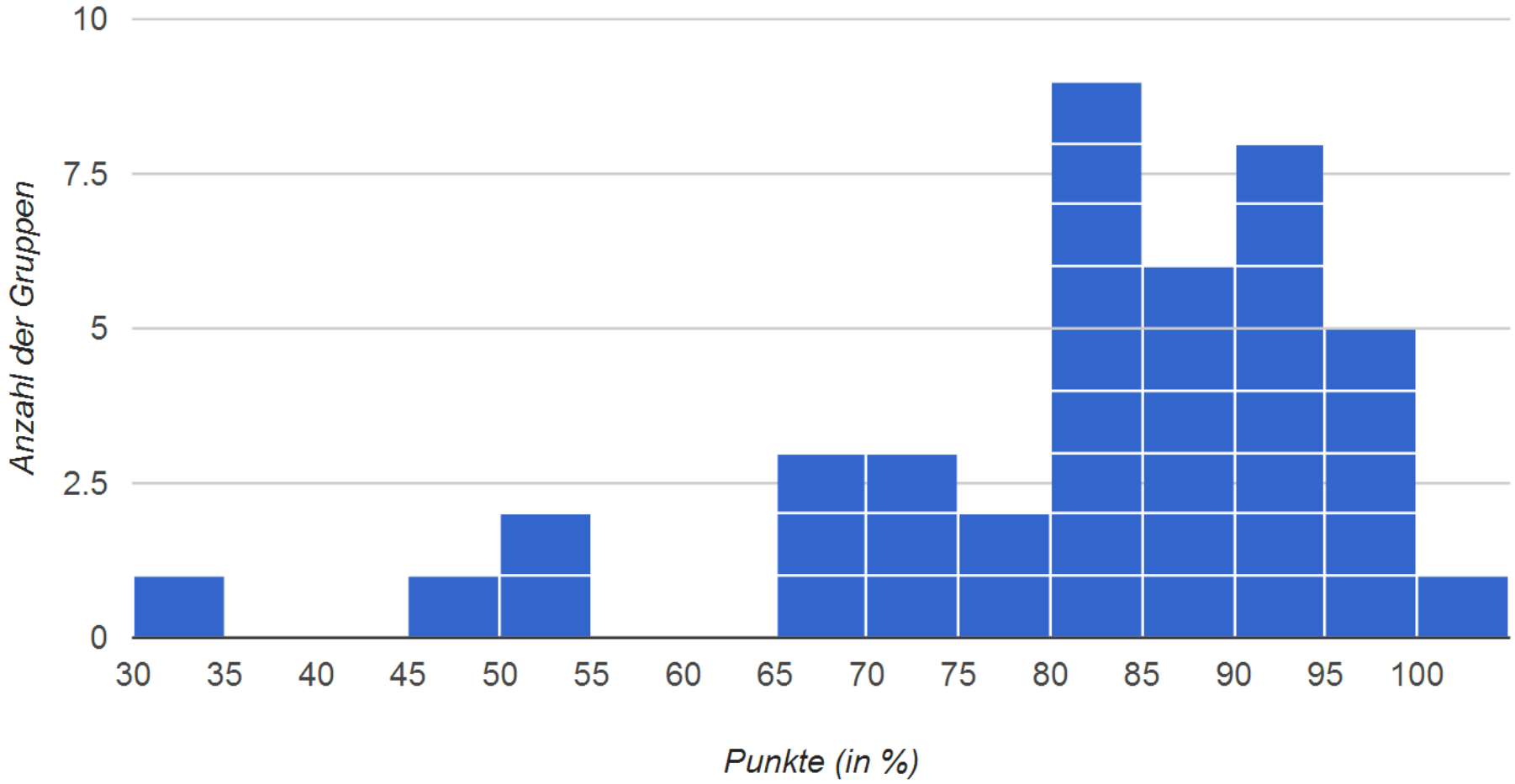
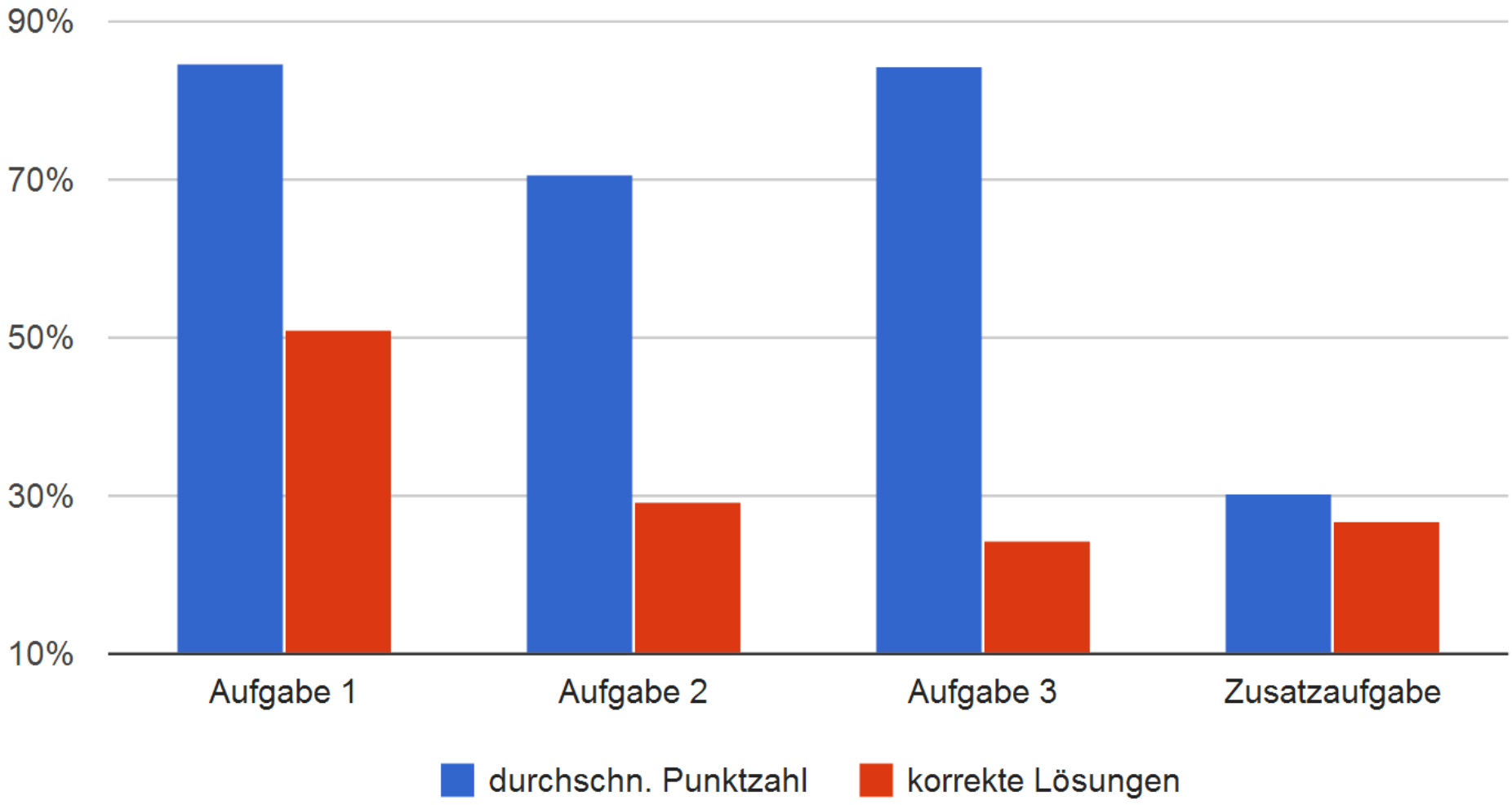
Übungsblatt 4 - Auswertung

Entwurf von Algorithmen

Punktverteilung (Gesamt)



Ergebnisse pro Aufgabe



Aufgabe 1 - Tankstellensuche

- Gegeben
  - Tankfüllung reicht für *d* km
  - Gesamtstrecke *z* km lang (*z>d*)
  - Tankstellenentfernungen *t[]* km
- Gesucht: Tankstopps

```
1  public static Integer[] getStopsI(int d, int z, int[] t) {
2    int traveledDist = 0;
3    int stopIdx = 0;
4    ArrayList<Integer> stops = new ArrayList<Integer>();
5    // drive until end
6    while(traveledDist+d <= z && stopIdx<t.length) {
7      // greedy loop (try to reach the next station)
8      for(int currDist = 0;currDist + t[stopIdx] <= d;stopIdx++) {
9        currDist+=t[stopIdx];
10       traveledDist+=t[stopIdx];
11     }
12     stops.add(stopIdx-1);
13   }
14   return stops.toArray(new Integer[stops.size()]);
15 }
```

Aufgabe 1 - Tankstellensuche

- Gegeben
  - Geschwindigkeit Prof. Naumann
  - Geschwindigkeit Fahrrad
  - #Tankstopps (1.a)
- Gesucht: Wer ist schneller?

```
1  public static int timeCompare(int kmhProf, int kmhBike, int d, int z, int[]t) {
2      Integer[] stops = getStopsI(d,z,t);
3      if(stops.length == 0) return Integer.MAX_VALUE;
4      float timeProf = z/(float)kmhProf + .25f * stops.length;
5      float timeBicycle = z/(float)kmhBike;
6      return (timeProf<timeBicycle)? 1 : ((timeProf>timeBicycle)?-1:0);
7  }
```

Aufgabe 2 – Teile und Herrsche

- Mergesort (siehe Volesungsfolien)
- Filesort
  - Teile Ursprungsdatei in Chunks ein
  - Sortiere Chunks einzeln
  - Merg
  - Lösch
  - Hilfsf
    - re
    - wr
      bc
    - de

```java
public static void filesort(String inputFile, String outputFile, int maxLineCount) {
  ArrayList<String> tempFiles = new ArrayList<>();
  ArrayList<String> buffer = new ArrayList<>(maxLineCount);

  // split
  Iterator<String> inputLines = read(Arrays.asList(inputFile)).get(0);
  String line;
  while(inputLines.hasNext() && null!=(line=inputLines.next())) {
    buffer.add(line);
    if(buffer.size()>=maxLineCount || !inputLines.hasNext()) {
      // new temp file
      String tmpFileName = outputFile + "_tmp_"+ tempFiles.size();
      tempFiles.add(tmpFileName);

      // sort buffer
      Collections.sort(buffer);
      write(buffer.iterator(), tmpFileName, false);

      buffer.clear();
    }
  }
  // ...
}
```

- M
- F

```java
public static void filesort(String inputFile, String outputFile, int maxLineCount) {
  ArrayList<String> tempFiles = new ArrayList<>();
  ArrayList<String> buffer = new ArrayList<>(maxLineCount);

  // split
  // ...

  // merge
  List<Iterator<String>> content = read(tempFiles);
  ArrayList<String> topLines = new ArrayList<>();
  for(int i=0;i<tempFiles.size();i++) {
    topLines.add(content.get(i).next());
  }

  while(true) {
    // search minimum
    int minI=-1;
    for(int i=0;i<tempFiles.size();i++) {
      if(topLines.get(i)!=null && (minI<0 || topLines.get(i).compareTo(topLines.get(minI))<0)) {
        minI = i;
      }
    }
    if(minI<0) {
      // all temp files read
      write(buffer.iterator(), outputFile, true);
      break;
    }
    // add min line to buffer
    buffer.add(topLines.get(minI));
    // get next line
    topLines.set(minI, content.get(minI).next());

    if(buffer.size()>=maxLineCount) {
      // frite
      write(buffer.iterator(), outputFile, true);
      buffer.clear();
    }
  }

  // ...
}
```

Aufgabe 2 – Teile und Herrsche

- Mergesort (siehe Volesungsfolien)
- Filesort
  - Teile Ursprungsdatei in Chunks ein
  - Sortiere Chunks einzeln
  - Merge Chunks durch ziehen des Minimums aller Dateien
  - Lösche Chunks
  - Hilfsfunktionen:
    - `read(List<String> filename)` → `List<Iterator<String>>`
    - `write(Iterator<String> data, String outputFile, bo`
    - `de`

```java
public static void filesort(String inputFile, String outputFile, int maxLineCount) {
    ArrayList<String> tempFiles = new ArrayList<>();
    ArrayList<String> buffer = new ArrayList<>(maxLineCount);

    // split & merge
    // ...

    // delete temp files
    delete(tempFiles);
}
```

Aufgabe 3 – Sudoku lösen

- Sudoku-Solver
- Backtracking

```java
public boolean solve(SudokuBoard sudoku) {
  // set all cells
  for (int row = 0; row < 9; row++) {
    for (int col = 0; col < 9; col++) {
      if (sudoku.isSet(row, col))
        // ignore filled cells
        continue;
      }
      for (int value = 1; value <= 9; value++) {
        // check if value-positioning is valid
        if (isValid(value, row, col, sudoku)) {
          // try to find a solution
          sudoku.setCell(value, row, col);
          if (solve(sudoku)) {
              return true;
          } else {
            // backtrack
            sudoku.removeValue(row, col);
          }
        }
      }
      return false;
    }
  }
  return true;
}
```

Aufgabe 3 – Sudoku erstellen

- Effiziente Generierung von Zufallsbelegungen:

```java
private Integer[] generateRandomNumbers(int n) {
  ArrayList<Integer> randoms = new ArrayList<Integer>();
  for (int i = 0; i < n; i++)
    randoms.add(i + 1);
  // shuffle all values randomly
  Collections.shuffle(randoms);
  return randoms.toArray(new Integer[n]);
}
```

- Erstelle zufällige Zellreihenfolge (Zellen 1-81)
- Pro Zufallszelle (bis genügend Zellen gefüllt sind)
  - Zellwerte (1-9) in zufälliger Reihenfolge
  - Pro mögliche Zellwert
    - Füge nächsten Wert zum Sudoku hinzu
    - Prüfe ob das Sudoku noch lösbar ist (Verwendung des Solvers)
    - Wenn nicht: wähle nächsten zufälligen Zellwert
    - Sonst: Wähle die nächste Zufallszelle

Zusatzaufgabe

- Das Netz der nächsten Generation