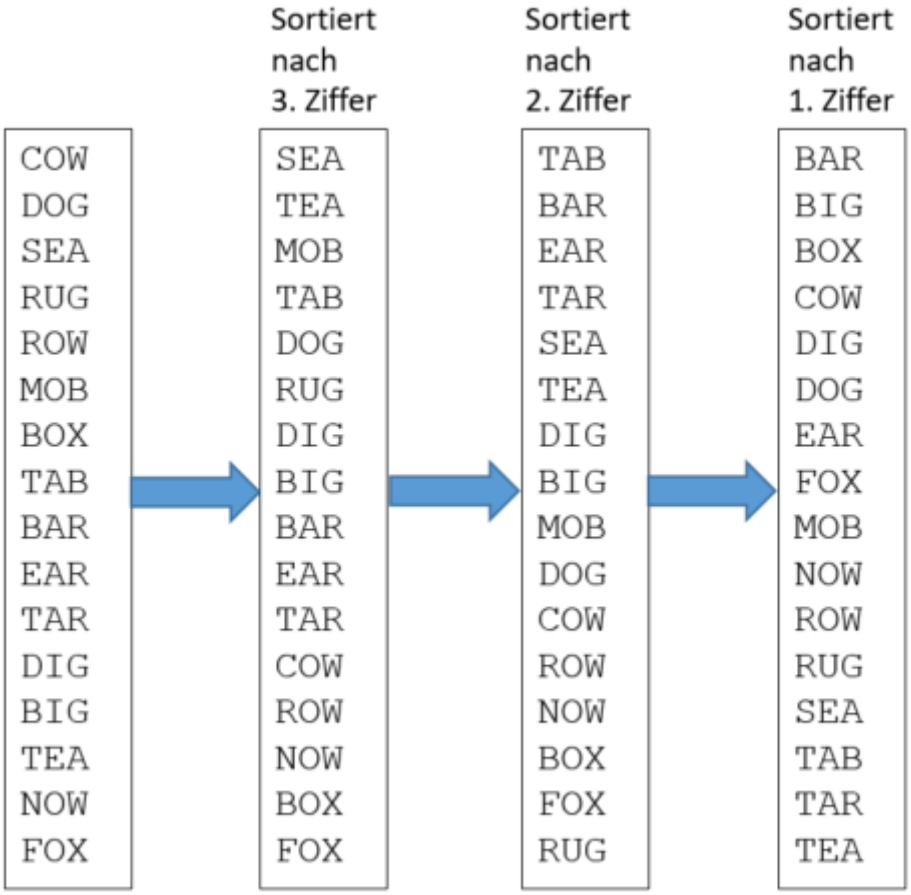
Lösung 06: Hashtabelle, Radixsort

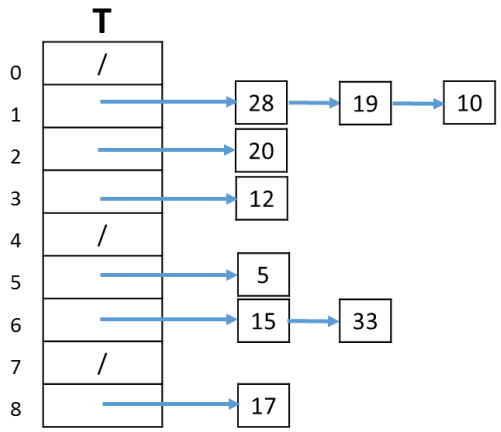
Aufgabe 1: Radixsort



Aufgabe 2: Hashtabellen – Verkettung der Überläufer

Es muss „modulo 9“ gerechnet werden.

a)



b)

- **Best Case:** 1 Schlüsselvergleich wenn gesuchter Schlüssel am Anfang einer Liste steht (z.B. 28).
- **Worst Case:** 3 Schlüsselvergleiche, falls man 10 sucht.
- **Average Case:** Es sind 9 Schlüssel vorhanden, jeder ist gleichwahrscheinlich (1/9). Manche Schlüssel benötigen nur 1 Schlüsselvergleich (z.B. Schlüssel 15), andere Schlüssel mehr Vergleiche (z.B. Schlüssel 33 benötigt 2 Vergleiche). Insgesamt ergibt sich als Erwartungswert: $\frac{1}{9} \cdot (1 + 2 + 3 + 1 + 1 + 1 + 1 + 2 + 1) = 1{,}44$. Im Mittel hat man also bei erfolgreicher Suche 1,44 Schlüsselvergleiche.

Aufgabe 3: Hashtabellen – Lineares Sondieren

a) Ergebnis beim linearen Sondieren:

0	1	2	3	4	5	6	7	8	9	10
22				4	15	28	59		31	10

b) Vor Beginn der 2. While-Schleife: Die 4 wurde gelöscht

0	1	2	3	4	5	6	7	8	9	10
22					15	28	59		31	10

Nach der 1. Iteration der 2. While-Schleife: Die 15 wurde zunächst gelöscht und dann erneut in die Hashstabelle eingefügt. Sie fällt dann auf Index 4.

0	1	2	3	4	5	6	7	8	9	10
22				15		28	59		31	10

Nach der 2. Iteration der 2. While-Schleife: Die 28 (28 mod 11 = 6) wurde zunächst gelöscht und dann erneut in die Hashstabelle eingefügt. Sie fällt erneut auf Index 6.

0	1	2	3	4	5	6	7	8	9	10
22				15		28	59		31	10

Nach der 3. Iteration der 2. While-Schleife: Die 59(59 mod 11 = 4)wurde zunächst gelöscht und dann erneut in die Hashstabelle eingefügt. Sie fällt erneut auf Index 7.

0	1	2	3	4	5	6	7	8	9	10
22				15	59	28			31	10

Es erscheint überflüssig, dass man bis zum rechten „Clusterende“ läuft und die Zahlen erneut „rehasht“. Genügt es nicht, dass man so lange durch den Cluster läuft, bis man auf die erste Zahl trifft, die „Modulo 11“ nicht mehr 4 ergibt? Konkret wäre das hier schon 28.
Dennoch gibt es keine Alternative zu diesem Vorgehen. Ansonsten würde man im konkreten Fall z.B. die 59 nicht auf Indexposition 5 bringen. 8 an der gleichen Position geblieben, aber 37 (33 mod 11 = 4) wäre beim „Rehashen“ auf den nun freien Index 5 gefallen.

Aufgabe 4: Lineares Sondieren – *Lazy Delete*

Lösung, siehe Quelltext im Gitlab

Hier eine Liste der Änderungen:

- Zusätzliche Variable, die die Anzahl der ungültigen / als gelöscht markierten Keys speichert:
```
private int numInvalid;  // TODO: number of invalid/deleted keys
```

- Methode **put**: Modifizierte Entscheidung, wann „rehasht“ wird und Verwenden eines als gelöscht markierten Slots, um eine neuen Wert einzutragen

```
// more than 50% of entries are valid, rehash und double table size
if (n >= m / 2) {
    resize(2 * m);
}
// rehash but keep table size if total number of entries (incl. invalid)
exceeds m/2
else if (n + numInvalid >= m / 2) {
    resize(m);
}
```

```
else if(vals[i] == null) { // TODO: key marked as invalid/deleted -> reuse
    keys[i] = key;
    vals[i] = val;
    n++;
   numInvalid--;
}
```

- Methode **get:** Schlüssel ignorieren, die ungültig sind (val-Value)

```
if (vals[i] != null && keys[i].equals(key))     // TODO
```

- Und natürlich Umschreiben der Methode **delete.**