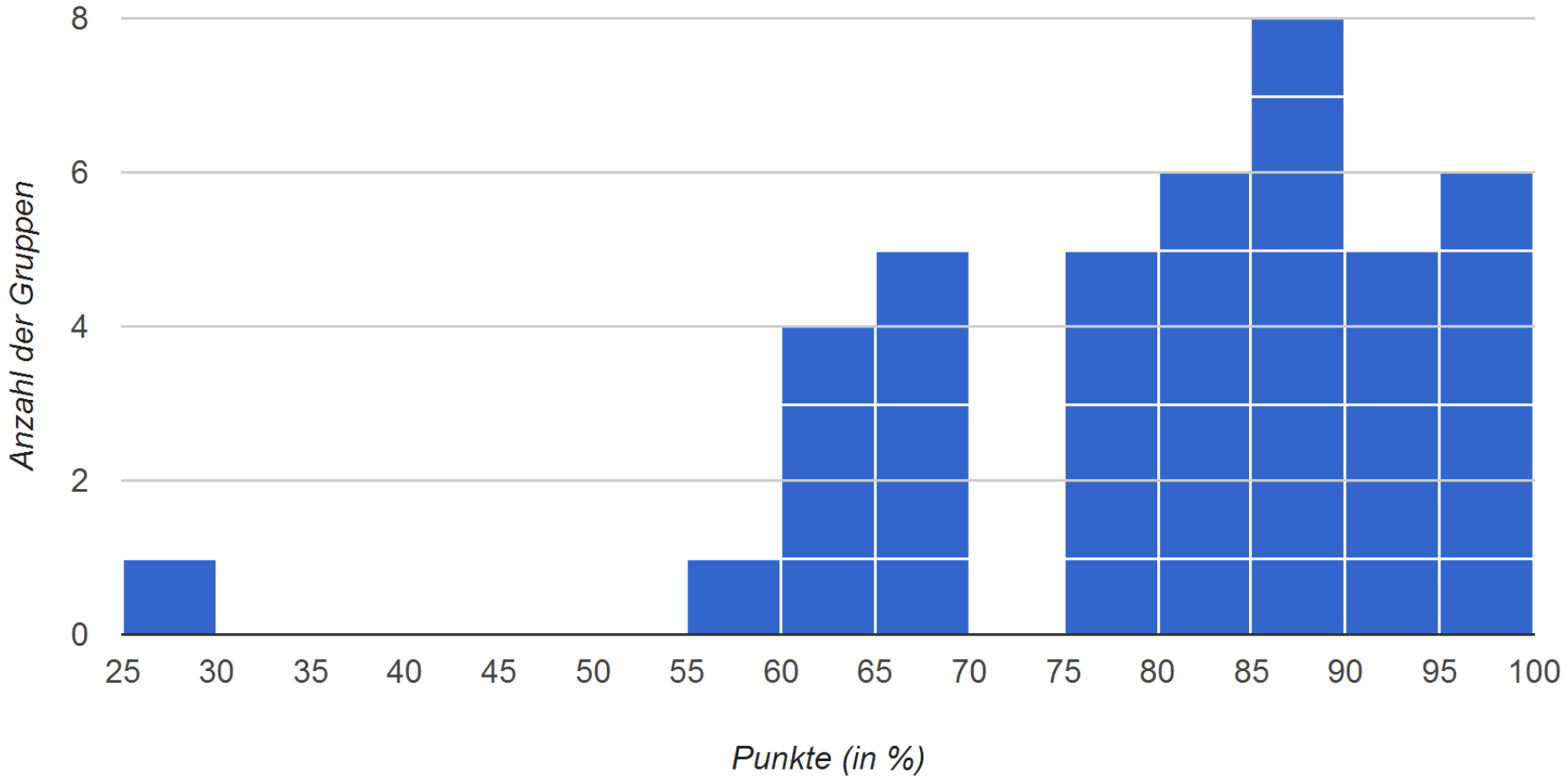
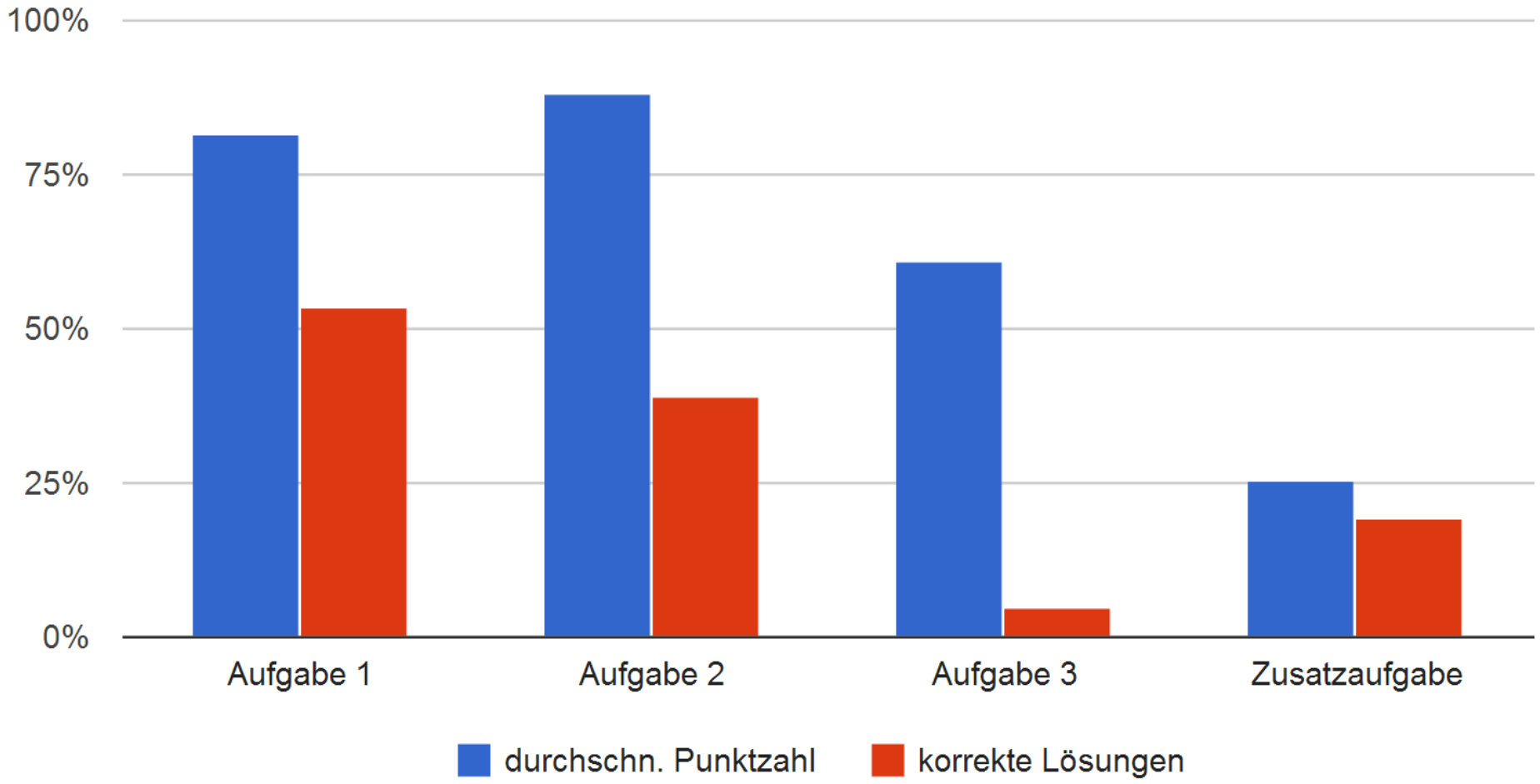
Übungsblatt 3 - Auswertung

Eigenschaften von Algorithmen

Punktverteilung (Gesamt)



Ergebnisse pro Aufgabe



Aufgabe 1 - Korrektheit

- Testabdeckung (C1-Testabdeckung/Branch coverage)
  - Häufige Fehler: nicht alle Zweige abgedeckt

Test

```
@Test
public void testMax() {
    SquareMatrix matrix = new SquareMatrix(3, InitStrategy.Plus);
    assertEquals(matrix.max(), 4.0, 0);
}
```

Abdeckung:

```
    /**
     * find the maximal value in matrix
     * @return the maximum
     */
    public double max() {
        double max = Double.NEGATIVE_INFINITY;
        for (int i = 0; i < size(); i++) {
            for (int j = 0; j < size(); j++) {
                final double current = get(i, j);
                if (Double.isNaN(current)) {
                    continue;
                } else if (!(max >= current)) {
                    max = current;
                }
            }
        }
        return max;
    }
```

Aufgabe 1 - Korrektheit

- Testabdeckung (C1-Testabdeckung/Branch coverage)
  - Häufige Fehler: nicht alle Zweige abgedeckt

Test

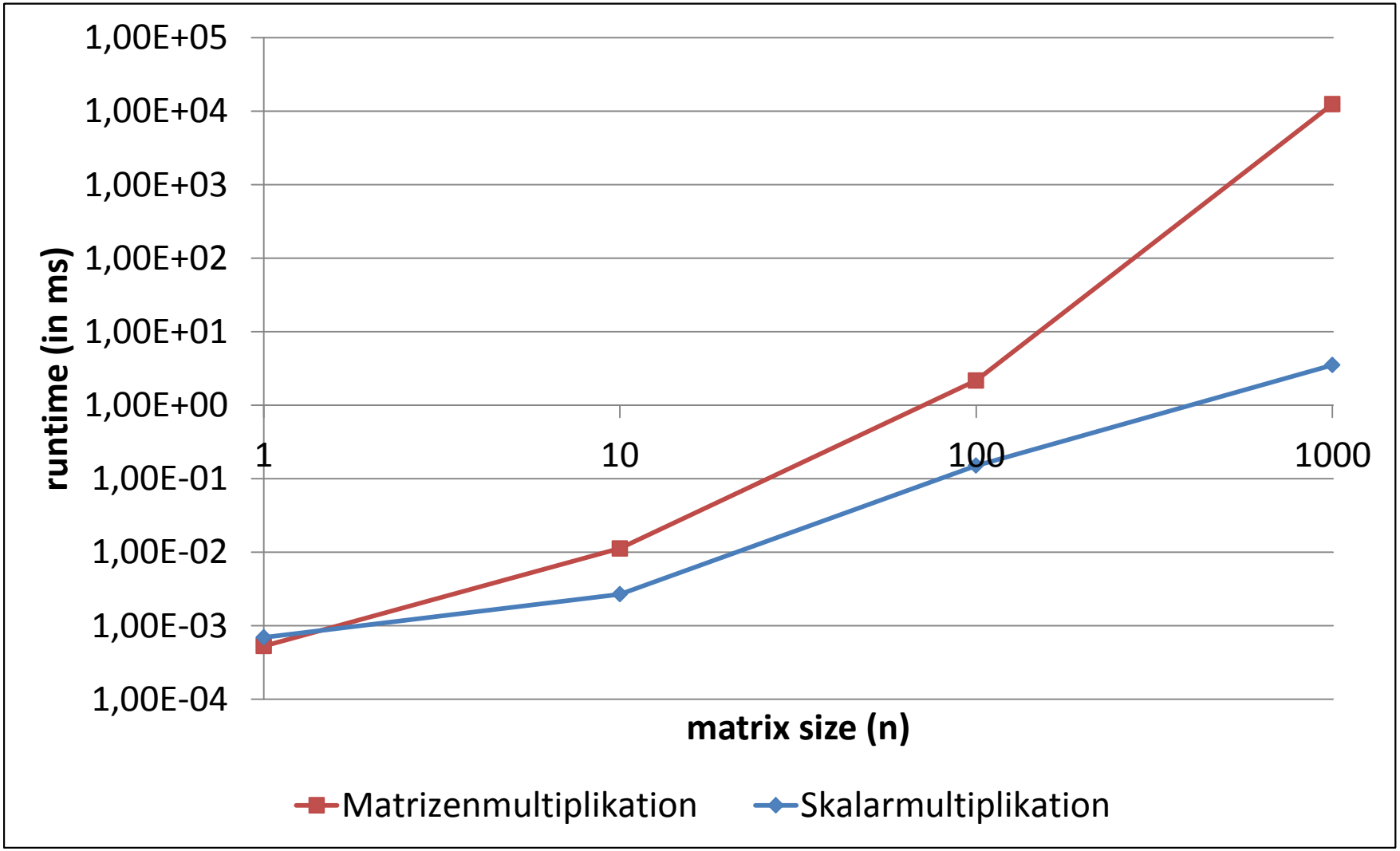
```
@Test
public void testMax() {
    SquareMatrix matrix = new SquareMatrix(3, InitStrategy.Plus);
    matrix.set(1, 1, Double.NaN); // add NaN
    assertEquals(matrix.max(), 4.0, 0);
}
```

Abdeckung:

```
    /**
     * find the maximal value in matrix
     * @return the maximum
     */
    public double max() {
        double max = Double.NEGATIVE_INFINITY;
        for (int i = 0; i < size(); i++) {
            for (int j = 0; j < size(); j++) {
                final double current = get(i, j);
                if (Double.isNaN(current)) {
                    continue;
                } else if (!(max >= current)) {
                    max = current;
                }
            }
        }
        return max;
    }
```

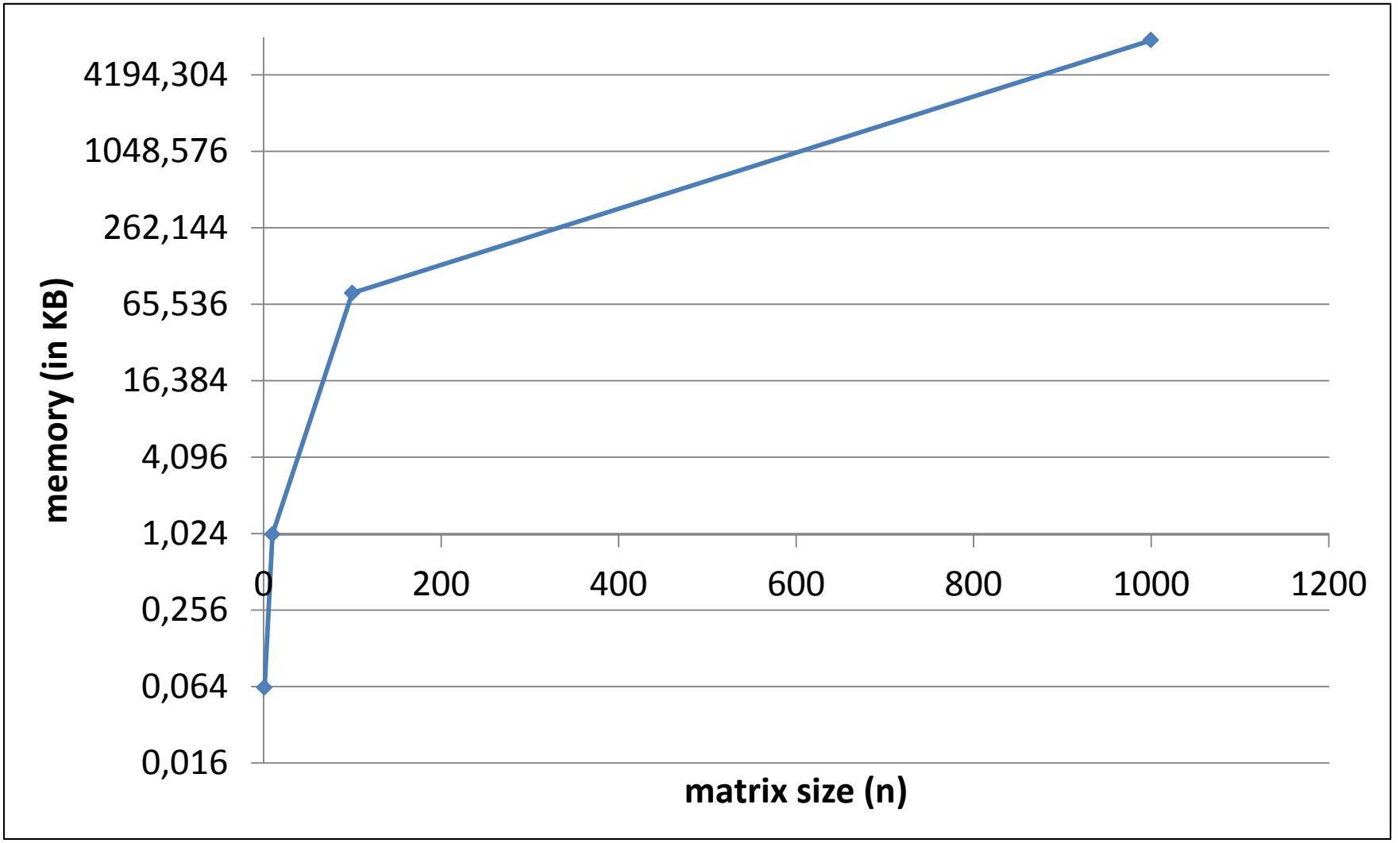
Aufgabe 2 - Zeitkomplexität

- Erwartete Laufzeit



Aufgabe 3 - Platzkomplexität

- Erwartete Speicherbedarf: n^2 (Zielmatrix)



Zusatzaufgabe

- Gegeben:
  - zwei Arrays *m[1..K]* und *z[1..K]* mit Werten
  - Ein Array *q[1..Q]*
- Gesucht *q[i]* kleinste Summe aus *m[j]* und *z[k]*
- Brutforce Lösung
  - Berechne alle Summen
  - Sortiere die K^2 Werte
  - Entnehme Werte *q[i]*
  - $O(K^2 + K^2 \log K + Q)$
- Es geht geschickter!

Zusatzaufgabe

- Sortiere *m* und *z* → Quicksort: *O(K log K)*
- Merke für jeden Wert *m[j]* den Verweis (*low[j]*) zum niedringsten Partner aus *z* → initial ist also *low[j]=0 (∀ j)*
- Speichere die Summen *m[j] + z[low[j]]* in einer priority queues (z.B. Min-Heap)
- Wiederhole für *i=0..max(q)-1*
  - Entnehme *min (j)* aus der queue: *minSum[i] = m[j] + z[low[j]]*
  - Ändere den Verweis von *j*: *low[j] = low[j]+1*
  - Füge den Nachfolger zur queue hinzu: *m[j] + z[low[j]]*
- Ermittle die Ergebnisse: *res[i]=minSum[q[i]] (i=0..Q-1)*
- *O(K log K + max(q) log K + Q)*