Übungen zu Computergrundlagen WS 2017/2018

Übungsblatt 10: Algorithmen und Datenstrukturen

12. Januar 2018

Allgemeine Hinweise

- Abgabetermin für die Lösungen ist **Freitag, 19.01.2018, 11:00 Uhr**
- Schickt die Lösungen bitte per Email an Euren Tutor:
  - Montag 11:30 – 13:00: Julian Zeller (`julian.zeller@icp.uni-stuttgart.de`)
  - Montag 14:00 – 15:30: Miriam Kohagen (`mkohagen@icp.uni-stuttgart.de`)
  - Dienstag 14:00 – 15:30: Ingo Tischler (`itischler@icp.uni-stuttgart.de`)
  - Dienstag 15:45 – 17:15: Konrad Breitsprecher (`konrad@icp.uni-stuttgart.de`)
  - Donnerstag 09:45 – 11:15: Ashreya Jayaram (`ashreyaj@icp.uni-stuttgart.de`)

Aufgabe 10.1: Numerische Berechnung einer Fakultät (4 Punkte)

In dieser Aufgabe sollt ihr euch überlegen, wie man in einem Computerprogramm die Fakultät (englisch: "factorial") einer natürlichen Zahl berechnen kann. Die Fakultät einer Zahl $n \in \mathbb{N}$ ist definiert als

$$n! = \prod_{i=1}^{n} i = 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n \,. \tag{1}$$

Einen Spezialfall stellt die Fakultät der Zahl Null dar, sie ist definiert als $0! = 1$.

- **10.1.1** (2 Punkte) Schreibe eine Funktion `recursive_factorial(n)`, welche die Fakultät einer an sie übergebene Zahl `n` *rekursiv* berechnet. Die Funktion soll also ausnutzen, dass $n! = n \cdot (n-1)!$ gilt. Dabei muss keine tatsächlich existierende Programmiersprache verwendet werden; eine Art von Pseudocode, der den Programmfluss eindeutig darstellt, ist vollkommen ausreichend.

  Hinweis: Achte darauf, dass die Funktion auch mit den Parametern `n=0` und `n=1` zurechtkommt!

- **10.1.2** (2 Punkte) Schreibe eine Funktion `loop_factorial(n)`, welche die Fakultät von `n` mit Hilfe einer Schleife (also *nicht* rekursiv!) berechnet. Begründe, welche Art Schleife du verwendest!

Aufgabe 10.2: Skalierungsverhalten von Sortieralgorithmen auf unterschiedlichen Datenstrukturen (6 Punkte)

In der Vorlesung wurden verschiedene Algorithmen zur Sortierung von Daten besprochen. Diese Aufgabe behandelt die beiden wohl einfachsten Sortieralgorithmen, nämlich die *bubble sort* und *insertion sort* Algorithmen. Die zu sortierenden Datenpunkte sollen der Einfachheit halber als Menge von n beliebigen natürlichen Zahlen angenommen werden, die dann in aufsteigender Reihenfolge sortiert werden sollen. Jede Zahl soll in der Menge nur ein einziges Mal enthalten sein.

- **10.2.1** (2 Punkte) Wie sieht die Reihenfolge der *vor* der Sortierung vorliegenden Daten für den jeweiligen Algorithmus im besten Fall aus, wie im schlechtesten Fall?
- **10.2.2** (2 Punkte)
  - Wie viele Vertauschungsoperationen und wie viele Vergleichsoperationen benötigt der bubble sort Algorithmus im schlechtesten Fall, um n Datenpunkte zu sortieren?
  - Wie viele der jeweiligen Operationen werden im besten Fall benötigt?
  - Wie skaliert der bubble sort Algorithmus demnach in den beiden Fällen?
- **10.2.3** (2 Punkte) Der insertion sort Algorithmus erfordert, dass einzelne Datenpunkte aus der zu sortierenden Datenstruktur entfernt und an anderer Stelle wieder eingefügt werden können, ohne die Integrität der Struktur dabei zu beeinträchtigen.
  - Wie viele Leseoperationen und wie viele Schreiboperationen werden benötigt, um das i-te Element eines *Arrays* mit Gesamtlänge n zu entfernen, sodass die Länge des Arrays danach $n - 1$ beträgt?
  - Wie viele der jeweiligen Operationen werden benötigt, um das entfernte Element an der j-ten Stelle wieder einzufügen?
  - Wie verhält sich dies bei einer doppelt verketteten Liste?
  - Welche Datenstruktur (Array oder Liste) ist demnach im Allgemeinen für den insertion sort Algorithmus zu bevorzugen?