12. Übungsblatt - Informatik 1 - Lösungsbeispiele

Aufgabe 1 (Median der Mediane Algorithmus)

Die in der Vorlesung vorgestellte Rekurrenzgleichung für den Median der Mediane Algorithmus war $T(n) = T(\frac{n}{5}) + T(\frac{3n}{4}) + cn$. Wie in der Vorlesung gezeigt gilt: $T(n) = O(n)$

- Geben Sie eine etwas bessere Abschätzung für den Teil $T(\frac{3n}{4})$ an.
- Wenn statt 5er-Gruppen nur 4er-Gruppen gebildet werden, gilt dann immer noch $T(n) = O(n)$?
- Und bei 3er-Gruppen?

Lösungsvorschlag:
Wenn man sich das Hasse-Diagramm zum Algorithmus genauer ansieht, dann sind 3/5 der Hälfte der Werte kleiner bzw. größer als der Median der Mediane, also 3/10. Eine etwas präzisere Abschätzung ist $T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + cn$. Diese Abschätzung ist genauer, da $7/10 < 3/4$ gilt.

4er-Gruppen gehen auch, da dann $T(n) = T(n/4) + T(7n/10) + cn$ gilt und mit $1/4 + 7/10 = (5 + 14)/20 < 1$ immer noch $T(n) = O(n)$ ist.

3er-Gruppen gehen nicht, da dann die Faktoren größer als Eins werden (ein noch bessere Abschätzung der Rekurrenz ist nicht möglich).

Aufgabe 2 (Rekurrenzgleichung)

Betrachten Sie folgende Rekurrenzgleichung:

$$T(n) = T(\frac{n}{4}) + T(\frac{3n}{8}) + cn$$

Zeigen Sie das $T(n) = O(n)$ gilt.

Lösungsvorschlag:
Aus der Vorlesung ist bekannt dass $T(n) = T(an) + T(bn) + cn = O(n)$ gilt, wenn $a + b < 1$ ist. Da $1/4 + 5/8 = 7/8 < 1$ gilt, ist $T(n) = O(n)$.

Aufgabe 3 (Sortieren durch direktes Einfügen)

Das folgende Programm zeigt Sortieren durch direktes Einfügen.

```
public void sortieren(int [] a) {
  for (int i = 1; i < a.length; i++) {
    int t = a[i];
    for (int j = i; j > 0 && a[j - 1] > a[j]; j--) {
      a[j] = a[j - 1];
      a[j - 1] = t;
    }
  }
}
```

Die Anweisung j > 0 behandelt nur einen Sonderfall, der in der Praxis zusätzliche Rechenzeit benötigt. Überlegen Sie sich eine Vorverarbeitung, so dass diese Anweisung wegfallen kann. Geben Sie das entsprechend geänderte Programm an. Wird dadurch das Programm in der Praxis schneller im schlimmsten Fall? Wenn ja, warum?

Lösungsvorschlag:

Die Vorverarbeitung sucht das Minimum und bringt es an die erste Stelle im Feld: Zusätzliche Kosten $O(n)$. Im schlimmsten Fall werden $O(n^2)$ Vergleiche $j > 0$ eingespart. Dies rentiert sich schon bei kleinem n. Der Zeitgewinn ist allerdings nur im niedrigen einstelligen Prozentbereich. Im besten Fall wird das Programm etwas langsamer, da die $O(n)$ Vorverarbeitung normalerweise etwas aufwändiger sind, als die eingesparten $O(n)$ Vergleich $j > 0$.

```
public void sortieren(int [] a) {
  int minimumIndex = 0;
  for (int i = 1; i < a.length; i++) {
    if (a[i] < a[ minimumIndex]) {
      minimumIndex = i;
    }
  }
  int t = a[0];
  a[0] = a[minimumIndex];
  a[minimumIndex] = t;

  for (int i = 1; i < a.length; i++) {
    int t = a[i];
    for (int j = i; && a[j - 1] > a[j]; j--) {
      a[j] = a[j - 1];
      a[j - 1] = t;
    }
  }
}
```