Ludwig-Maximilians-Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme
Prof. Dr. Christian Böhm
Dr. Thomas Bernecker, Sebastian Goebl

München, 02.05.2013

Algorithmen und Datenstrukturen

SS 2013

Übungsblatt 3: O-Notation

Besprechung (**Achtung: Turnuswechsel wegen der Feiertage**): **13. - 17.05.2013**

Abgabe dieses Übungsblattes bis spätestens Monatg, 13.05.2013, 14:00 Uhr.

Hinweise zur Abgabe:
Geben Sie bitte Ihre gesammelten Lösungen zu diesem Übungsblatt in einer Datei `loesung03.zip` unter `https://uniworx.ifi.lmu.de` ab.

Aufgabe 3-1 *O-Notation* **5 Punkte**

Geben Sie an und begründen Sie kurz, ob die folgenden Aussagen wahr oder falsch sind.

(a) $3n^2 + 30n + 300 = \Omega(n^2)$

(b) $3n^2 + 30n + 300 = \Theta(n^2)$

(c) $\sqrt{n+1} = O(\sqrt{n})$

(d) $3^{n+1} = O(3^n)$

(e) $x^n = O(n)$

Aufgabe 3-2 *Maximale Fallhöhe einer Glaskugel* **4 Punkte**

Finden Sie die maximale Fallhöhe (in Stockwerken) einer Glaskugel. Dazu haben Sie Zugang zu einem n-stöckigen Hochhaus und können in jedem Stockwerk eine Glaskugel aus dem Fenster werfen um zu überprüfen, ob sie den Fall übersteht. Nehmen Sie dabei an, dass alle Glaskugeln die selbe maximale Fallhöhe haben. Bei einem Sturz aus größerer Höhe geht eine Glaskugel kaputt und kann nicht mehr verwendet werden. Stürze aus geringerer Höhe werden immer unbeschadet überstanden.

Beschreiben Sie Ihr Vorgehen, wenn Sie

- eine Glaskugel,
- zwei identische Glaskugeln,
- unendlich viele identische Glaskugeln

zur Verfügung haben und die Anzahl der Würfe minimieren möchten.

Schätzen Sie außerdem die Worst-Case-Komplexität Ihrer Algorithmen ab.

Aufgabe 3-3 *O-Notation 2* **4 Punkte**

Geben Sie an, für welche $x \in \mathbb{R}^+$ die folgende Gleichung gilt:

$$\sum_{i=0}^{n} x^i = O(n)$$

Beweisen Sie Ihr Ergebnis.

Aufgabe 3-4 *Maximum-Subarray* **10 Punkte**

Das Maximum-Subarray Problem besteht darin, in einem Array aus ganzen Zahlen der Länge n die **nicht leere** Teilfolge von Zahlen zu finden, deren Summe maximal ist.

Beispiel: Für die Eingabefolge $[x_0, ..., x_9]$

$$[12, -32, 38, 22, -27, 32, 17, -13, -10, 10]$$

ist die maximale Summe einer Teilfolge 82 für die Teilfolge $[x_2, ..., x_6]$.

Ein naiver Algorithmus zur Lösung dieses Problems ist der folgende: Berechne für jedes mögliche Start- und Endelement die dazwischenliegende Summe und merke Dir die größte Summe und die entsprechenden Start- und Endelemente. Als Java-Algorithmus ausgedrückt:

```
public void naiveSolution(Integer[] x) {
  int maxSum = 0;
  for (int u = 0; u < x.length; u++) {
    for (int o = 0; o < x.length; o++) {
      // bestimme die Summe der Elemente der Teilfolge von [x_u, ..., x_o]
      int sum = 0;
      for (int i = u; i <= o; i++) {
        sum = sum + x[i];
      }
      // vergleiche die gefundene Summe mit der bis jetzt maximalen
      maxSum = Math.max(maxSum, sum);
    }
  }
}
```

(a) Schätzen Sie die Zeitkomplexität des obigen Algorithmus ab und begründen Sie Ihre Angabe kurz.

(b) Geben Sie die Idee für einen Algorithmus an, der obiges Problem in linearer Komplexität löst und demonstrieren Sie diese am obigen Zahlenbeispiel.

(c) Implementieren Sie Ihre Lösungsidee aus Aufgabe (b) in Java. Das Programm soll dabei die Summe, das erste und das letzte Element der maximalen Teilfolge als Ergebnis ausgeben. Die Eingabe des Inputarrays als Konstante ist ausreichend.