- Variante 1

Lösung Teil 1

Die heutige Aufgabe war ziemlich schwer zu verstehen und dadurch auch noch schwerer zu erklären. Auch mangels Zeit gibt es daher für diese Variante 1 leider nur einen Lösungsvorschlag. Diese Lösung ist noch dazu etwas verzwickt nachzuvollziehen, da sie als Haupt-Datenstruktur eine ArrayList von ArrayLists von long-Arrays nutzt (`ArrayList<ArrayList<long[]>`). Dafür ist sie aber sehr effizient und verzichtet quasi auf jegliche Schleifen (von der direkten Länge der Eingabedaten abgesehen).

Lösungsvorschlag Teil 1

```
private long[] numbersToIntArray(String numbersStr) {
    String[] numbers = numbersStr.trim().split(" ");
    long[] result = new long[numbers.length];

    for (int i = 0; i < numbers.length; i++) {
        result[i] = Long.parseLong(numbers[i]);
    }

    return result;
}

private long getMapping(ArrayList<long[]> map, long input) {
    for (long[] line : map) {
        if (input >= line[1] && input < line[1] + line[2]) {
            return line[0] + (input - line[1]);
        }
    }
    return input;
}

public long partOne() {

    long[] seeds = numbersToIntArray(inputLines.get(0).split(":")[1]);
    ArrayList<ArrayList<long[]>> allMaps = new
ArrayList<ArrayList<long[]>>();
    ArrayList<long[]> map = new ArrayList<long[]>();

    for (int l = 2; l < inputLines.size(); l++) {
        String line = inputLines.get(l);

        if (line.length() <= 1) {
            allMaps.add(map);
            continue;
        }

        if (line.contains(":")) {
            map = new ArrayList<long[]>();
            continue;
```

```
        }

        map.add(numbersToIntArray(line));

    }
    // add last map to allMaps
    allMaps.add(map);

    ArrayList<Long> seedResults = new ArrayList<Long>();
    for (long seed : seeds) {
        for (int mapCount = 0; mapCount < allMaps.size(); mapCount++) {
            seed = getMapping(allMaps.get(mapCount), seed);
        }
        seedResults.add(seed);
    }

    long smallest = seedResults.get(0);
    for (int i = 1; i < seedResults.size(); i++) {
        if (seedResults.get(i) < smallest) {
            smallest = seedResults.get(i);
        }
    }

    return smallest;
}
```

Lösung Teil 2

Der Lösungsvorschlag für Teil 2 baut auf Teil 1 auf. Es wird also nur minimal am Code geändert und für jeden möglichen Eingabe-Seed wird das Ergebnis berechnet. Dies ist ziemlich ineffizient und benötigt viel Rechenzeit, da größenordnungsmäßig 2 Milliarden Eingaben berechnet werden müssen (dauert etwa mindestens 10 Minuten zum Berechnen).

Vermutlich wäre eine effizientere Lösung möglich, indem man die ganzen Mappings rückwärts in der Eingabedatei von unten nach oben durchgeht. Man könnte also z. B. bei der kleinsten möglichen location (0) beginnen, die ganzen Rechnungen der Mappings rückwärts gehen und dann so lange die location erhöhen bis man einen erlaubten Seed findet. Das sollte hoffentlich schneller gehen.

Hier aber der "langsame" Lösungsvorschlag:

Lösungsvorschlag Teil 2

```
public long partTwo() {
    long[] seeds = numbersToIntArray(inputLines.get(0).split(":")[1]);
    ArrayList<ArrayList<long[]>> allMaps = new
ArrayList<ArrayList<long[]>>();
    ArrayList<long[]> map = new ArrayList<long[]>();

    for (int l = 2; l < inputLines.size(); l++) {
```

```
        String line = inputLines.get(l);

        if (line.length() <= 1) {
            allMaps.add(map);
            continue;
        }

        if (line.contains(":")) {
            map = new ArrayList<long[]>();
            continue;
        }

        map.add(numbersToIntArray(line));

    }
    // add last map to allMaps
    allMaps.add(map);

    long smallest = Long.MAX_VALUE;
    for (int i = 0; i < seeds.length; i+=2) {
        for (int x = 0; x < seeds[i+1]; x++) {
            long seed = seeds[i] + x;

            for (int mapCount = 0; mapCount < allMaps.size(); mapCount++) {
                seed = getMapping(allMaps.get(mapCount), seed);
            }
            if (seed < smallest) {
                smallest = seed;
            }
        }

    }

    return smallest;

}
```