

Aufgabe 1 [40 Punkte]

Zur sortierten Abspeicherung von ganzen Zahlen wird ein `int`-Array wie folgt benutzt: In Element 0 steht die Anzahl n der Einträge. In den Elementen 1 bis n stehen die eingetragenen Zahlen in aufsteigender Reihenfolge. Die restlichen Elemente $n+1$, $n+2$ usw. bleiben unbenutzt und können beliebige Werte aufweisen. Ein solches Array wird hier „sortiertes Array“ genannt.

Beispiel eines sortierten Arrays mit 8 Elementen und 6 Einträgen (* bezeichnet unbenutzte Elemente, die einen beliebigen Wert aufweisen können).

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
6	3	11	18	21	21	30	*	*

- a) Programmieren Sie zu dem Methoden-Kopf `int[] neu()` einen Methoden-Rumpf, so dass der Aufruf `neu()` ein leeres sortiertes Array mit Platz für 4 Einträge liefert. [2 Punkte]

[0]	[1]	[2]	[3]	[4]
0	*	*	*	*

- b) Programmieren Sie zu dem Methoden-Kopf `boolean sortiert(int[] a)` einen Methoden-Rumpf, so dass der Aufruf `sortiert(a)` genau dann `true` liefert, wenn `a` ein sortiertes Array ist. [6 Punkte]

- c) Programmieren Sie zu dem Methoden-Kopf `int[] sortiereein(int[] a, int e)` einen Methoden-Rumpf, der die Zahl `e` in ein sortiertes Array `a` einsortiert. Dabei müssen evtl. einige Einträge in dem Array verschoben werden. In dieser Teilaufgabe darf davon ausgegangen werden, dass in `a` noch mindestens ein Element frei ist, um einen Eintrag `e` aufzunehmen. Der Aufruf `sortiereein(a, e)` soll einen Verweis auf das sortierte Array liefern, in das `e` einsortiert ist. [10 Punkte]

Dazu ein Beispiel:

Parameter a:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
5	3	4	8	8	14	*	*	*

Parameter e: 7

Gelieferter Rückgabewert:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
6	3	4	7	8	8	14	*	*

- d) Programmieren Sie zu dem Methoden-Kopf `int suche(int[] a, int e)` einen Methoden-Rumpf, der den Index des Eintrags `e` liefert. Falls `e` in dem sortierten Array `a` nicht vorkommt, soll der Wert 0 geliefert werden. [4 Punkte]

Wenn Sie eine Lösung finden, die auch bei einer hohen Anzahl von Einträgen besonders schnell arbeitet, erhalten Sie 8 Punkte zusätzlich.

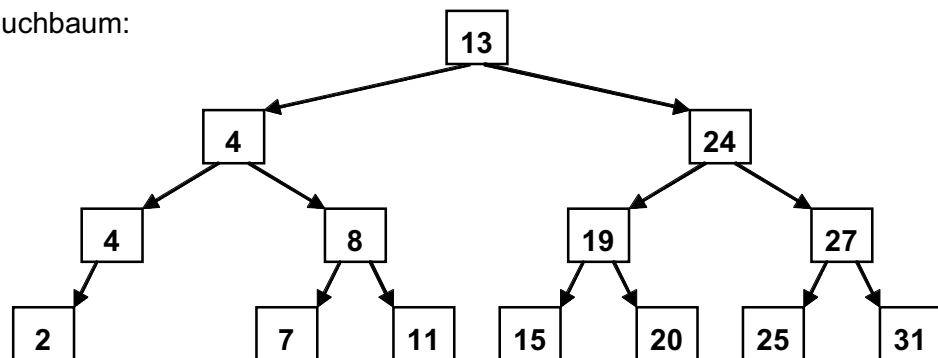
- e) Programmieren Sie zu dem Methoden-Kopf `Knoten Suchbaumwurzel(int[] a)` einen Methoden-Rumpf, der die Einträge des sortierten Arrays `a` in einen Suchbaum übernimmt und einen Verweis auf dessen Wurzel liefert. Dabei soll der Suchbaum nicht zu einer linearen Liste entarten, sondern möglichst vollständig sein. Aus diesem Grund soll ein Eintrag in der Mitte des sortierten Arrays `a` die Wurzel des Suchbaums bilden. Ein Eintrag in der Mitte der verbleibenden linken Hälfte des sortierten Arrays soll den linken Nachfolger der Wurzel bilden. Entsprechend soll ein Eintrag in der Mitte der verbleibenden rechten Hälfte des sortierten Arrays den rechten Nachfolger der Wurzel bilden. Dieses Verfahren wird fortgesetzt, bis der Suchbaum jedes Element des sortierten Arrays `a` genau einmal enthält. Es empfiehlt sich, zur Lösung dieser Aufgabe eine rekursive Methode zu programmieren. [13 Punkte]

Dazu ein Beispiel:

a:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]
14	2	4	4	7	8	11	13	15	19	20	24	25	27	31	*	*

Erzeugter Suchbaum:



Hinweis: Für den Suchbaum sei die Klasse `Knoten` wie üblich definiert:

```

class Knoten
{ int Zahl; Knoten links, rechts;

  Knoten(int Zahl)
  { this.Zahl = Zahl; links = rechts = null; }
}

```

- f) Vergleichen Sie ein sortiertes Array, eine sortierte Liste und einen Suchbaum jeweils für eine große Anzahl von zufälligen Einträgen. Geben Sie für die folgenden Operationen und Datenstrukturen durch Ankreuzen an, ob sie vergleichsweise schnell oder langsam ausgeführt werden. Eine Operation gilt als langsam, wenn im Mittel die Hälfte der Elemente oder fast alle Elemente durchlaufen werden müssen. [5 Punkte]

	Sortiertes Array	Sortierte Liste	Suchbaum
Einsortieren	<input type="checkbox"/> schnell <input type="checkbox"/> langsam	<input type="checkbox"/> schnell <input type="checkbox"/> langsam	<input type="checkbox"/> schnell <input type="checkbox"/> langsam
Suchen	<input type="checkbox"/> schnell <input type="checkbox"/> langsam	<input type="checkbox"/> schnell <input type="checkbox"/> langsam	<input type="checkbox"/> schnell <input type="checkbox"/> langsam
Löschen	<input type="checkbox"/> schnell <input type="checkbox"/> langsam	<input type="checkbox"/> schnell <input type="checkbox"/> langsam	<input type="checkbox"/> schnell <input type="checkbox"/> langsam

Aufgabe 2 [25 Punkte]

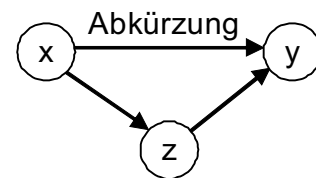
Ein gerichteter Graph wird durch die aus der Vorlesung bekannten Klassen `Graph`, `Knoten` und `Kante` repräsentiert:

```
class Graph
{ Knoten Kopf, Fuss; // Liste der Knoten des Graphen.
  ... // Methoden zur Bearbeitung des Graphen.
}

class Knoten
{ String Bez; Knoten Nf; // Nf zeigt auf den Nachfolger-Knoten.
  Kante Kopf, Fuss; // Kanten, die von einem Knoten ausgehen.
}

class Kante
{ String Bez; Kante Nf; // Nf zeigt auf die Nachfolger-Kante.
  Knoten Kante; // Knoten, zu dem die Kante hinführt.
}
```

Hier wird definiert: Eine Kante von einem Knoten x zu einem Knoten y heißt „Abkürzung“, wenn es einen Knoten z und Kanten von x nach z sowie von z nach y gibt.



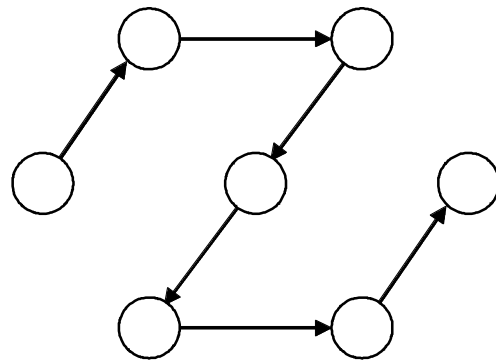
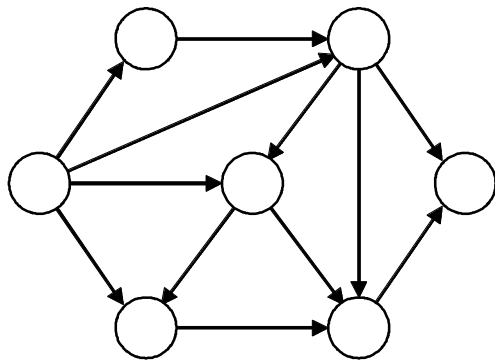
a) Programmieren Sie zu dem Methoden-Kopf

`boolean Abk(Knoten k, Kante e)` einen Methoden-Rumpf, der angibt, ob die von Knoten k ausgehende Kante e eine Abkürzung ist. Die Methode `boolean Abk(Knoten k, Kante e)` sei in der Klasse `Graph` angeordnet. [10 Punkte]

- b) Programmieren Sie zu dem Methoden-Kopf `void loescheAbk()` einen Methoden-Rumpf, der aus einem beliebigen Graphen alle Abkürzungen löscht. Die Methode `void loescheAbk()` sei in der Klasse `Graph` angeordnet.

Hinweis: Je nach dem von Ihnen eingeschlagenen Lösungsweg kann es nötig sein, dass Sie in die Klasse `Kante` eine zusätzliche Variable einfügen müssen. [15 Punkte]

Ein Beispiel zu dieser Teilaufgabe: Links der gegebene Graph, rechts der Graph, nachdem durch den Aufruf `loescheAbk()` alle Abkürzungen entfernt worden sind.



Aufgabe 3 [15 Punkte]

Betrachten Sie die nebenstehende Methode.

- a) Ist die Rekursion der Methode `int Fkt(int x)` linear?
Geben Sie bitte eine Antwort mit Begründung. [2 Punkte]

```
int Fkt(int x)
{ System.out.println(x);
  if (x < 9)
    return x;
  if (Fkt(x/2) % 2 == 0)
    return Fkt(x - 10);
  else
    return Fkt(x - 20);
}
```

- b) Ist die Rekursion der Methode `int Fkt(int x)`
schlicht? Geben Sie bitte eine Antwort mit Begründung. [2 Punkte]

- c) Welche Ausgabe liefert der Aufruf `System.out.println(34 + Fkt(34));`
[11 Punkte]

Aufgabe 4 [20 Punkte]

In einem Projekt soll der Benzinverbrauch von verschiedenen Fahrzeugen untersucht werden. Dabei ist jedes Fahrzeug durch die folgenden Eigenschaften charakterisiert:

E₁: den Typ des Fahrzeugs, z.B. "PKW-Kombi Fabrikat X"

E₂: die Höchstgeschwindigkeit des Fahrzeugs (ganzzahlig in km/h), z.B. 172

E₃: die Bezinverbauchsfunktion $b(v)$, die für jede Geschwindigkeit v (ganzzahlig in km/h) den Bezinverbrauch (in Liter / 100 km als Gleitkommazahl) angibt, z.B.

$$b(v) = 6 + \frac{(v - 91)^2}{1920} + 0,024 \cdot v$$

Ein Sportwagen könnte dagegen folgende Eigenschaften aufweisen (als Beispiel):

Typ "Sportwagen Fabrikat Y", Höchstgeschwindigkeit 210,

Bezinverbauchsfunktion $b(v) = 7,8 \cdot 10^{-6} \cdot (v - 111)^2 \cdot (v + 90) + 0,09 \cdot v - 2,6 \cdot 10^{-4} \cdot v^2$

Untersuchungsgegenstände des Projekts sind:

U₁: Bestimmung der Geschwindigkeit mit dem geringsten Bezinverbrauch
(Wenn mehrere Geschwindigkeiten den gleichen geringsten Bezinverbrauch aufweisen, soll die größte dieser Geschwindigkeiten bestimmt werden).

U₂: Bestimmung der größten Geschwindigkeit, deren Bezinverbrauch nicht mehr als 10% über dem geringsten Bezinverbrauch des jeweiligen Fahrzeugs liegt.

Alle Programmteile zur Behandlung der Untersuchungsgegenstände U₁ und U₂ sind in einer Oberklasse `Fahrzeug` anzusiedeln. Die Programmteile zur Realisierung der Fahrzeugeigenschaften E₁, E₂ und E₃ sollen dagegen in Unterklassen davon angesiedelt sein, die spezifisch für das jeweilige Fahrzeug sind, z.B. `PKWKombiX` oder `SportwagenY`.

Die Anzahl der untersuchten Fahrzeuge kann beliebig groß sein. Für jedes Fahrzeug wird ein entsprechendes Objekt erzeugt.

Die Methoden, die U₁ und U₂ behandeln, sollen als Rückgabewert die entsprechenden Ergebnisse liefern.

a) Programmieren Sie die Oberklasse `Fahrzeug` mit den Methoden, die für die Untersuchungsgegenstände U₁ und U₂ erforderlich sind. [15 Punkte]

b) Programmieren Sie die Unterklasse `PKWKombiX` für ein Fahrzeug mit den folgenden Eigenschaften:

- Typ "PKW-Kombi Fabrikat X",
- Höchstgeschwindigkeit 172,
- Bezinverbauchsfunktion $b(v) = 6 + \frac{(v - 91)^2}{1920} + 0,024 \cdot v$

[5 Punkte]

Lösung 1

```
a) { int[] a = new int[5];    a[0] = 0;    return a;
    }

b) { boolean b = 0 <= a[0] && a[0] < a.length;
    for (int i = 1; i < a[0] && b; i++)
        b = b && a[i] <= a[i + 1];
    return b;
    }

c) { int i, j;
    for (i = 1; i <= a[0] && a[i] < e; i++);
    for (j = a[0]; j >= i; j--) a[j + 1] = a[j];
    a[i] = e; a[0]++;
    return a;
    }

d) { for (int i = 1; i <= a[0]; i++)
    if (a[i] == e) return i;
    return 0;
    }

d) { return suche(a, e, 1, a[0]);    // Schnellere Lösung.
    }

int suche(int[] a, int e, int uG, int oG)
{ if (uG >= oG)
    if (uG > oG || a[uG] != e) return 0;
    else return uG;
    int Mitte = (uG + oG)/2;
    if (a[Mitte] < e) return suche(a, e, Mitte + 1, oG);
    else return suche(a, e, uG, Mitte);
}

e) { return sbw(a, 1, a[0]); }

Knoten sbw(int[] a, int uG, int oG)
{ if (uG > oG) return null;
  int Mitte = (uG + oG)/2;
  Knoten k = new Knoten(a[Mitte]);
  k.links = sbw(a, uG, Mitte - 1);
  k.rechts = sbw(a, Mitte + 1, oG);
  return k;
}
```

f)

	Sortiertes Array	Sortierte Liste	Suchbaum
Einsortieren	<input type="checkbox"/> schnell <input checked="" type="checkbox"/> langsam	<input type="checkbox"/> schnell <input checked="" type="checkbox"/> langsam	<input checked="" type="checkbox"/> schnell <input type="checkbox"/> langsam
Suchen	<input checked="" type="checkbox"/> schnell <input type="checkbox"/> langsam	<input type="checkbox"/> schnell <input checked="" type="checkbox"/> langsam	<input checked="" type="checkbox"/> schnell <input type="checkbox"/> langsam
Löschen	<input type="checkbox"/> schnell <input checked="" type="checkbox"/> langsam	<input type="checkbox"/> schnell <input checked="" type="checkbox"/> langsam	<input checked="" type="checkbox"/> schnell <input type="checkbox"/> langsam

Lösung 2

```

a) { for (Knoten z = Kopf; z != null; z = z.Nf)
    if (exKante(k, z) && exKante(z, e.Kante)) return true;
    return false;
}

boolean exKante (Knoten a, Knoten b)
{ for (Kante e = a.Kopf; e != null; e = e.Nf)
    if (e.Kante == b) return true;
    return false;
}

```

a) Alternative Lösung mit geringerem Aufwand:

```

{ for (Kante x = k.Kopf; x != null; x = x.Nf)
    if (x != e)
        for (Kante y = x.Kante.Kopf; y != null; y = y.Nf)
            if (y.Kante == e.Kante) return true ;
    return false;
}

```

```

b) class Kante
{ String Bez; Kante Nf; Knoten Kante;
  boolean Abkuerzung;
}

void loescheAbk()
{ Knoten k; Kante v, e;
  for (k = Kopf; k != null; k = k.Nf)
    for (e = k.Kopf; e != null; e = e.Nf)
      e.Abkuerzung = Abk(k, e);
  for (k = Kopf; k != null; k = k.Nf)
  { v = null;
    for (e = k.Kopf; e != null; e = e.Nf)
    { if (e.Abkuerzung)
      { if (k.Kopf == e) k.Kopf = e.Nf;
        if (k.Fuss == e) k.Fuss = v;
        if (v != null) v.Nf = e.Nf;
      }
      else v = e;
    }
  }
}

```

Es müssen alle Kanten vorhanden sein, um festzustellen, welche Kanten Abkürzungen sind. Erst danach dürfen Kanten gelöscht werden.

Lösung 3

- a) Nicht linear, weil es in f dynamisch zwei Aufrufe von f gibt: einen in der Bedingung der if-Abfrage und einen im if- bzw. else-Zweig.
- b) Nicht schlicht, weil die Rekursion nicht linear ist.
- c) Ausgabe:
- ```

34
17
8
7
14
7
-6
28

```

| Erläuterung dazu (nicht verlangt): | Ausgabe |
|------------------------------------|---------|
| f(34)                              | 34      |
| f(17)                              | 17      |
| f(8)                               | 8       |
| liefert 8                          |         |
| if-Zweig                           |         |
| f(7)                               | 7       |
| liefert 7                          |         |
| liefert 7                          |         |
| else-Zweig                         |         |
| f(14)                              | 14      |
| f(7)                               | 7       |
| liefert 7                          |         |
| else-Zweig                         |         |
| f(-6)                              | -6      |
| liefert -6                         |         |
| liefert -6                         |         |
| liefert -6                         |         |
| -6 + 34                            | 28      |

## Lösung 4

a) class Fahrzeug

```
{ Fahrzeug Nf;
 float minb;

 int U1()
 { minb = b(1); int minv = 1;
 for (int i = 2; i <= vmax(); i++)
 if (b(i) < minb)
 { minv = i; minb = b(i);
 }
 return minv;
 }

 int U2()
 { int min10v = 1; U1();
 for (int i = 2; i <= vmax(); i++)
 if (b(i) <= 1.1 * minb) min10v = i;
 return min10v;
 }

 String Typ() { return ""; }
 int vmax() { return 0; }
 float b(int v) { return 0.0f; }
}
```

b) class PKWKombiX extends Fahrzeug

```
{ String Typ() { return "PKW-Kombi Fabrikat X"; }
 int vmax() { return 173; }

 float b(int v)
 { return 6 + (v - 91) * (v - 91) / 1920.0f + 0.024f * v;
 }
}
```