# VL11, Lösung 1

a)

| Algorithmus | Laufzeit im worst case | Anwendungsgebiete |
|---|---|---|
| Lineare Suche | $\mathrm{O}(\mathrm{n})$ | Universell einsetzbar, einziges elementares Verfahren für unsortierte Folgen |
| Binäre Suche | $\mathrm{O}(\log \mathrm{n})$ | Nur für sortierte Folgen geeignet |
| Interpolationssuche | $\mathbf{O(n)}$ | Nur für sortierte Folgen geeignet, $\mathrm{O}(\log\log \mathrm{n})$ 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: $m = l + \frac{r-l}{2} = 1 + \frac{10-1}{2} = 1 + \frac{9}{2} = 5$
- $\mathrm{z}[5] = 20$, Fortsetzen der Suche bei den Elementen 6 bis 10
- Neue Mitte bestimmen: $m = l + \frac{r-l}{2} = 6 + \frac{10-6}{2} = 6 + \frac{4}{2} = 8$
- $\mathrm{z}[8] = 160$, Fortsetzen der Suche bei den Elementen 6 bis 7
- Neue Mitte bestimmen: $m = l + \frac{r-l}{2} = 6 + \frac{7-6}{2} = 6 + \frac{1}{2} = 6$
- $\mathrm{z}[6] = 32$, 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: $m = l + \frac{32-z[l]}{z[r]-z[l]}(r-l) = 1 + \frac{(32-26)\cdot(10-1)}{38-26} = 1 + \frac{54}{12} = 5$
- $\mathrm{z}[5] = 31$, Fortsetzen der Suche bei den Elementen 6 bis 10
- Neue Mitte bestimmen: $m = l + \frac{32-z[l]}{z[r]-z[l]}(r-l) = 6 + \frac{(32-32)\cdot(10-6)}{38-32} = 6 + 0 = 6$
- $\mathrm{z}[6] = 32$, 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!

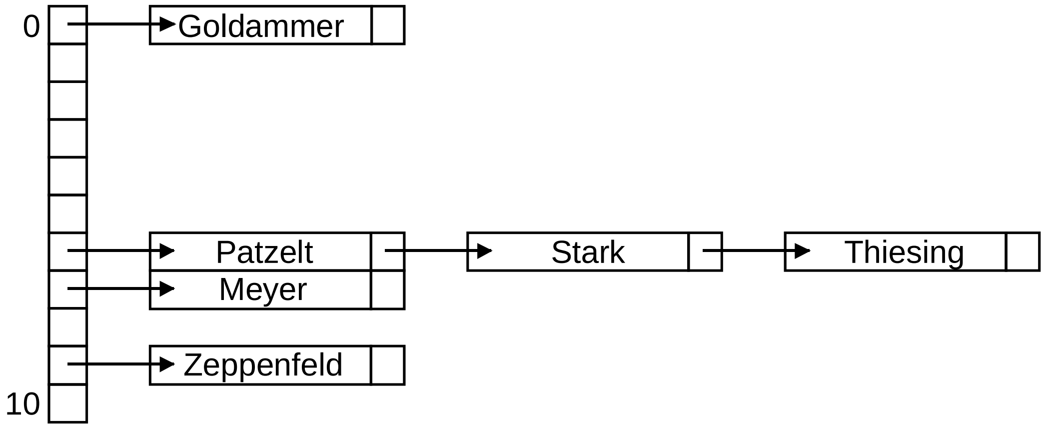
## VL11, Lösung 3

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

$\mathrm{h}(\text{“MEYER“}) = (13+5)\ \mathrm{mod}\ 11 = 7$
$\mathrm{h}(\text{“GOLDAMMER“}) = (7+15)\ \mathrm{mod}\ 11 = 0$
$\mathrm{h}(\text{“PATZELT“}) = (16+1)\ \mathrm{mod}\ 11 = 6$
$\mathrm{h}(\text{“STARK“}) = (19+20)\ \mathrm{mod}\ 11 = 6$
$\mathrm{h}(\text{“THIESING“}) = (20+8)\ \mathrm{mod}\ 11 = 6$
$\mathrm{h}(\text{“ZEPPENFELD“}) = (26+5)\ \mathrm{mod}\ 11 = 9$

Kollisionen entstehen für `PATZELT`, `STARK` und `THIESING` (jeweils Hashwert 6).

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



b) **Quadratisches Sondieren:**

| | |
|---|---|
| 0 | Goldammer |
| | |
| | |
| | |
| | Thiesing |
| | |
| | Patzelt |
| | Meyer |
| | |
| | Zeppenfeld |
| 10 | Stark |

**Ausgangssituation:** die Hashtabelle ist leer. Die Schlüssel `MEYER`, `GOLDAMMER`, und `PATZELT` können an den für sie berechneten Hash-Adressen $7$, $0$ und $6$ abgespeichert werden. Beim Einfügen von `STARK` ist Hash-Adresse $6$ bereits durch `PATZELT` belegt. Also wird erste Ausweichposition $6+1 = 7$ gewählt. Diese ist jedoch bereits durch `MEYER` belegt. Also wird nächste Ausweichposition $6+4 = 10$ geprüft. Da diese noch frei ist, wird `STARK` dort abgespeichert.

**Einfügen von `THIESING`:** Die berechnete Hash-Adresse $6$ ist nicht mehr frei. Die erste Ausweichposition $6+1 = 7$ ist belegt durch `MEYER`. Die nächste Ausweichposition $6+4 = 10$ ist ebenfalls belegt durch `STARK`. Die dritte Ausweichposition $(6+9) \bmod 11 = 4$ ist noch frei. Also wird `THIESING` dort abgespeichert.

**Einfügen von `ZEPPENFELD`:** Die berechnete Hash-Adresse $9$ ist frei. Also kann `ZEPPENFELD` dort problemlos abgespeichert werden.

# VL11, Lösung 4

Hashtabelle $t$:

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

- **Schlüssel 86:** $h(86) = 86 \bmod 7 = (12*7 + 2) \bmod 7 = 2$
  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:** $h(100) = (14*7 + 2) \bmod 7 = 2$.
  $t[2] = 86 \neq 100$. 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

```java
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

```java
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;
}
```