Aufgabe 6 (Algorithmen und Datenstrukturen)

Aus dem Känguru-Wettbewerb 2017 — Klassenstufen 3 und 4.

> Luna hat für den Kuchenbasar Muffins mitgebracht: 10 Apfelmuffins, 18 Nussmuffins, 12 Schokomuffins und 9 Blaubeermuffins. Sie nimmt immer 3 verschiedene Muffins und legt sie auf einen Teller. Welches ist die kleinste Zahl von Muffins, die dabei übrig bleiben können?
>
> A: 1, B: 3, C: 4, D: 7, E: 8

(a) Geben Sie die richtige Antwort auf die im Känguru-Wettbewerb gestellte Frage und begründen Sie sie.

4 [a]

[a] https://www.youtube.com/watch?v=ceJW9kAplVY

(b) Lunas Freundin empfiehlt den jeweils nächsten Teller immer aus den drei aktuell häufigsten Muffinsorten zusammenzustellen. Leiten Sie aus dieser Idee einen effizienten GreedyAlgorithmus her, der die Fragestellung für beliebige Anzahlen von Muffins löst (nach wie vor soll es nur vier Sorten und je drei pro Teller geben). Skizzieren Sie in geeigneter Form, wie Ihr Algorithmus die Beispielinstanz von oben richtig löst.

```java
public static int berechneRest4Sorten3ProTeller() {
  int[] muffins = new int[] { 10, 18, 12, 9 };
  int n = muffins.length;
  sortiere(muffins);

  // Wir nehmen uns 3 verschiedene Muffins solange, wie die
  ↪ dritthäufigste
  // Muffinsorte noch Muffins hat.
  while (muffins[n - 3] > 0) {
    muffins[n - 1]--;
    muffins[n - 2]--;
    muffins[n - 3]--;
    sortiere(muffins);
  }
  return berechneGesamtzahl(muffins);
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2019/fruehjahr/Muffin.java`

(c) Beschreiben Sie eine mögliche und sinnvolle Verallgemeinerung Ihrer Lösung auf n Muffinsorten und k Muffins pro Teller für $n > 4$ und $k > 3$.

```java
public static int berechneRestAllgemein(int[] muffins, int k) {
  int n = muffins.length;
  sortiere(muffins);
  while (muffins[n - k] > 0) {
    for (int i = 1; i <= k; i++) {
      muffins[n - i]--;
    }
    sortiere(muffins);
```

```
75        }
76        return berechneGesamtzahl(muffins);
77    }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2019/fruehjahr/Muffin.java`

(d) Diskutieren Sie, wie man die Korrektheit des Greedy-Algorithmus zeigen könnte, also dass er tatsächlich immer eine optimale Lösung findet. Ein kompletter, rigoroser Beweis ist nicht verlangt.

Additum

Der komplette Code:

```
3  import java.util.Arrays;
4
5  public class Muffin {
6
7    /**
8     * Diese Hilfemethode berechnet die Gesamtzahl der Muffins.
9     *
10    * @param muffins Ein Feld mit der Gesamtzahl der Muffins je Sorte.
11    *
12    * @return Gesamtzahl an Muffins (Alle Muffins egal von welcher Sorte
      ↪ addiert)
13    */
14   public static int berechneGesamtzahl(int[] muffins) {
15     int gesamtzahl = 0;
16     for (int i = 0; i < muffins.length; i++) {
17       gesamtzahl += muffins[i];
18
19     }
20     return gesamtzahl;
21   }
22
23   /**
24    * Sortiere die Anzahl je Muffinsorte aufsteigend. Das letzte Element im
      ↪ Feld
25    * ist die Sorte mit den meisten Muffins. Diese Methode gibt das Feld nach
      ↪ der
26    * Sortierung auf der Konsole aus. So können wir den Greedy-Algorithmus
      ↪ besser
27    * nachvollziehen.
28    *
29    * @param muffins Ein Feld mit der Gesamtzahl der Muffins je Sorte.
30    */
31   public static void sortiere(int[] muffins) {
32     Arrays.sort(muffins);
33     System.out.println(Arrays.toString(muffins));
34   }
35
36   /**
37    * Implementiert den Greedy-Algorithmus für 4 Muffinsorten und 3
      ↪ Muffinentnahmen
38    * pro Teller.
39    *
40    * @return Die Anzahl der übrig gebliebenen Muffins.
41    */
42   public static int berechneRest4Sorten3ProTeller() {
```

```java
    int[] muffins = new int[] { 10, 18, 12, 9 };
    int n = muffins.length;
    sortiere(muffins);

    // Wir nehmen uns 3 verschiedene Muffins solange, wie die dritthäufigste
    // Muffinsorte noch Muffins hat.
    while (muffins[n - 3] > 0) {
      muffins[n - 1]--;
      muffins[n - 2]--;
      muffins[n - 3]--;
      sortiere(muffins);
    }
    return berechneGesamtzahl(muffins);
  }

  /**
   * Implementiert den Greedy-Algorithmus für allgemeine Werte.
   *
   * @param muffins Ein Feld mit der Gesamtzahl der Muffins je Sorte.
   * @param k       Die Anzahl an verschiedenen Muffings, die pro Teller
   ↪  entnommen
   *                werden.
   *
   * @return Die Anzahl der übrig gebliebenen Muffins.
   */
  public static int berechneRestAllgemein(int[] muffins, int k) {
    int n = muffins.length;
    sortiere(muffins);
    while (muffins[n - k] > 0) {
      for (int i = 1; i <= k; i++) {
        muffins[n - i]--;
      }
      sortiere(muffins);
    }
    return berechneGesamtzahl(muffins);
  }

  public static void main(String[] args) {
    System.out.println(berechneRest4Sorten3ProTeller());
    System.out.println(berechneRestAllgemein(new int[] { 10, 18, 12, 9, 11
    ↪  }, 4));
  }

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2019/fruehjahr/Muffin.java`