Spalte gerade ist. Schreiben Sie ein Java-Programm, das mit Hilfe dieser Regel bei einer gegebenen Spielsituation den optimalen Zug ausführt.

Aufgabe 45:

Die Oh-Notation wird verwendet, um asymptotische Laufzeiten von Algorithmen anzugeben. Für eine Funktion $g : \mathbb{N} \to \mathbb{R}^+$ gilt

$$O(g(n)) := \{f : \mathbb{N} \to \mathbb{R}^+ : \exists c > 0, n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq cg(n)\}.$$

Für $f(n) \in O(g(n))$ sagt man, dass f asymptotisch nicht schneller als g wächst und schreibt auch $f \in O(g)$.

Analog wächst f nicht langsamer als g, wenn $f(n) \in \Omega(g(n))$ gilt mit

$$\Omega(g(n)) := \{f : \mathbb{N} \to \mathbb{R}^+ : \exists c > 0, n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \geq cg(n)\}.$$

Wir betrachten die Funktion $f(n) = 3n^2 + 4n - 3$.

1. Die Funktion f liegt in $O(n^2)$. Gib hierfür einen Beweis an, indem du entsprechende Konstanten c und n_0 berechnest.

2. Die Funktion f liegt nicht in $O(1)$. Gib auch hierfür einen Beweis an.[1]

   Hinweis: Nimm an, die Funktion f **wäre** in $O(1)$. Dann lässt sich aus der Definition von $O(1)$ ein Widerspruch herleiten.

Aufgabe 46:

Seien $f, F, g, G : \mathbb{N} \to \mathbb{R}^+$ mit $f \in O(F), g \in O(G)$. Dann gelten die folgenden Rechenregeln für die Oh-Notation:

a) Sei $\alpha \in \mathbb{R}$ konstant und $h(n) := \alpha * f(n)$. Dann gilt $h \in O(F)$.

b) Sei $h(n) := f(n) + g(n)$ und $H(n) := F(n) + G(n)$. Dann gilt $h \in O(H)$.

c) Sei $h(n) := f(n) * g(n)$ und $H(n) := F(n) * G(n)$. Dann gilt $h \in O(H)$.

d) Falls die Funktionen F und g übereinstimmen, gilt $f \in O(G)$.

Beweise die Regeln a, b und d.

[1] $O(1)$ ist eine verkürzte Schreibweise für $O(g)$ mit $g : \mathbb{N} \to \mathbb{R}^+, g(n) \equiv 1$.

Aufgabe 47:

Im folgenden sind einige Behauptungen über die Oh-Notation angegeben. Einige dieser Aussagen sind richtig, die anderen sind falsch. Gib an, welche der Aussagen korrekt und welche falsch sind und begründe deine Entscheidung.

1. $\frac{1}{n} = O(1)$
2. $2^{n+\alpha} = O(2^n)$, $\alpha \in \mathbb{R}$ konstant
3. $2^{\alpha n} = O(2^n)$, $\alpha > 1$ konstant
4. $\log_a = O(\log_b)$ für $a, b \in \mathbb{N}$

Aufgabe 48:

In der untenstehenden Tabelle sind in der linken Spalte verschiedene Funktionen aufgelistet. In die Tabelle sollst du eintragen, ob eine Funktion f in $O(g)$ liegt oder nicht. Die Tabelle soll vollständig ausgefüllt werden.

Beispiel: Die Funktion $f = 3n^2 + 4n - 3$ liegt in der Klasse $O(n^2)$, wie in einer anderen Aufgabe gezeigt wurde. Also steht hier ein "JA". Hingegen liegt diese Funktion nicht in $O(1)$, wie ebenfalls in der anderen Aufgabe bewiesen wurde. Hier steht also ein "NEIN".

	$O(1)$	$O(\log n)$	$O(\sqrt{n})$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(2^n)$	$O(5^n)$
$f(n) = 3$								
$f(n) = 2n - 7$								
$f(n) = 3n^2 + 4n - 3$	NEIN					JA		
$f(n) = 3n^2 - \log n$								
$f(n) = n^{27}$								
$f(n) = 2^{10} * 2^{n+1}$								
$f(n) = e^n$								
$f(n) = 2^{2n}$								

Aufgabe 49:

Sei $g : \mathbb{N} \to \mathbb{N}$. In dieser Aufgabe sollst du zeigen, dass die Definitionen der Oh-Notation aus der Vorlesung und vom ersten Übungsblatt übereinstimmen. Dazu beschränken wir die Definitionen von Blatt 1 auf Funktionen in die natürlichen Zahlen:

$$\begin{aligned} O(g(n)) &:= \{f : \mathbb{N} \to \mathbb{N} \mid \exists c > 0, n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq cg(n)\} \\ \Omega(g(n)) &:= \{f : \mathbb{N} \to \mathbb{N} \mid \exists c > 0, n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \geq cg(n)\}. \end{aligned}$$

In der Vorlesung wurden folgende Definitionen angegeben:

$$\begin{aligned} O_V(g(n)) &:= \{f : \mathbb{N} \to \mathbb{N} \mid \exists c_1, c_2 > 0 \; \forall n \in \mathbb{N} : f(n) \leq c_1 g(n) + c_2\} \\ \Omega_V(g(n)) &:= \{f : \mathbb{N} \to \mathbb{N} \mid g(n) \in O_V(f(n))\} \\ \Theta_V(g(n)) &:= O_V(g(n)) \cap \Omega_V(g(n)) \end{aligned}$$

1. Zeige, dass $O(g(n)) = O_V(g(n))$ gilt.
2. Zeige, dass $\Omega(g(n)) = \Omega_V(g(n))$ gilt.
3. Sei $f(n) \in O(g(n))$. Gilt dann $f(n) + g(n) \in \Theta_V(g(n))$?

Aufgabe 50:

Sei A eine Matrix mit n Zeilen und r Spalten und B eine Matrix mit r Zeilen und q Spalten. Dann liefert das Matrizenprodukt $A * B$ eine $n \times q$-Matrix. Beschreibe einen Algorithmus, um $A * B$ zu berechnen, und gib dessen Laufzeit an.

Aufgabe 51:

Das Maximum-Subarray-Problem besteht darin, für ein Array von n Zahlen die maximale Summe in einem zusammenhängenden Teilbereich des Arrays zu finden. In der Vorlesung wurden verschiedene Algorithmen vorgestellt, um dieses Problem zu lösen. Unter anderem wurde eine Divide-and-Conquer-Strategie beschrieben. Sei $T(n)$ die maximale Laufzeit dieses Algorithmus' auf einem Array der Länge n. Wir nehmen an, dass n eine Potenz von 2 ist. Mit geeigneten Konstanten c, d und a gelten die folgenden Ungleichungen:

$$\begin{aligned} T(n) &\leq 2T(\frac{n}{2}) + cn + d \\ T(1) &\leq a \end{aligned}$$

In der Vorlesung wurde anschaulich motiviert, dass $T(n) = O(n \log n)$ gilt. Gib einen Beweis hierfür an.

Hinweis: Du kannst die Aussage z.B. mit vollständiger Induktion beweisen.

Aufgabe 52:

Das *Maxsummenproblem* wurde in der Vorlesung wie folgt definiert: Finde ein Index-Paar (i, j) in einem Array `a[1..n]` für das $f(i, j) = a_i + \ldots + a_j$ maximal ist. Als Rechenschritte zählen arithmetische Operationen und Vergleiche. Implementieren Sie die vier verschiedenen Ansätze zur Lösung des Problems.

Der naive Algorithmus Es sollen alle $f(i, j)$ berechnet werden und dann der maximale f-Wert ermittelt werden. Offensichtlich genügen zur Berechung von $f(i, j)$ genau $j - i$ Additionen. Der Algoritmus startet mit $\max \leftarrow f(1, 1)$ und aktualisiert $\max$ wenn nötig.

Der normale Algorithmus Der naive Ansatz berechnet $a_1 + a_2$ für $f(1, 2), f(1, 3), \ldots, f(1, n)$, also $(n - 1)$-mal.

Besser geht's mit folgender Erkenntnis. Es gilt:

$$f(i, j + 1) = f(i, j) + a_{j+1}$$

Damit braucht man für alle $f(i, \cdot)$-Werte genau $(n - i)$ Additionen.

Divide-And-Conquer Annahme: $n = 2^k$. Unterteilung in drei Klassen:

$1 \leq i, j \leq n/2$ (Divide-And-Conquer)
$1 \leq i \leq n/2 < j \leq n$ (Direkt)
$n/2 < i \leq j \leq n$ (Divide-And-Conquer)

$g(i) = a_i + \ldots + a_{n/2}$ und $h(j) = a_{n/2+1} + \ldots + a_j$

Dann gilt: $f(i,j) = g(i) + h(j)$

Um f zu maximieren, maximiere g und h einzeln.

Berechne nacheinander $g(n/2) = a[n/2], g(n/2-1), \ldots, g(1)$ und den max. g Wert in $(n/2-1)$ Vergleichen und $(n/2-1)$ Additionen. Berechne h analog.

Der clevere Algorithmus Wir wollen das Problem in $[1..n]$ lösen und verwalten dabei nach dem Lesen von a_k in $\max$ den größten Wert von $f(i,j)$ aller Paare (i,j) für $1 \leq i \leq j \leq k$. Für $k = 1$ setzen wir $\max$ auf a_1.

Wenn a_{k+1} gelesen ist, aktualisiere $\max^*_{neu} = \max\{\max^*_{alt} + a_{k+1}, a_{k+1}\}$.

Für $\max_{neu}$ kommen folgende Paare in Frage:
$1 \leq i \leq j \leq k$ (maximaler Wert $\max_{alt}$)
$1 \leq i \leq k, j = k+1$ (maximaler Wert $\max^*_{neu}$)

Aufgabe 53:

Ein Stack (*Stapel*) ist ein abstrakter Datentyp, den man mit einem Stapel von Tellern vergleichen kann. Bei einem Tellerstapel kann man entweder einen weiteren Teller oben drauflegen oder den obersten Teller wegnehmen. Diese Strategie wird als LIFO (*last in first out*) bezeichnet.

Ein Stack beschränkter maximaler Grösse bietet folgende Operationen an:

init(S): Initialisiert S als leeren Stack.

push(S,x): Legt das Objekt x "zuoberst" auf den Stack S. Falls der Stack S bereits voll ist (d.h. die maximale Grösse erreicht hat), gibt es eine Fehlermeldung.

top(S): Liefert das "oberste" Objekt x des Stacks S. Falls der Stack S leer ist, gibt es eine Fehlermeldung.

pop(S): Liefert das "oberste" Objekt x des Stacks S und entfernt es vom Stack. Falls der Stack S leer ist, gibt es eine Fehlermeldung.

empty(S): Liefert $TRUE$, falls der Stack S leer ist, andernfalls $FALSE$.

full(S): Liefert $TRUE$, falls der Stack S voll ist, andernfalls $FALSE$.

1. Schreibe ein Programm, das einen Stack und seine Operationen zur Verfügung stellt. Der Stack soll maximal 451 Elemente enthalten. Die Elemente sind ganze Zahlen.
2. Verwende den Stack aus Aufgabenteil a, um eine Zahlenfolge, die der Benutzer bzw. die Benutzerin eingibt, umzukehren.

Aufgabe 54:

Gegeben sei die Rekursionsgleichung $T(n) = 9T(n/3) + n, T(1) = 1$. Gilt dann $T(n) \in O(n^2)$?

Aufgabe 55:

Seien $a, b \in \mathbb{R}$ und $n \in \mathbb{N}$. Wir führen die **Gaussklammern** ein, mit denen Zahlen explizit gerundet werden können. Mit $\lfloor a \rfloor$ wird die grösste ganze Zahl bezeichnet, die kleiner oder gleich a ist. Hingegen ist $\lceil a \rceil$ die kleinste ganze Zahl, die grösser oder gleich a ist.

Beispiele:

$$\lfloor 3 \rfloor = \lceil 3 \rceil = 3 \qquad \lfloor 3.6 \rfloor = 3 \qquad \lceil 3.6 \rceil = 4$$
$$\lfloor -2 \rfloor = \lceil -2 \rceil = -2 \qquad \lfloor -2.3 \rfloor = -3 \qquad \lceil -2.3 \rceil = -2$$
$$\lfloor 0 \rfloor = \lceil 0 \rceil = 0$$

Zeige folgende Eigenschaften der Gaussklammern:

1. $\lfloor a + n \rfloor = \lfloor a \rfloor + n$
2. $\lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil = n$
3. $\lfloor a + b \rfloor - 1 \leq \lfloor a \rfloor + \lfloor b \rfloor \leq \lfloor a + b \rfloor$

Aufgabe 56:

1. Was macht der folgende Algorithmus `TuWas` mit der Eingabefolge X im Bereich 1 bis n?

```
void TuWas (int X[]) {
  int msum = -unendlich;
  for(int l = 1;l <= n; l++) {
    for(int r = l;r <= n; r++) {
      int sum = 0;
      for(int i = l;i <= r; i++)
        sum = sum + X[i];
      if (sum > msum) msum = sum;
    }
  }
}
```

2. Bestimmen Sie die Laufzeit dieses Verfahrens in Abhängigkeit zur Größe n (in O-Notation).
3. Welche anderen Verfahren zur Bestimmung von `msum` kennen Sie und was ist ihre Komplexität (O-Notation, kurze Begründung).

Aufgabe 57:

Die Funktion *random()* liefere für die ersten 30 Aufrufe die folgende Sequenz von Werten aus $\{0, 1\}$: 11100110010100110010110010011. Die Funktion *randomheight()* sei wie folgt definiert:

function *randomheight()* : integer;
var height : integer;

```
begin
    height := 0;
    while random() = 1 do height := height+1;
    randomheight := height;
end
```

a) Die Funktion *randomheight()* wird für die Bestimmung der Höhen von neu eingefügten Elementen in einer Skip-Liste verwendet. Geben Sie die Struktur der Skip-Liste nach dem Einfügen der Schlüssel 51, 82, 3, 14, 5, 61, 7, 24, 6 und 19 an.

b) Berechnen Sie die Kosten (Anzahl der Vergleiche), um auf alle Elemente dieser Skip-Liste genau einmal zuzugreifen. Vergleichen Sie diese Kosten mit den entsprechenden Zugriffskosten in einer gewöhnlichen, einfach verketteten Liste.