

VL11, Lösung 1

a)

Algorithmus	Laufzeit im worst case	Anwendungsgebiete
Lineare Suche	$O(n)$	Universell einsetzbar, einziges elementares Verfahren für unsortierte Folgen
Binäre Suche	$O(\log n)$	Nur für sortierte Folgen geeignet
Interpolationssuche	$O(n)$	Nur für sortierte Folgen geeignet, $O(\log \log n)$ im average case bei Gleichverteilung

b)

1.

1	2	3	4	5	6	7	8	9	10
2	5	11	15	20	32	50	160	300	456

Binäre Suche:

- Mitte bestimmen: $m = l + \frac{r-l}{2} = 1 + \frac{10-1}{2} = 1 + \frac{9}{2} = 5$
- $z[5] = 20$, Fortsetzen der Suche bei den Elementen 6 bis 10
- Neue Mitte bestimmen: $m = l + \frac{r-l}{2} = 6 + \frac{10-6}{2} = 6 + \frac{4}{2} = 8$
- $z[8] = 160$, Fortsetzen der Suche bei den Elementen 6 bis 7
- Neue Mitte bestimmen: $m = l + \frac{r-l}{2} = 6 + \frac{7-6}{2} = 6 + \frac{1}{2} = 6$
- $z[6] = 32$, die Zahl wurde nach drei Schritten gefunden

2.

1	2	3	4	5	6	7	8	9	10
30	72	36	12	44	59	27	32	8	66

Lineare Suche:

Die Folge ist unsortiert, andere elementare Verfahren sind nicht einsetzbar.

Die Zahl 32 wird nach 8 Schritten gefunden.

3.

1	2	3	4	5	6	7	8	9	10
26	27	28	30	31	32	34	36	37	38

Interpolationssuche:

Die Elemente der Folge sind (fast) gleichverteilt.

- Mitte bestimmen: $m = l + \frac{32-z[l]}{z[r]-z[l]}(r-l) = 1 + \frac{(32-26) \cdot (10-1)}{38-26} = 1 + \frac{54}{12} = 5$
- $z[5] = 31$, Fortsetzen der Suche bei den Elementen 6 bis 10
- Neue Mitte bestimmen: $m = l + \frac{32-z[l]}{z[r]-z[l]}(r-l) = 6 + \frac{(32-32) \cdot (10-6)}{38-32} = 6 + 0 = 6$
- $z[6] = 32$, die Zahl wurde nach 2 Schritten gefunden

VL11, Lösung 2

- i) Bubblesort besitzt eine Zeitkomplexität von $O(n^2)$ im schlechtesten Fall. Die binäre Suche besitzt eine Zeitkomplexität von $O(\log(n))$. Die Zeitkomplexität der Gesamtlösung ist also $O(n^2) + n/2 \cdot O(\log(n)) = O(n^2)$.
- ii) HeapSort besitzt eine Zeitkomplexität von $O(n \log(n))$ im schlechtesten Fall. Die Interpolationssuche besitzt eine Zeitkomplexität von $O(n)$ im schlechtesten Fall. Die Zeitkomplexität der Gesamtlösung ist also $O(n \log(n)) + n/2 \cdot O(n) = O(n^2)$.
- iii) Die lineare Suche besitzt eine Zeitkomplexität von $O(n)$ im schlechtesten Fall. Die Zeitkomplexität der Gesamtlösung ist also $n/2 \cdot O(n) = O(n^2)$.
- iv) Mergesort besitzt eine Zeitkomplexität von $O(n \log(n))$ im schlechtesten Fall. Die binäre Suche besitzt eine Zeitkomplexität von $O(\log(n))$. Die Zeitkomplexität der Gesamtlösung ist also $O(n \log(n)) + n/2 \cdot O(\log(n)) = O(n \log(n))$. Die ist die beste Gesamtlösung!

VL11, Lösung 3

Mit $h(k) = (\text{ord}(1. \text{ Buchstabe von } k) + \text{ord}(2. \text{ Buchstabe von } k)) \bmod 11$ berechnete Hashwerte:

$$h(\text{"STARK"}) = (19+20) \bmod 11 = 6$$

$$h(\text{"RETTINGER"}) = (18+5) \bmod 11 = 1$$

$$h(\text{"KUKUK"}) = (11+21) \bmod 11 = 10$$

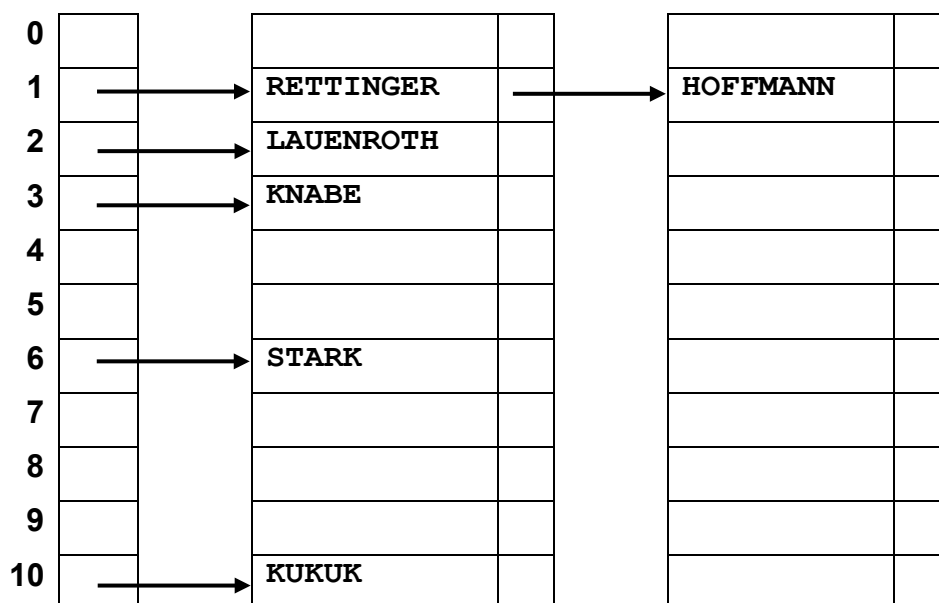
$$h(\text{"KNABE"}) = (11+14) \bmod 11 = 3$$

$$h(\text{"HOFFMANN"}) = (8+15) \bmod 11 = 1$$

$$h(\text{"LAUENROTH"}) = (12+1) \bmod 11 = 2$$

Eine Kollision entsteht für RETTINGER und HOFFMANN (jeweils Hashwert 1).

a) **Hashtabelle mit direkter Verkettung der Überläufer:**



b) Quadratisches Sondieren:

0	LAUENROTH
1	RETTINGER
2	HOFFMANN
3	KNABE
4	
5	
6	STARK
7	
8	
9	
10	KUKUK

Ausgangssituation: die Hashtabelle ist leer. Die Schlüssel STARK, RETTINGER, KUKUK und KNABE können an den für sie berechneten Hash-Adressen 6, 1, 10 und 3 abgespeichert werden.

Einfügen von HOFFMANN: Die berechnete Hash-Adresse 1 ist bereits durch RETTINGER belegt. Es wird erste Ausweichposition $1+1 = 2$ berechnet. Diese ist noch frei. Also wird HOFFMANN dort abgespeichert.

Einfügen von LAUENROTH: Die berechnete Hash-Adresse 2 ist nicht mehr frei. Die erste Ausweichposition $2+1 = 3$ ist belegt durch KNABE. Die nächste Ausweichposition $2+4 = 6$ ist ebenfalls belegt durch STARK. Die dritte Ausweichposition $(2+9) \bmod 11 = 0$ ist noch frei. Also wird LAUENROTH dort abgespeichert.

VL11, Lösung 4

Hashtabelle t:

115	15	86	38	53	75	100
-----	----	----	----	----	----	-----

- **Schlüssel 86:** $h(86) = 86 \bmod 7 = (12 \cdot 7 + 2) \bmod 7 = 2$
Der gesuchte Schlüssel befindet sich an der durch die Hashfunktion berechneten Position: $t[2] = 86$. Zum Finden des Schlüssels musste nur ein Eintrag betrachtet werden.
- **Schlüssel 100:** $h(100) = (14 \cdot 7 + 2) \bmod 7 = 2$.
 $t[2] = 86 \neq 100$. Also muss die Strategie zur Kollisionsauflösung angewandt werden, um die 100 zu finden oder festzustellen, dass sie nicht in der Hashtabelle enthalten ist. Erste Ausweichposition: $(2 + 1^2) \bmod 7 = 3$.
 $t[3] = 38 \neq 100$. Nächste Ausweichposition: $(2 + 2^2) \bmod 7 = 6$.
 $t[6] = 100$, der Schlüssel wurde gefunden. Zum Finden des Schlüssels mussten insgesamt 3 Einträge betrachtet werden: 86, 38 und 100.

VL11, Lösung 5

```
public class BinäreSuche
{
    static boolean binaereSuche(final String[] worte, final String begriff)
    {
        int linkerRand = 0;
        int rechterRand = worte.length - 1;

        while (linkerRand <= rechterRand)
        {
            int mittleresElement = (linkerRand + rechterRand) / 2;

            // Lexikographischer Vergleich zweier Zeichenketten
            // Ergebnis zwischenspeichern, da es zweimal benötigt wird
            final int vergleich =
                begriff.compareTo(worte[mittleresElement]);

            // Die gesuchte Zeichenkette wurde gefunden, Methode verlassen
            if (vergleich == 0)
                return true;

            if (vergleich < 0)
            {
                rechterRand = mittleresElement - 1;
            }
            else
            {
                linkerRand = mittleresElement + 1;
            }
        }

        // Die gesuchte Zeichenkette konnte nicht gefunden werden
        return false;
    }

    public static void main(String[] args)
    {
        String[] worte = {"Bär", "Hund", "Katze", "Löwe", "Maus", "Uhu"};

        System.out.println(binaereSuche(worte, "Bär"));
        System.out.println(binaereSuche(worte, "Katze"));
        System.out.println(binaereSuche(worte, "Uhu"));
        System.out.println(binaereSuche(worte, "Ameise"));
        System.out.println(binaereSuche(worte, "Koalabär"));
        System.out.println(binaereSuche(worte, "Wanderfalke"));
    }
}
```

VL11, Lösung 6

```
public int findePosition(IHashable o)
{
    int collisionCount = 0;

    // Verwendung der Hashmethode des einzufügenden Elements
    int hashWert = o.hash(table.length);
    int currentPos = hashWert;

    while ((table[currentPos] != null) && !table[currentPos].equals(o))
    {
        collisionCount++;
        currentPos = (hashWert + collisionCount * collisionCount) % table.length;
    }

    return currentPos;
}
```

Java-Programme zu den Lösungen der Aufgaben 5 und 6 finden Sie zusätzlich in der Datei ML11-Aufgabe_5_6.zip.