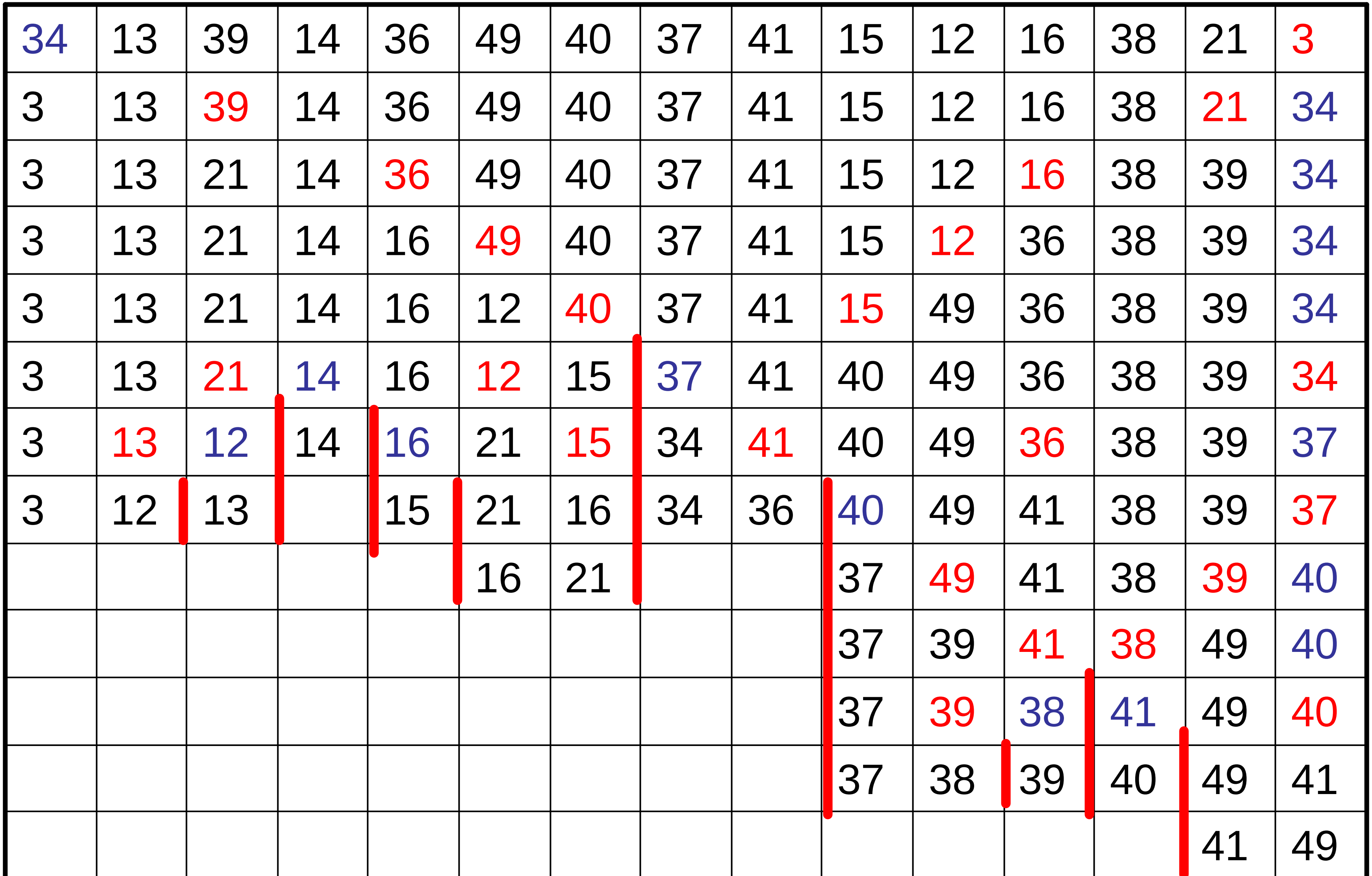
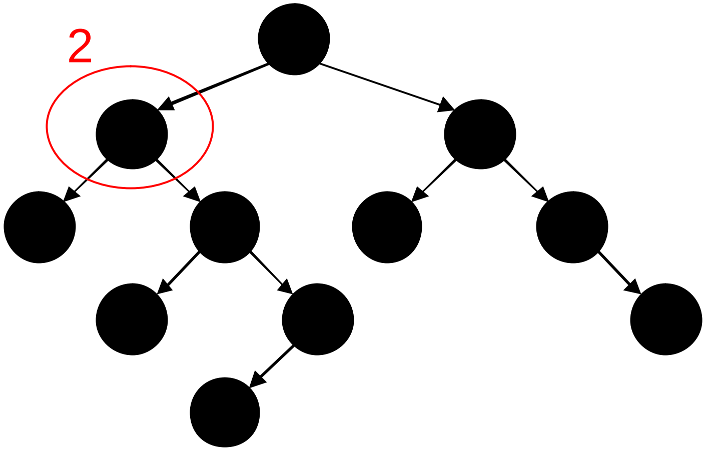
Quicksort



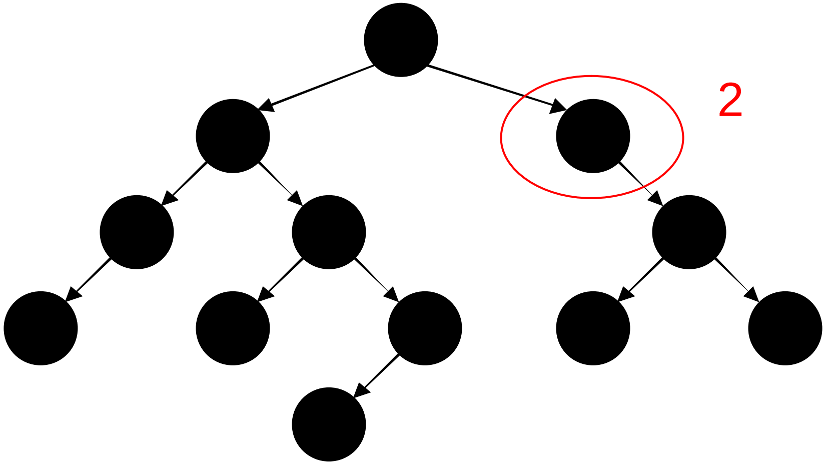
34	13	39	14	36	49	40	37	41	15	12	16	38	21	3
3	13	39	14	36	49	40	37	41	15	12	16	38	21	34
3	13	21	14	36	49	40	37	41	15	12	16	38	39	34
3	13	21	14	16	49	40	37	41	15	12	36	38	39	34
3	13	21	14	16	12	40	37	41	15	49	36	38	39	34
3	13	21	14	16	12	15	37	41	40	49	36	38	39	34
3	13	12	14	16	21	15	34	41	40	49	36	38	39	37
3	12	13		15	21	16	34	36	40	49	41	38	39	37
					16	21			37	49	41	38	39	40
									37	39	41	38	49	40
									37	39	38	41	49	40
									37	38	39	40	49	41
													41	49

AVL-Bäume

a)



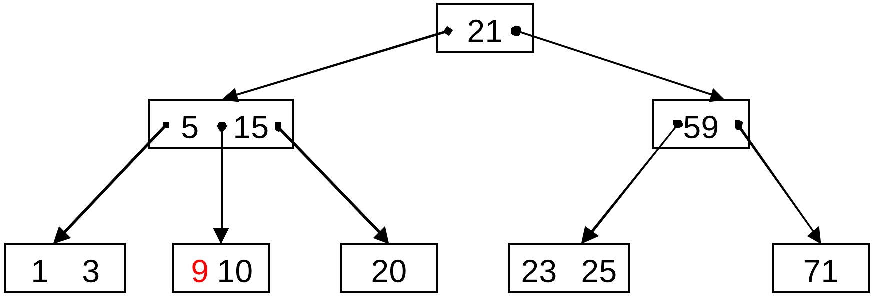
c)

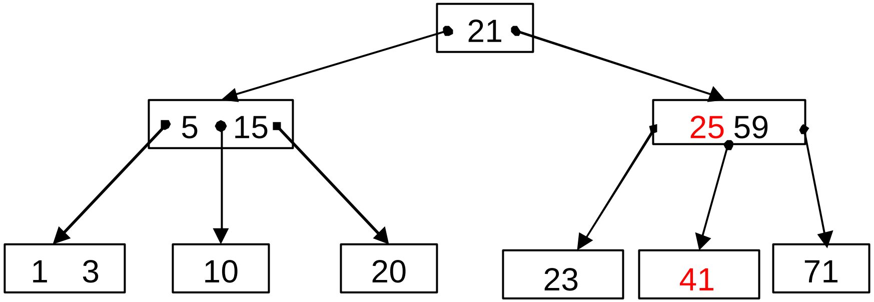


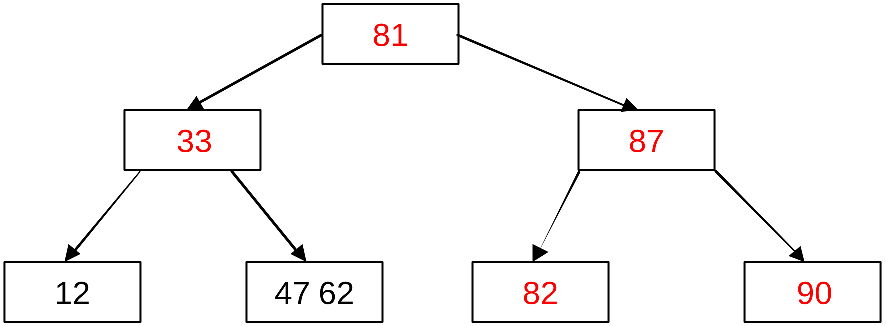
Dijkstra

v_i	B	C	D	E	F	B	C	D	E	F
A	--	35	4	--	--	A	A	A	A	A
D	--	34	4	6	8	A	D	A	D	D
E	18	32	4	6	7	E	E	A	D	E
F	17	31	4	6	7	F	F	A	D	E
B	17	22	4	6	7	F	B	A	D	E
C	17	22	4	6	7	F	B	A	D	E

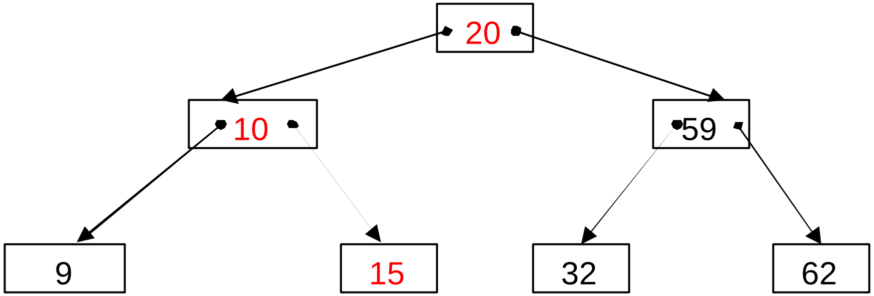
B-Bäume



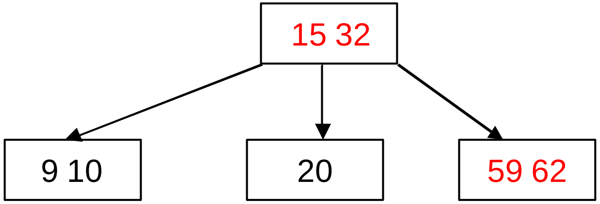


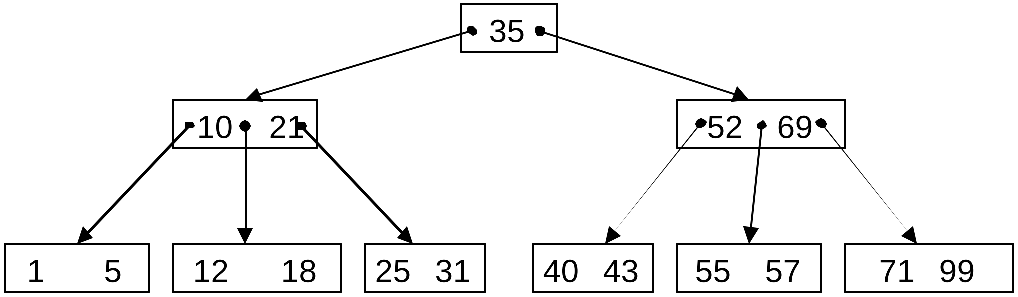


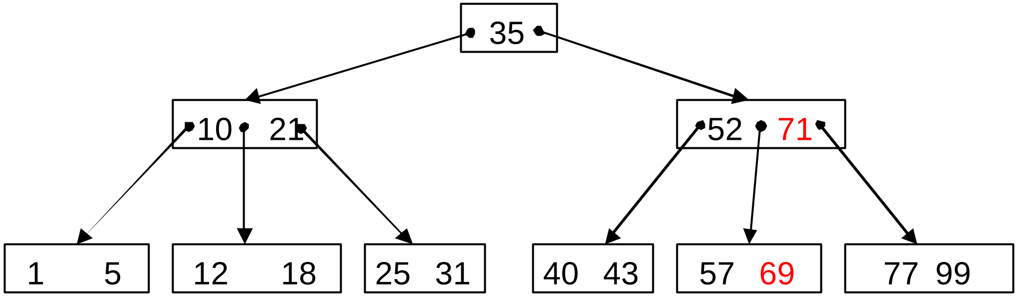
Variante links

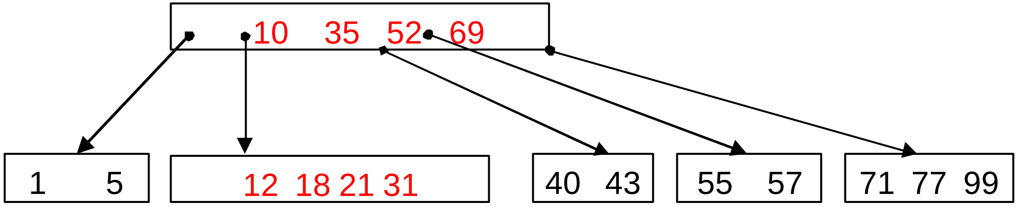


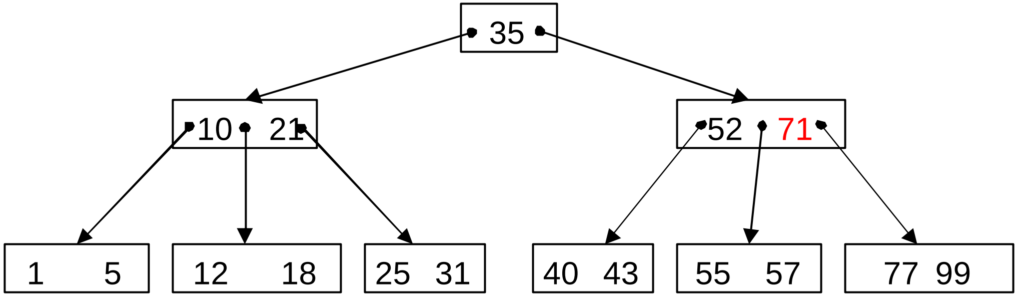
Variante rechts

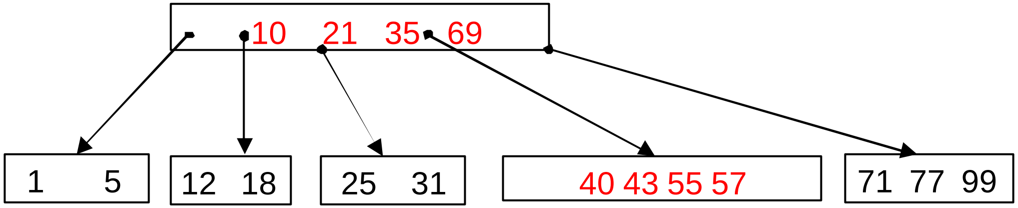



35
10 21
52 69
1 5
12 18
25 31
40 43
55 57
71 99


35
10 21
52 71
1 5
12 18
25 31
40 43
57 69
77 99


10 35 52 69
1 5
12 18 21 31
40 43
55 57
71 77 99


35
10 21
52 71
1 5
12 18
25 31
40 43
55 57
77 99


10 21 35 69
1 5
12 18
25 31
40 43 55 57
71 77 99

Rekursion

```
public class BinaryTreeA {
    private class Node {
        Object content;
        Node left, right;
        Node(Node original) {
            content = original.content;
            if (original.left != null)
                left = new Node(original.left);
            else left = null;
            if (original.right != null)
                right = new Node(original.right);
            else right = null;
        }
    }
    private Node root, current;
    public BinaryTreeA() {
        root = null;
        current = root;
    }
    public BinaryTreeA(BinaryTreeA original) {
        if (original.root != null)
            root = new Node(original.root);
        else root = null;
        current = root;
    }
}
```

```
public class BinaryTreeB {
        private class Node {
                Object content;
                Node left, right;
                Node(Object newContent) {
                        content = newContent;
                        left = null;
                        right = null;
                }
        }
        private Node root, current;
        public BinaryTreeB() {
                root = null;
                current = root;
        }
        private Node copyTree(Node original) {
                if (original == null)
                        return null;
                Node copy = new Node(original.content);
                copy.left = copyTree(original.left);
                copy.right = copyTree(original.right);
                return copy;
        }
        public BinaryTreeB(BinaryTreeB original) {
                root = copyTree(original.root);
                current = root;
        }
}
```

Boyer-Moore

```
public class Zahlenfolge {
    private String text;
    private int[] getSkipTabelle(String pat) {
        int tab[] = new int[10];
        int l = pat.length();
        for (int i = 0; i < tab.length; i++) {
            tab[i] = l + 1;
        }
        for (int i = 0; i < l; i++) {
            tab[pat.charAt(i) - '0'] = l - i;
        }
        return tab;
    }
    private boolean fits(String pat, int pos) {
        return pat.equals(text.substring(pos, pos + pat.length()));
    }
```

```
public int findFirst(String pat) {
        int textl = text.length();
        int patl = pat.length();
        int skip[] = getSkipTabelle(pat);
        int pos = 0;
        while (pos + patl < textl) {
                if (fits(pat, pos))
                        return pos;
                pos += skip[text.charAt(pos + patl) - '0'];
        }
        if (pos + patl == textl && fits(pat, pos))
                return pos;
        return -1;
}
```

Stack

```
public class Stack {
        private double stack[] = new double[10];
        private int top = 0;
        private void resize(int size) {
                double tmp[] = new double[size];
                System.arraycopy(stack, 0, tmp, 0, top);
                stack = tmp;
        }
        public void push(double value) {
                if (top == stack.length)
                        resize(2 * top);
                stack[top++] = value;
        }
        public double pop() throws EmptyStackException {
                if (top == 0)
                        throw new EmptyStackException();
                double result = stack[--top];
                if (3 * top < stack.length && top > 5)
                        resize(stack.length / 2);
                return result;
        }
}
```

Breiten- und Tiefensuche

A E B F C I F

B

H

D A

H

G D

Tiefensuche:

A E B F C I H D G

Breitensuche:

A E G B D H F C I

Heap

- 11, 15, 19, 7, 12, 13, 5, 3, 6, 1, 4 20
  19, 15, 13, 7, 12, 11, 5, 3, 6, 1, 4 20
- 4, 15, 13, 7, 12, 11, 5, 3, 6, 1 19, 20
  15, 12, 13, 7, 4, 11, 5, 3, 6, 1 19, 20
- 1, 12, 13, 7, 4, 11, 5, 3, 6 15, 19, 20
  13, 12, 11, 7, 4, 1, 5, 3, 6 15, 19, 20
- 6, 12, 11, 7, 4, 1, 5, 3 13, 15, 19, 20
  12, 7, 11, 6, 4, 1, 5, 3 13, 15, 19, 20
- 3, 7, 11, 6, 4, 1, 5 12, 13, 15, 19, 20
  11, 7, 5, 6, 4, 1, 3 12, 13, 15, 19, 20
- 11, 7, 5, 6, 4, 1, 3, 10
  11, 10, 5, 7, 4, 1, 3, 6