Aufgabe 2

Geben Sie jeweils die kleinste, gerade noch passende Laufzeitkomplexität folgender Java-Methoden im O-Kalkül (Landau-Notation) in Abhängigkeit von n und ggf. der Länge der Arrays an.

(a)

```
5   int matrixSumme(int n, int[][] feld) {
6     int sum = 0;
7     for (int i = 0; i < n; i++) {
8       for (int j = 0; j < n; j++) {
9         sum += feld[i][j];
10      }
11    }
12    return sum;
13  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2016/herbst/Komplexitaet.java`

Die Laufzeit liegt in $\mathcal{O}(n^2)$.

Begründung (nicht verlangt): Die äußere Schleife wird n-mal durchlaufen. Die innere Schleife wird dann jeweils wieder n-mal durchlaufen. Die Größe des Arrays spielt hier übrigens keine Rolle, da die Schleifen ohnehin immer nur bis zum Wert n ausgeführt werden.

(b)

```
15  int find(int key, int[][] keys) {
16    int a = 0, o = keys.length;
17    while (o - a > 1) {
18      int m = (a + o) / 2;
19      if (keys[m][0] > key)
20        o = m;
21      else
22        a = m;
23    }
24    return a;
25  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2016/herbst/Komplexitaet.java`

Die Laufzeit liegt in $\mathcal{O}(\log(\text{keys.length}))$. Dabei ist keys.length die Größe des Arrays bezüglich seiner ersten Dimension.

Begründung (nicht verlangt): Der Grund für diese Laufzeit ist derselbe wie bei der binären Suche. Die Größe des Arrays bezüglich seiner zweiten Dimension spielt hier übrigens keine Rolle, da diese Dimension hier ja nur einen einzigen festen Wert annimmt.