Lösung der 6. Übung – Informatik I für VIW

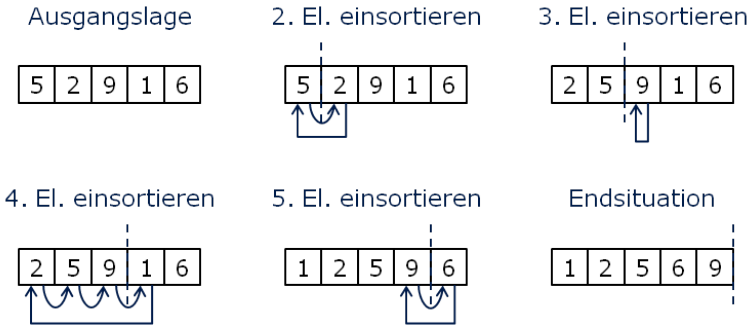
Fakultät Verkehrswissenschaften
Fachrichtung Verkehrsingenieurwesen

Zeitraum: 10.01. bis 21.01.2011 (WS 2010/11)

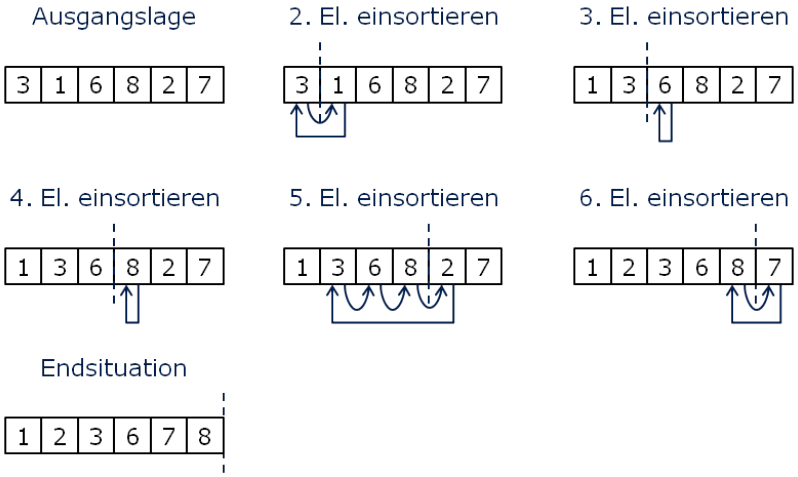
Aufgabe 1

(a)

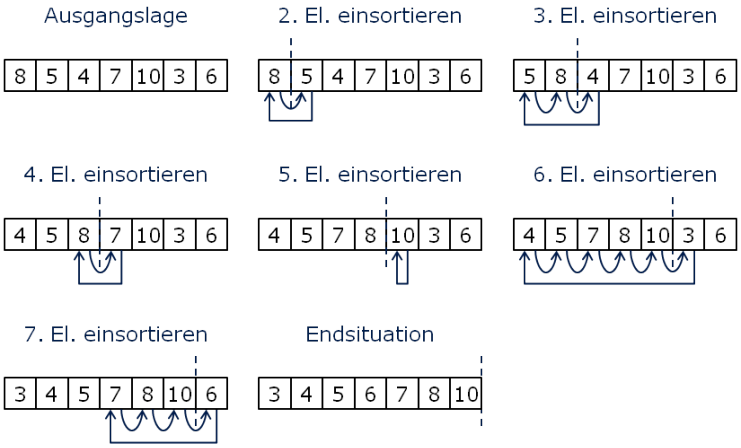
- Problem 1



- Problem 2



- Problem 3



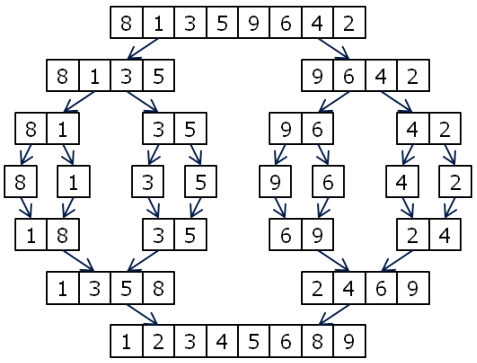
(b) Bei der Folge $3, 4, 5, 6, 1, 2$ sind 8 Verschiebungen nötig. Acht Verschiebungen sind auch bei der Folge $5, 6, 1, 2, 3, 4$ nötig. Es gibt noch viele weitere Beispiele.

Bei keiner Folge der Länge sechs sind 16 Verschiebungen erforderlich. Die maximale Anzahl an Verschiebungen für Folgen der Länge sechs (also der Worst-Case) ist 15 und wird zum Beispiel durch die Folge $6, 5, 4, 3, 2, 1$ erreicht.

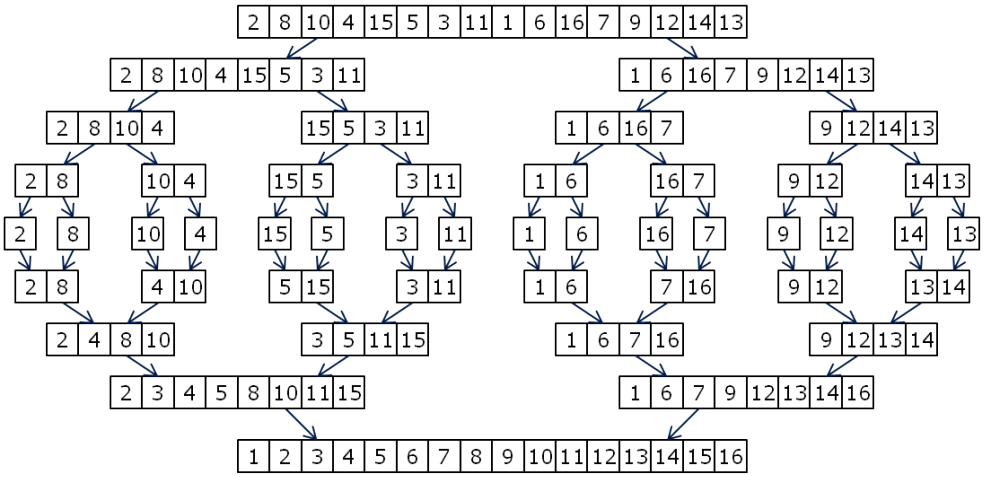
Aufgabe 2

(a)

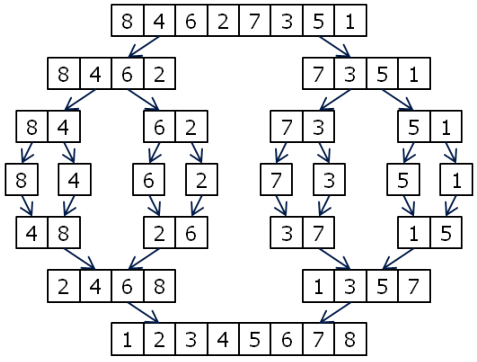
- Problem 1



- Problem 2



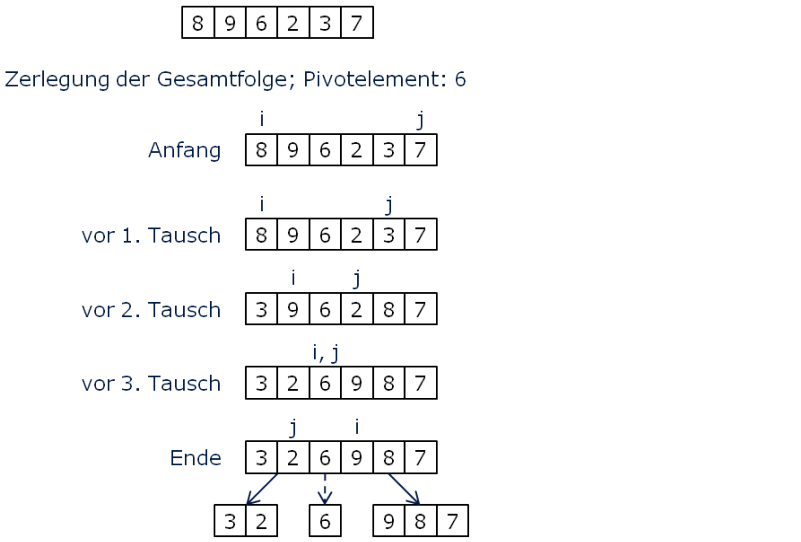
(b) Die Folge $8, 4, 6, 2, 7, 3, 5, 1$ ist die einzige Folge der Länge acht, die aus den Zahlen 1 bis 8 besteht und diese Eigenschaft hat.



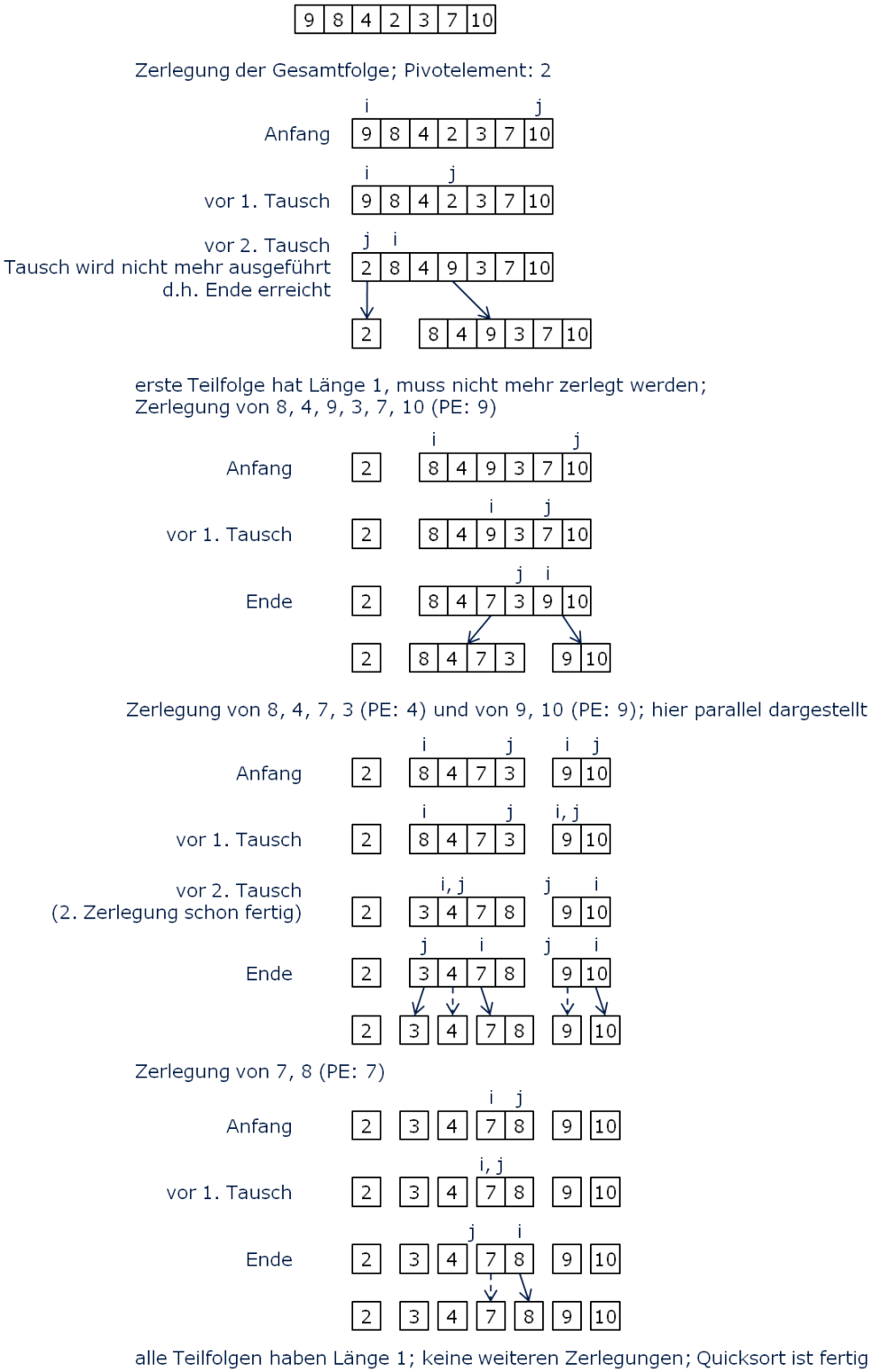
Aufgabe 3

(a)

- Problem 1

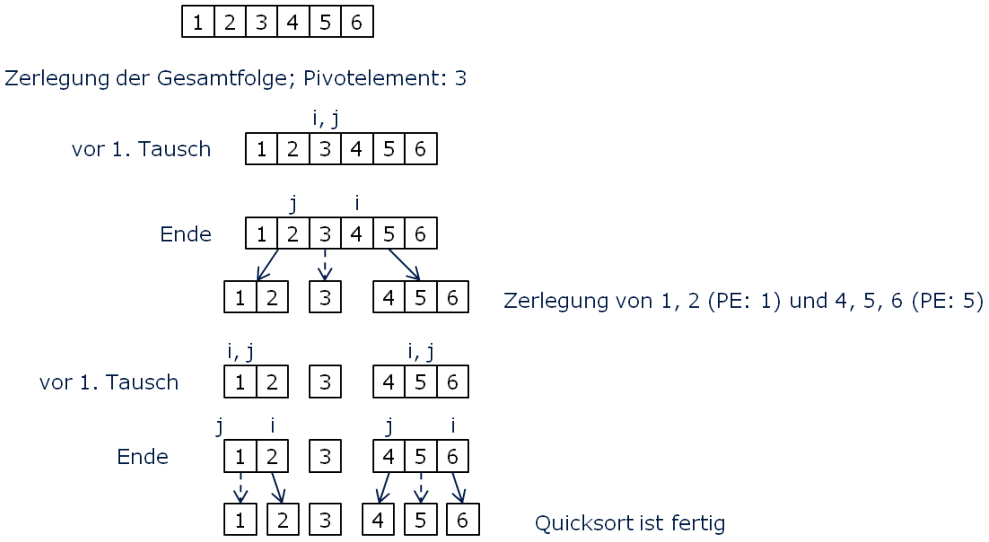


- Problem 2


9 8 4 2 3 7 10
Zerlegung der Gesamtfolge; Pivotelement: 2
Anfang
vor 1. Tausch
vor 2. Tausch
Tausch wird nicht mehr ausgeführt
d.h. Ende erreicht
erste Teilfolge hat Länge 1, muss nicht mehr zerlegt werden;
Zerlegung von 8, 4, 9, 3, 7, 10 (PE: 9)
Anfang
vor 1. Tausch
Ende
Zerlegung von 8, 4, 7, 3 (PE: 4) und von 9, 10 (PE: 9); hier parallel dargestellt
Anfang
vor 1. Tausch
vor 2. Tausch
(2. Zerlegung schon fertig)
Ende
Zerlegung von 7, 8 (PE: 7)
Anfang
vor 1. Tausch
Ende
alle Teilfolgen haben Länge 1; keine weiteren Zerlegungen; Quicksort ist fertig

(b) Die Eingabe $5, 3, 1, 2, 4, 6$ führt zu einem solchen Worst-Case.

Der Best-Case entsteht zum Beispiel, wenn die Eingabe bereits sortiert ist.



Aufgabe 4 (Zusatz)

Wir nehmen in dem folgenden Programm an, dass alle Zahlen in der Eingabe zwischen 0 und 99 liegen. Ansonsten weiß man noch nicht zum Zeitpunkt des Programmierens, wieviele Zähler man bereitstellen muss, man müsste dann mit dynamischer Speicherverwaltung arbeiten (siehe Vorlesung 12).

Die Zähler stellen wir durch ein Array der Länge 100 dar.

In der ersten Schleife initialisieren wir alle Zähler auf 0. In der zweiten Schleife durchlaufen wir die Eingabe, und für jedes Element der Eingabe (also `feld[i]`) inkrementieren wir den entsprechenden Zähler (also `zaehler[feld[i]]`).

In der letzten Schleife gehen wir die Zähler der Reihe nach durch und schreiben für jeden Zähler `zaehler[i]` genau m mal den Wert i in die Eingabeliste, wobei m der Wert des Zählers ist. Der Index k dient dazu, um anzuzeigen, wo der nächste Eintrag in die Eingabeliste zu schreiben ist.

```
void bucketsort(int feld[], int n) {
   int zaehler[100], i, j, k;
   for (i = 0; i < 100; i++) zaehler[i] = 0;
   for (i = 0; i < n; i++) zaehler[feld[i]]++;

   k = 0;
   for (i = 0; i < 100; i++) {
      for (j = 1; j <= zaehler[i]; j++) {
         feld[k] = i;
         k++;
      }
   }
}
```

Laufzeit Die erste Schleife hat eine konstante Laufzeit (unabhängig von n). Die zweite Schleife hat die Laufzeit $O(n)$. In der letzten Schleife werden die beiden innersten Befehle `feld[k] = i` und `k++` genau `n` Mal ausgeführt, denn die Summe der in den Zählern gespeicherten Werte muss `n` sein (für jedes Element der Eingabefolge wird genau einer der Zähler um eins erhöht). Damit ist die Gesamtlaufzeit $O(1) + O(n) + O(n) = O(n)$.

Bucketsort läuft also in linearer Zeit ab, ist demnach deutlich schneller als Quicksort und Mergesort. Allerdings kann man Bucketsort nur zum Sortieren von Zahlenfolgen benutzen, deren Werte beschränkt sind.