Algorithmen und Datenstrukturen 1

Serie 4

1. a) Analyse des Quicksort-Verfahrens:

7	12	14	2	13	8	5	4	57	5
5	4	5	2	13	8	14	12	57	7
2	4	5	5	13	8	14	12	57	7
2	4	5	5	13	8	14	12	57	7
2	4	5	5	7	8	12	14	57	13
2	4	5	5	7	8	12	14	57	13
2	4	5	5	7	8	12	14	57	13
2	4	5	5	7	8	12	13	57	14
2	4	5	5	7	8	12	13	14	57

4	6	3	14	15	3	7	9	12	1	6	2
2	1	3	3	15	14	7	9	12	6	6	4
1	2	3	3	15	14	7	9	12	6	6	4
1	2	3	3	15	14	7	9	12	6	6	4
1	2	3	3	15	14	7	9	12	6	6	4
1	2	3	3	4	14	7	9	12	6	6	15
1	2	3	3	4	14	7	9	12	6	6	15
1	2	3	3	4	6	7	9	12	6	14	15
1	2	3	3	4	6	7	9	12	6	14	15
1	2	3	3	4	6	6	9	12	7	14	15
1	2	3	3	4	6	6	7	12	9	14	15
1	2	3	3	4	6	6	7	9	12	14	15

   b) Worst-Case: vorsortierte Liste
   Best-Case: das mittlere Element einer Teilliste steht links

2. Klassifikation von Bäumen:

	a)	b)	c)	d)	e)	f)	g)
strikter Binärbaum				X			X
ausgeglichener Binärbaum		X		X		X	X
vollständiger Binärbaum							X
Heap	X			X			X
Binärbaum	X	X		X		X	X
binärer Suchbaum							
Höhe des Baumes/Graphen	5	4	3	4	4	4	2
Grad des Baumes/Graphen	1	2	2	2	2	2	2

3. a) Hier der Quellcode der Aufgabe:

```
/**
 * Klasse die einen bin"arsuchbaum implementiert
 * geschrieben im Rahmen von ADS1, Serie 4, Aufgabe 3
 *
 * Date: Jan 5, 2003
 * Time: 8:19:49 PM
 * @author Arne Brutschy
 */
public class BinaryDic {

    // inline Klasse die einen einzelnen Knoten im Baum repr"asentiert
    private class IntTreeNode {
        IntTreeNode parent;
        IntTreeNode left;
        IntTreeNode right;
        Integer value;

        public IntTreeNode(IntTreeNode parent, Integer value) {
            this.parent = parent;
            this.value = value;
        }
    }

    private IntTreeNode root;
    private int         elements;


    /**
     * Erzeugt einen bin"aren Suchbaum mit leerer Wurzel
     */
    public BinaryDic() {
        root = null;
        elements = 0;
    }

    /**
     * Erzeugt einen bin"aren Baum mit Inhalt
     * Die Elemente werden als integer Array uebergeben und
     * werden in Reihenfolge eingefuegt
     * @param intArray Elemente des Baumes
     */
    public BinaryDic(int[] intArray) {
        root = null;
        elements = 0;

        for (int i = 0; i < intArray.length; i++)
            insert(new Integer(intArray[i]));
```

```
}

/**
 * Fuegt ein Integer-Objekt in den Baum ein
 * @param x Integer Objekt das eingefuegt werden soll
 */
public void insert(Integer x) {
    // gibts es schon eine Wurzel?
    if (root != null) {
        int res = 0;
        IntTreeNode last = null;
        IntTreeNode current = root;

        // wir suchen bis wir an einem Blatt sind
        while (current != null) {
            last = current;
            // vergleiche mit Wert in momentanen Knoten
            res = x.compareTo(current.value);
            // kleiner -> linker Sohn
            if (res < 0)
                current = current.left;
            // groesser -> rechter Sohn
            else if (res > 0)
                current = current.right;
            // ex. schon -> raus
            else
                return;
        }

        // Wert an den richtigen Sohn anfuegen
        if (res < 0)
            last.left = new IntTreeNode(last, x);
        else
            last.right = new IntTreeNode(last, x);

    // Wurzel existiert nicht, wird neu angelegt
    } else
        root = new IntTreeNode(null, x);

    elements++;
}

/**
 * Sucht ein Integer Objekt im Baum
 * @param x zu suchendes Integer Objekt
 * @return gesuchtes Objekt wenn gefunden, null wenn nicht gefunden
 */
public Integer search(Integer x) {
    int res = 0;
```

```
        IntTreeNode current = root;
        StringBuffer out = new StringBuffer();

        // wir suchen bis wir den Wert gefunden haben oder an einem Blatt sind
        while (current != null) {
            out.append(fixedLengthString(current.value.intValue()));
            // vergleiche mit Wert in momentanen Knoten
            res = x.compareTo(current.value);
            // kleiner -> linker Sohn
            if (res < 0)
                current = current.left;
            // groesser -> rechter Sohn
            else if (res > 0)
                current = current.right;
            // gefunden -> raus
            else {
                System.out.println(out);
                return x;
            }
        }

        System.out.println("Nicht gefunden!");
        return null;
    }

    /**
     * Fuehrt eine Breitensuche aus und gibt die erhaltenden Knoten mit ihren
     * linken und rechten S"ohnen als String zuruck
     * @return Auflistung Knoten Breitensuche
     */
    public String toString() {
        int rpos = 0;           // leseposition im Stack
        int wpos = 0;           // schreibposition im Stack

        // der Stack in dem die zu bearbeitenden Knoten gespeichert werden
        IntTreeNode stack[] = new IntTreeNode[elements];
        StringBuffer out = new StringBuffer("Knoten Lsohn Rsohn\n");

        // wir starten mit der Wurzel
        stack[rpos] = root;

        while (rpos < elements) {
            // momentanen Knoten ausgeben
            out.append(fixedLengthString(stack[rpos].value.intValue()));

            // linker Sohn in den Stack (wenn ex) und Ausgabe erzeugen
            if (stack[rpos].left != null) {
                stack[++wpos] = stack[rpos].left;
                out.append(fixedLengthString(stack[rpos].left.value.intValue()));
```

```
                // linker Sohn ex. nicht, fuege Spaces ein
                } else
                    out.append("      ");
                // rechter Sohn in den Stack und ausgabe
                if (stack[rpos].right != null) {
                    stack[++wpos] = stack[rpos].right;
                    out.append(fixedLengthString(stack[rpos].right.value.intValue()));
                }

                // naechsten Knoten lesen
                rpos++;
                out.append("\n");
            }

            return out.toString();
        }

        // kleine Hilfsfunktion die ein int auf einen String
        // gleichbleibender Laenge formatiert
        private static String fixedLengthString(int i) {
            String out ="      "+i;
            return out.substring(out.length() - 6);
        }

        // Hauptfunktion
        public static void main(String argv[]) {
            int vars[] = {74, 27, 60, 26, 12, 2, 50, 25, 75, 80, 81, 79, 15};
            BinaryDic btree = new BinaryDic(vars);

            System.out.println("Breitensuche:");
            System.out.println(btree.toString());

            System.out.println("Suche nach 15:");
            btree.search(new Integer(15));

            System.out.println("Suche nach 27:");
            btree.search(new Integer(27));

            System.out.println("Suche nach 75:");
            btree.search(new Integer(75));

            System.out.println("Suche nach 100:");
            btree.search(new Integer(100));
        }
    }
```

b) Hier die Ausgabe des Programmes:

```
Breitensuche:
Knoten Lsohn Rsohn
    74    27    75
    27    26    60
    75          80
    26    12
    60    50
    80    79    81
    12     2    25
    50
    79
    81
     2
    25    15
    15

Suche nach 15:
    74    27    26    12    25    15
Suche nach 27:
    74    27
Suche nach 75:
    74    75
Suche nach 100:
Nicht gefunden!
```