Aufgabe 33: (2+2+2+2=8 Punkte) Betrachten Sie ein Feld $\mathtt{a}[0,\ldots,n-1]$, das n paarweise verschiedene Zahlen in unsortierter Folge speichert. Gesucht ist die größte im Feld $\mathtt{a}$ gespeicherte Zahl.

(a) Entwerfen Sie einen Algorithmus für dieses Problem, der nach dem *Divide-and-Conquer*-Prinzip arbeitet. Erstellen Sie hierzu in Pseudocode eine rekursiv definierte Funktion MAXIMUM, die neben einer Referenz auf das zu untersuchende Feld $\mathtt{a}$ zwei Indizes b und e entgegen nimmt und die größte Zahl zurückliefert, die im Feld $\mathtt{a}[b,\ldots,e-1]$ gespeichert ist.

(b) Beweisen Sie induktiv, dass Ihr Algorithmus korrekt arbeitet.

(c) Bestimmen Sie die asymptotische Anzahl der von diesem Algorithmus durchgeführten Vergleiche zwischen Elementen der Eingabe.

(d) Implementieren Sie den Algorithmus unter Verwendung der im Learnweb bereitgestellten Klassen.

Hinweise:

- Zur Vereinfachung dürfen Sie für die Aufgabenteile (a)–(c) annehmen, dass n eine ganzzahlige Potenz der Zahl 2 ist.
- Aus Kapitel 3 sollte bekannt sein, dass für dieses Problem eine untere Schranke von $\Omega(n)$ gilt. Ihre Lösung kann also aymptotisch nicht besser als die lineare Suche sein. In dieser Aufgabe geht es daher primär darum, den Umgang mit dem *Divide-and-Conquer*-Paradigma zu üben.

Lösung:

(a)

```
1: function MAXIMUM(Feld a, int b, int e)
2:     if (e − b = 1) then
3:         return a[b];                      ▷ Basisfall: Nur ein Element.
4:     m := (b + e)/2;                       ▷ Bestimme Aufteilungspunkt, ganzzahlige Division.
5:     ℓ := MAXIMUM(a, b, m);                ▷ Rekursive Betrachtung der ersten Hälfte.
6:     r := MAXIMUM(a, m, e);                ▷ Rekursive Betrachtung der zweiten Hälfte.
7:     if (ℓ > r) then                       ▷ Bestimme besseres Teilergebnis.
8:         return ℓ;
9:     return r;
```

(b) Wir beweisen die Korrektheit induktiv über $n := e - b$.

Induktionsanfang $(n = 1)$**:** Wenn das zu untersuchende Feld aus nur einem Element $\mathtt{a}[b]$ besteht, ist dieses automatisch ein maximales Element. In Zeile 2 wird die Bedingung $e - b = 1$ geprüft; im Erfolgsfall wird wie gefordert $\mathtt{a}[b]$ zurückgegeben.

Induktionsvoraussetzung: Der Algorithmus arbeite korrekt auf allen Eingaben, für die $e - b < n$ gilt.

Induktionsschritt $(n - 1 \to n)$**:** Wir betrachten eine beliebige Eingabe, für die $e - b = n$ gilt. Auf Grund des obigen Induktionsanfangs dürfen wir annehmen, dass zudem $n \geq 2$ gilt. Das Problem wird nun im Algorithmus auf zwei Teilprobleme der Größe $\lfloor (e-b)/2 \rfloor$ und $\lceil (e-b)/2 \rceil$ zurückgeführt, da gilt:

$$\left\lfloor \frac{b+e}{2} \right\rfloor \;=\; \left\lfloor b + \frac{e-b}{2} \right\rfloor \;=\; b + \left\lfloor \frac{e-b}{2} \right\rfloor .$$

Da nach Annahme $2 \leq n = (e-b)$ gilt, ist insbesondere $\lfloor (e-b)/2 \rfloor \leq \lceil (e-b)/2 \rceil = \lceil n/2 \rceil < n$. Somit kann für die rekursiven Aufrufe induktiv angenommen werden, dass sie jeweils korrekt ein maximales Element bestimmen, d.h., dass gilt:

$$\ell \;=\; \max\{\mathtt{a}[i] \mid b \leq i < m\} \quad \text{und} \quad r \;=\; \max\{\mathtt{a}[i] \mid m \leq i < e\}$$

Für $x := \max\{\ell, r\}$ gilt somit

$$x \;=\; \max\{\max\{\mathtt{a}[i] \mid b \leq i < m\}, \max\{\mathtt{a}[i] \mid m \leq i < e\}\} \;=\; \max\{\mathtt{a}[i] \mid b \leq i < e\}.$$

Dies aber ist genau der gesuchte Wert.

Durch Induktion über $\mathbb{N}$ folgt die Behauptung.

(c) Aus dem obigen Pseudocode ist zu erkennen, dass in jedem rekursiven Aufruf ein Elementvergleich durchgeführt wird. Da die Rekursion auf zwei Eingabe mit maximal je $n/2$ Elementen angewandt wird, erhalten wir also folgende Rekursionsgleichung für die Anzahl der Elementvergleiche:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + c.$$

Gemäß des Mastertheorems ($a = 2, b = 2, f(n) = c, c > 0$) betrachten wir nun $\log_b a = \log_2 2 = 1$. Mit (z.B.) $\varepsilon = 1$ gilt $f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$, also Fall 1 des Master-Theorems.

Somit folgt, dass der betrachtete Algorithmus $\Theta(n^1) = \Theta(n)$ Elementvergleiche durchführt. (Diese Aussage lässt sich im Übrigen auch für die Laufzeit herleiten, wenn ein Vergleich in $\mathcal{O}(1)$ durchgeführt werden kann, hiernach war aber nicht gefragt.)

(d) Die Implementierung gestaltet sich wie folgt:

```
/**
 *  L&ouml;sung zu Aufgabe 33 (Sommersemester 2015)
 */
public class Aufgabe33 {

    /**
     * Finde ein maximales Element im &uuml;bergebenen Feld.
     * @param a Feld mit zu untersuchenden Werten.
     * @return Element mit maximalem Wert.
     */
    public static <T extends Comparable<T>> T maximum(T[] a) {
        if (a==null) {
            throw new IllegalArgumentException("Uebergebene Referenz ist null.");
        }
        if (a.length == 0) {
            throw new IllegalArgumentException("Uebergebenes Feld hat die Laenge Null.")
        }
        return maximumIntern(a, 0, a.length);
    }

    /**
     * Finde ein maximales Element in einem Teilfeld des &uuml;bergebenen Felds.
     * @param a Feld mit zu untersuchenden Werten.
     * @param b Index des ersten Elements im zu untersuchenden Feld (inklusive).
     * @param e Index des letzten Elements im zu untersuchenden Feld (exklusive).
     * @return Element mit maximalem Wert im untersuchten Teilfeld.
     */
    protected static <T extends Comparable<T>> T maximumIntern(T[] a, int b, int e) {

        // Besteht das Feld nur aus einem Element, ist dieses
        // automatisch das maximale Element. Der Fall, dass
        // e=b gilt, wird in der aufrufenden Methode ausgeschlossen.
        if (e-b == 1) {
            return a[b];
        }

        // Bestimme den Index, an dem das Feld fuer die
        // rekursiven Aufrufe aufgeteilt wird.
        int m = b + (e - b) / 2;

        // Bestimme ein maximales Element im linken Teilfeld.
        T maxLinks  = maximumIntern(a, b, m);

        // Bestimme ein maximales Element im rechten Teilfeld.
        T maxRechts = maximumIntern(a, m, e);

        // Vergleiche die beiden soeben gefundenen Elemente
        // und gibt das Element zurueck, das nicht kleiner
        // ist als das andere Element.
        if (maxLinks.compareTo(maxRechts) > 0) {
            return maxLinks;
        }
```

```java
        return maxRechts;
    }
}
```

```java
import static org.junit.Assert.*;

import org.junit.Test;


public class Aufgabe33Test {

    @Test
    public void testMaximumIdentisch() {

        Integer[] daten = { 3, 3, 3, 3, 3, 3, 3 };
        Integer expected = 3;

        assertEquals("Maximum-Suche schlaegt bei identischen Elementen fehl.",
                expected,
                Aufgabe33.maximum(daten));
    }

    @Test
    public void testMaximumAufsteigendSortiert() {

        Integer[] daten = { 1, 2, 3, 4, 5 };
        Integer expected = 5;

        assertEquals("Maximum-Suche schlaegt bei aufsteigend sortierten Elementen fehl."
                expected,
                Aufgabe33.maximum(daten));
    }

    @Test
    public void testMaximumAbsteigendSortiert() {

        Integer[] daten = { 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3 };
        Integer expected = 9;

        assertEquals("Maximum-Suche schlaegt bei absteigend sortierten Elementen fehl.",
                expected,
                Aufgabe33.maximum(daten));
    }

    @Test
    public void testMaximumNegativeZahlen() {

        Integer[] daten = { -4, -5, -1, -2, -3 };
        Integer expected = -1;

        assertEquals("Maximum-Suche schlaegt bei negativen Elementen fehl.",
                expected,
                Aufgabe33.maximum(daten));
    }

    @Test
    public void testMaximumEinElement() {

        Integer[] daten = { 42 };
        Integer expected = 42;

        assertEquals("Maximum-Suche schlaegt bei einem Elementen fehl.",
                expected,
                Aufgabe33.maximum(daten));
    }

    @Test
    public void testMaximumUngueltigeEingaben() {
```

```java
        Integer[] daten1 = null;

        boolean abgefangen = false;

        try {
            Aufgabe33.maximum(daten1);
        }
        catch (IllegalArgumentException e) {
            abgefangen = true;
        }
        assertTrue("Maximum-Suche reagiert nicht mit Ausnahme auf null-Parameter.",
                abgefangen);

        Integer[] daten2 = { };
        abgefangen = false;

        try {
            Aufgabe33.maximum(daten2);
        }
        catch (IllegalArgumentException e) {
            abgefangen = true;
        }
        assertTrue("Maximum-Suche reagiert nicht mit Ausnahme auf leeres Feld.",
                abgefangen);

    }

}
```

Bewertungsschema:

(a) Für diese Teilaufgabe werden zwei Punkte vergeben.

(b) Für diese Teilaufgabe werden zwei Punkte vergeben.

- Es wird ein Punkt für die Formulierung der zu beweisenden Aussage und den Induktionsanfang vergeben.
- Es wird ein Punkt für den Induktionsschritt vergeben.

(c) Für diese Teilaufgabe werden zwei Punkte vergeben.

- Es wird ein Punkt für das korrekte Aufstellen der Rekursionsgleichung vergeben.
- Es wird ein Punkt für die korrekte Auflösung der Rekursionsgleichung vergeben.

(d) Für diese Teilaufgabe werden zwei Punkte vergeben.

- Es wird ein Punkt für die Implementation des Algorithmus vergeben.
- Es wird ein Punkt für die Kommentierung des Algorithmus vergeben.