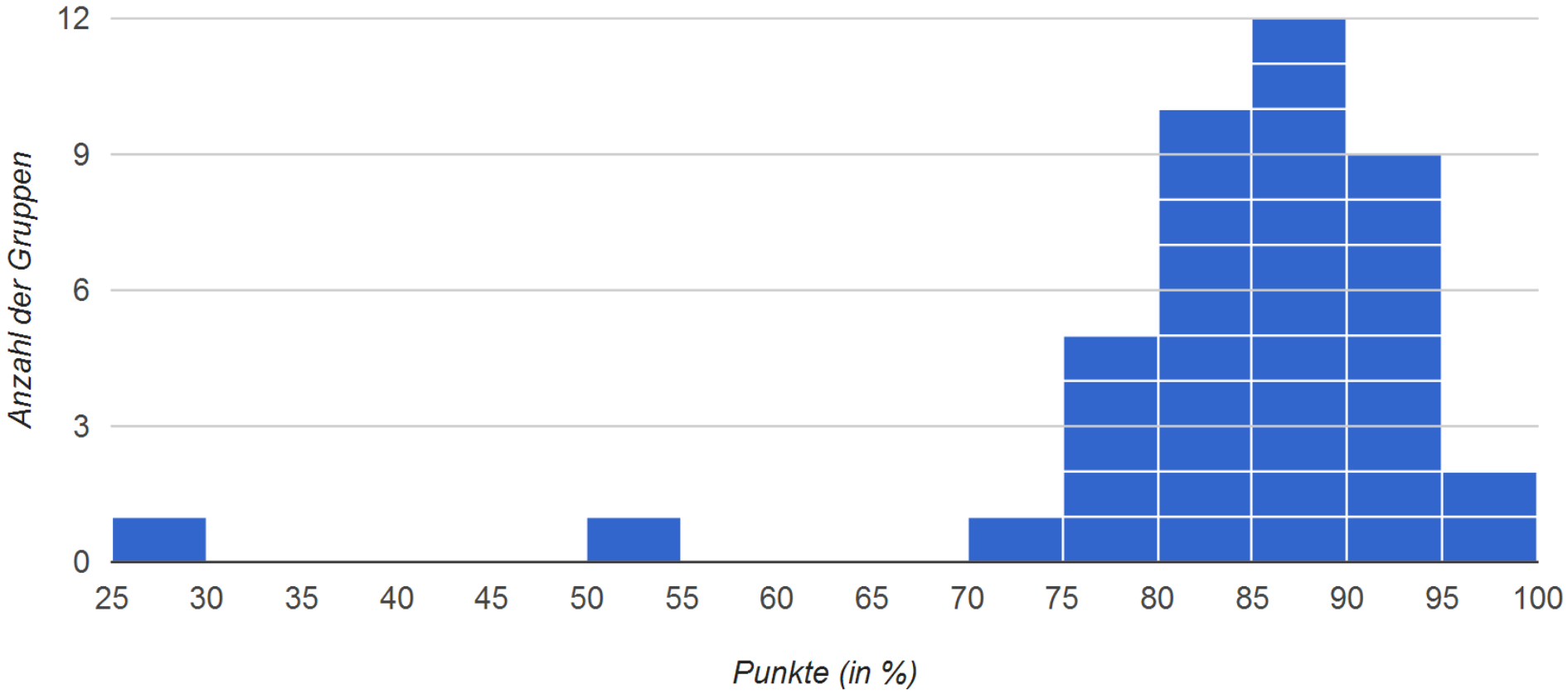
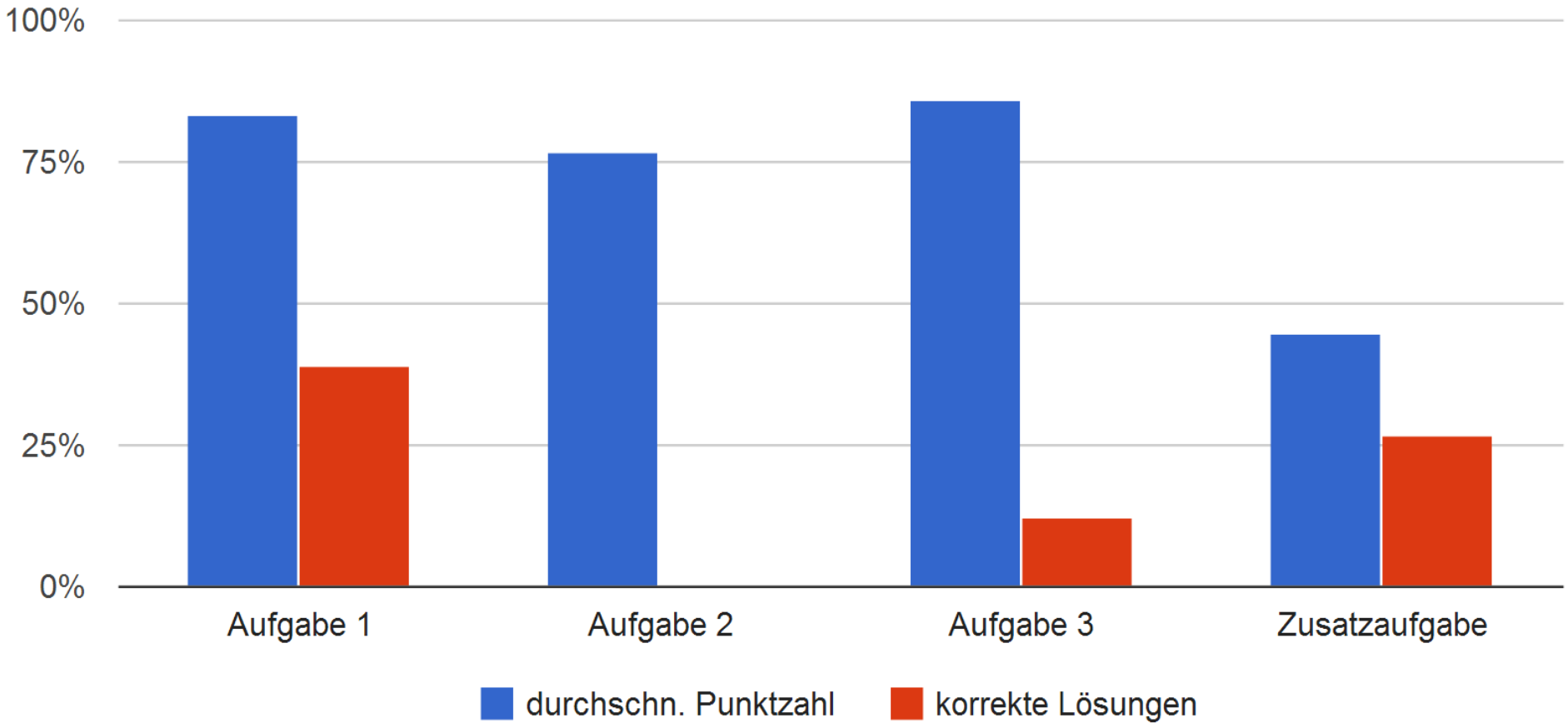
Übungsblatt 2 - Auswertung

Suchen und Sortieren

Punktverteilung (Gesamt)



Ergebnisse pro Aufgabe



Allgemeines

- Kommentare!
- Tests!

Aufgabe 1 - Suchen

- Suche **alle** Vorkommen (m) eines Wertes in einer Liste
- Unsortierte Liste → linearer Scan der Daten
  - Bester Fall: n
  - Durchschnitt (erfolgreiche Suche): n
  - Durchschnitt (erfolglose Suche): n
  - Schlechtester Fall: n
- Sortierte Liste → binäre Suche
  - Bester Fall: m
  - Durchschnitt (erfolgreiche Suche): $\approx m + \log_2 n$
  - Durchschnitt (erfolglose Suche): $\approx \log_2 n$
  - Schlechtester Fall: n $(m = n)$

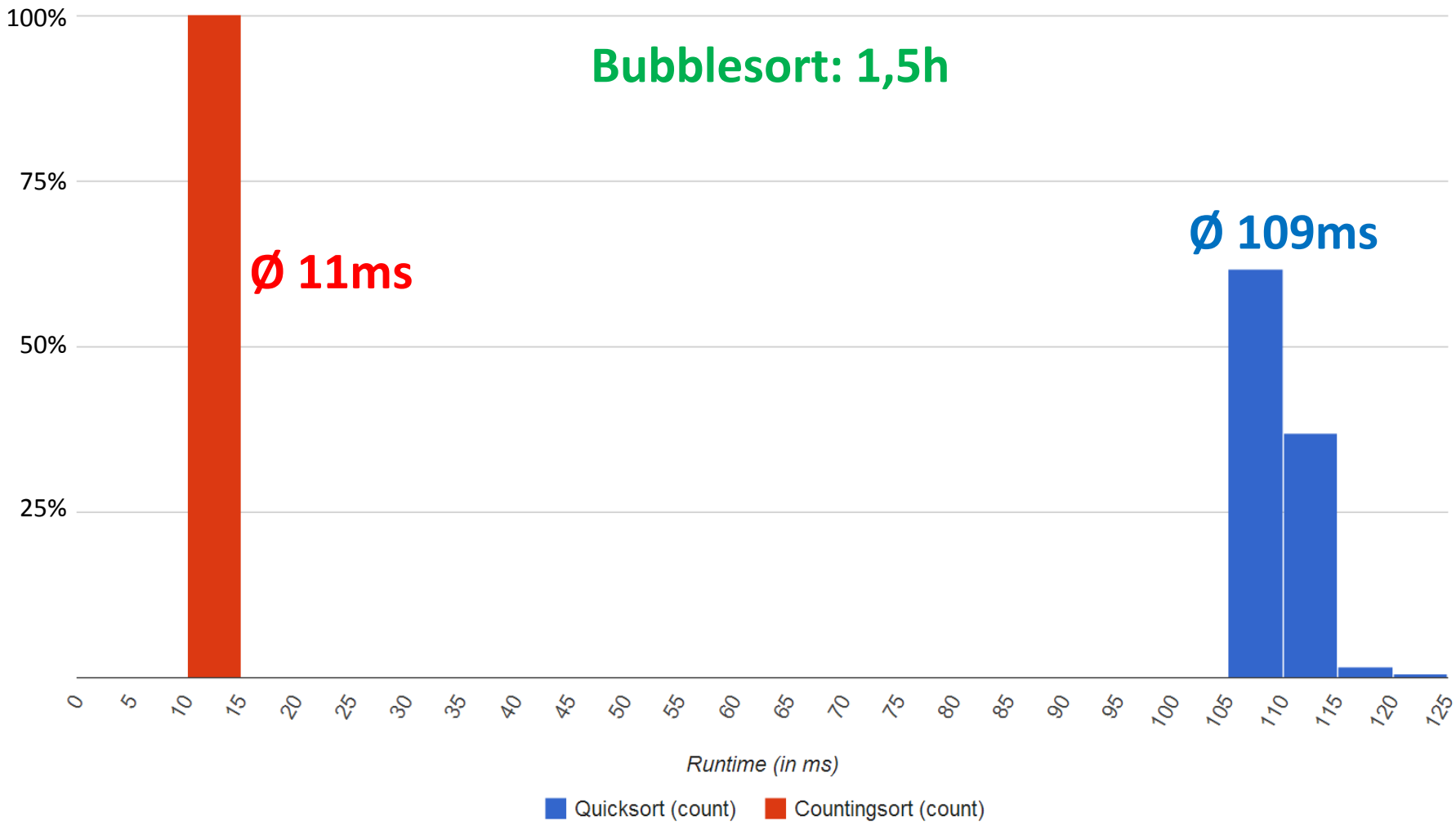
Aufgabe 2 - Sortieren

- Bubblesort
  - $\mathcal{O}(n^2)$
  - Laufzeit empirisch ($n = 1M$): 1,5h
- Quicksort
  - $\mathcal{O}(n \log n)$
  - Laufzeit empirisch ($n = 1M$): 109ms
- Countingsort
  - $\mathcal{O}(n + m)$
  - Laufzeit empirisch ($m \approx n = 1M$): 11ms

- Häufige Fehler:
  - Sortieren von leeren Arrays führt zu Exceptions

Aufgabe 2 – Sortieren

- Laufzeit empirisch ($m \approx n = 1M$)



Aufgabe 3 - Kartenspiel

- Comparable vs. Comparator
  - natürliche Sortierung vs. variable Sortierung (siehe auch `java.util.Collections.reverseOrder()`)
- Anpassung von Bubblesort/Quicksort auf das Kartenspiel ist simpel
- Countingsort[1]:
  - arbeitet nicht vergleichsbasiert, sondern zählt die Werte der Eingabe
  - setzt voraus, dass die Werte der Eingabe sich auf ein beschränktes Intervall abbilden lassen:
  - Benötigt:
    Mapping von Kartenwert (Kreuz 3) zu Sortierschlüssel (2)

1 - http://de.wikipedia.org/wiki/Countingsort

Zusatzaufgabe - Spürhunde

- Häufig nicht beachtet:
  „… **müssen** die gesuchten Gegenstände in den Boxen der beiden Hunde unterschiedlich sein …“

```
1  public class DogTraining {
2    public static int secondsToWait(int k, int[] b) {
3      if(k < 1  || k > 1000000 || b.length < 2 || b.length > 500000)
4        throw new IllegalArgumentException();
5
6      boolean[] found = new boolean[k];
7      for (int i = 0; i <= (b.length + 1) / 2; i++)
8        if (check(b[i], k, found) || check(b[b.length - i - 1], k, found))
9          return i + 1;
10     return -1;
11   }
12   public static boolean check(int val, int k, boolean[] found) {
13     if(val < 1 || val > 1000000) throw new IllegalArgumentException();
14     return val < k && val != k - val && (found[val] = true) && found[k - val];
15   }
16 }
17
18
19
20
```