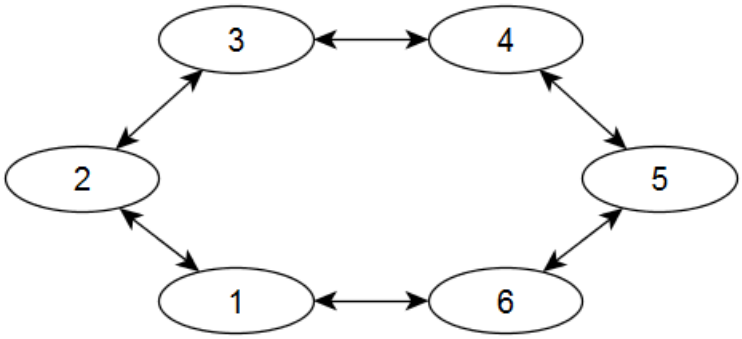
1 Datenstruktur (10 Punkte)

Ein Ring ist eine doppelt verkettete Liste, bei der das erste Element mit dem letzten Element verbunden ist. Beispiel:



Die Implementierung des Ringes soll allein über eine Klasse `RingElement` realisiert werden. Aufgrund der Ringstruktur bedarf es keines ausgezeichneten Start-Elementes. Anders als in der aus der Vorlesung bekannten Implementierung der verketteten Liste wird hier keine zu `VListe` analoge Klasse benötigt.

a) Schreiben Sie die generische Klasse `RingElement`, das in der oben gezeigten Datenstruktur verwendet werden kann. Das `RingElement` soll einen Vorgänger und einen Nachfolger als Attribut besitzen, sodass Vorwärts- und Rückwärts-Navigation möglich ist. Sehen Sie außerdem ein Nutzdaten-Attribut vor.

b) Implementieren Sie in der Klasse die Methode `insertRingElement`, die ein übergebenes `RingElement`-Objekt als Nachfolger des aktuellen `RingElement`s einfügt und stellen Sie sicher, dass dabei die Ringstruktur erhalten bleibt. Denken Sie daran, dass der Ring zu Beginn aus nur einem Element besteht!

c) Implementieren Sie in der Klasse die Methode `getRingElements`, die eine Liste der in der Ringstruktur erhaltenen `RingElement`-Objekte zurückliefert. Wählen Sie einen geeigneten Rückgabetyp.

2 Rekursion (10 Punkte)

Gegeben sei folgender Quellcode:

```
public static void main(String[] args) {
	int[] a = { 1, 2, 3, 4, 5 };
	doit(a, 0);
}

public static void doit(int[] a, int i) {
	if (i >= a.length / 2) {
		return;
	}
	int tmp = a[i];
	a[i] = a[a.length - 1 - i];
	a[a.length - 1 - i] = tmp;
	doit(a, i + 1);
}
```

a) Wie sieht das Array `a` aus, nachdem das Programm ausgeführt wurde? Zeichnen Sie alle rekursiven Aufrufe auf!
b) Geben Sie die Komplexität „Groß Oh“ des Algorithmus an! Begründen Sie!
c) Schreiben Sie eine äquivalente iterative Methode zu `doit`!

3 Comparator (14 Punkte)

Sie haben für ein Astronomie-Labor eine Webanwendung mitentwickelt. Die dort arbeitenden Wissenschaftler möchten in der Planeten-Übersicht eine neue Sortierung auswählen können. Sie wurden beauftragt, die Wünsche der Wissenschaftler umzusetzen. Sie wünschen sich, dass die Planeten aufsteigend nach der Bezeichnung ihres Sonnensystems sortiert werden. Innerhalb dieser Sortierung soll nach Masse des Planeten sortiert werden. Ob aufsteigend oder absteigend nach der Masse sortiert wird, soll dynamisch (d.h. zur Laufzeit) festgelegt werden können.
Folgende Klassen sind gegeben – Getter und Setter dürfen Sie als gegeben voraussetzen. Außerdem dürfen Sie davon ausgehen, dass keines der Attribute `null` ist:

```
public class AstronomischesObjekt {
        protected String bezeichnung;
        protected double masse;

        public AstronomischesObjekt(String bezeichnung, double masse) {
                this.bezeichnung = bezeichnung;
                this.masse = masse;
        }
}
```

```
public class Galaxie extends AstronomischesObjekt {
        private List<Sonnensystem> sonnensysteme;

        public Galaxie(String bezeichnung, double masse) {
                super(bezeichnung, masse);
        }
}
```

```
public class Sonnensystem extends AstronomischesObjekt {
        private Galaxie galaxie;
        private List<Planet> planeten;

        public Sonnensystem(String bezeichnung, double masse, Galaxie galaxie) {
                super(bezeichnung, masse);
                this.galaxie = galaxie;
        }
}
```

```
public class Planet extends AstronomischesObjekt {
        private Sonnensystem sonnensystem;

        public Planet(String bezeichnung, double masse, Sonnensystem sonnensystem) {
                super(bezeichnung, masse);
                this.sonnensystem = sonnensystem;
        }
}
```

a) Schreiben Sie einen Comparator, der die Wünsche der Wissenschaftler erfüllt!
Auszug aus der Java-Doc zum Comparator:

compare

```
int compare(T o1,
            T o2)
```

Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

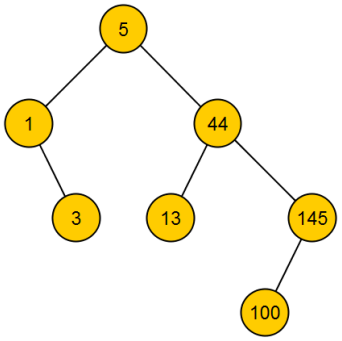
b) Angenommen, Sie finden im Quellcode der Webanwendung folgenden Comparator:

```
class MyComparator<T extends AstronomischesObjekt> implements
                Comparator<T> { }
```

Geben Sie die Namen aller oben definierten Klassen an, die dieser Comparator vergleichen kann!

4 AVL-Baum (6 Punkte)

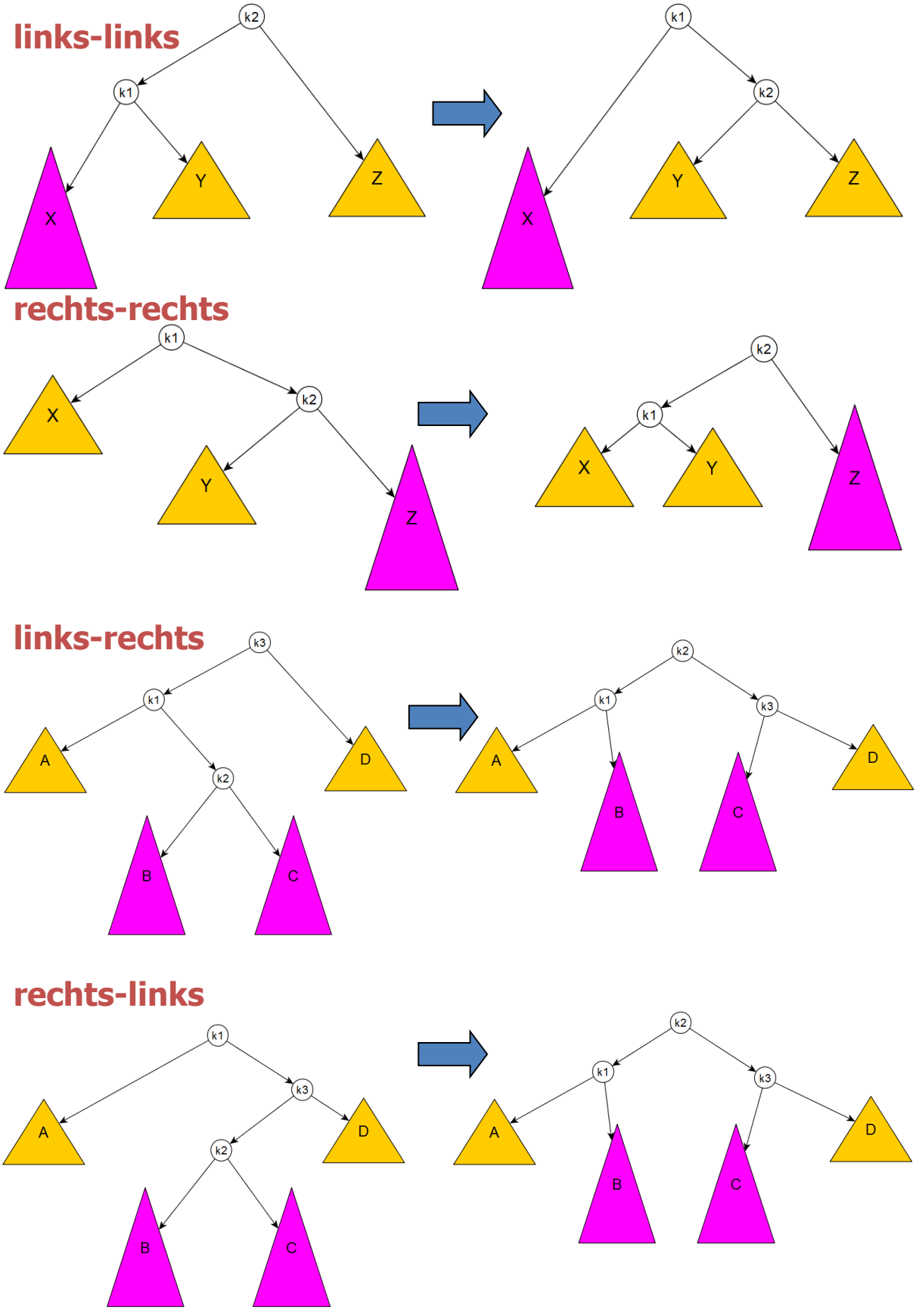
Gegeben sei folgender AVL-Baum:



Löschen Sie die Elemente: 3, 44 (in genau dieser Reihenfolge).
Fügen Sie dann in den resultierenden Baum folgende Elemente ein: 99, 500, 600 (in genau dieser Reihenfolge).
Stellen Sie nach jedem Schritt sicher, dass die AVL-Eigenschaft erhalten bleibt.
Zeichnen Sie nach jedem Löschen bzw. Einfügen den Zustand des AVL-Baums!
Hinweis: Ersetzen Sie den gelöschten Knoten entsprechend dem einfachen binären Suchbaum durch das Maximum aus dem linken Teilbaum!

Auf der nächsten Seite finden Sie die allgemeinen Fälle für die Rotationen.

Allgemeine Rotationen AVL-Baum



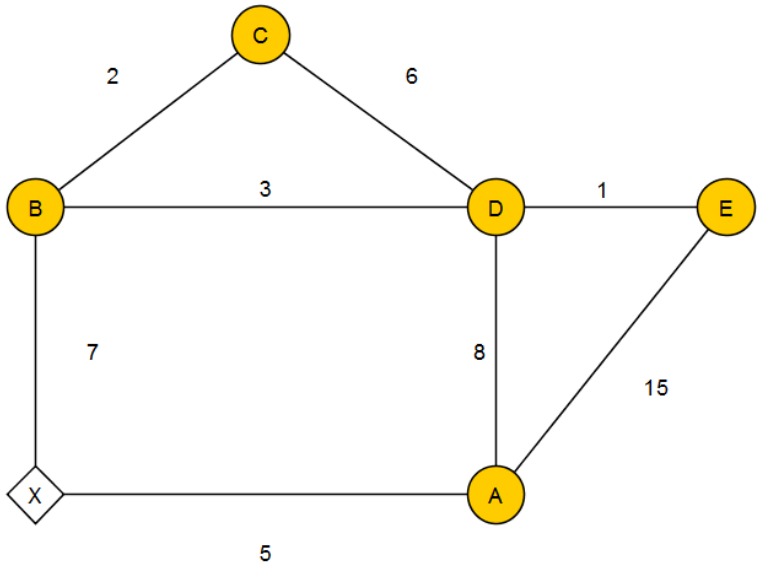
5 Wahr oder Falsch (10 Punkte)

Beantworten Sie die nachfolgenden Behauptungen. Für eine korrekte Antwort gibt es +1 Punkt, für „weiß nicht“ 0 Punkte, für eine falsche Antwort -1 Punkt (jedoch in Summe nicht weniger als 0 für diese Aufgabe). Kreuzen Sie Ihre Antwort in der entsprechenden Zelle an.

Behauptung	**Wahr**	**Falsch**	**Weiß nicht**
Jedes Java-Objekt besitzt die Methode `compareTo`			
Das Suchen eines Objektes in einer unsortierten Liste hat die Komplexität O(n) = n			
Ein binärer Suchbaum kann nicht zur Sortierung von Objekten verwendet werden			
Mit dem downHeap wird erreicht, dass ein Binary Heap weniger Speicherplatz benötigt			
Jedes Java-Objekt besitzt die Methode `equals(Object o)`			
Der Dijkstra-Algorithmus lässt sich auf gerichteten Graphen anwenden			
Eine HashMap besteht aus Schlüssel-Wertpaaren			
Das `Iterator`-Interface definiert die Methode `insert(Object o)`			
Generics dienen der Typsicherheit			
Die ForEach-Schleife kann auf allen Objekten angwandt werden, die das `Iterable`-Interface implementieren			

6 Wegfindung (8 Punkte)

Sie steuern ein Raumschiff in einem Computerspiel. Ihr Raumschiff (mit X gekennzeichnet) soll von seiner gegenwärtigen Position zum Ort E gelangen. Dabei möchten Sie möglichst wenig Energie verbrauchen. Die Zahl an der Kante gibt den Energieverbrauch für diesen Weg an.



Nutzen Sie einen aus der Vorlesung bekannten Algorithmus, um Ihr Raumschiff mit möglichst wenig Energie nach E zu bringen. Zeigen Sie in einer Tabelle, wie Sie den Weg gefunden haben. Wie lautet ein optimaler Weg nach E?