Ludwig-Maximilians-Universität München
Institut für Informatik
PD. Dr. Peer Kröger
Johannes Niedermayer, Andreas Züfle

München, 11.12.2012

Einführung in die Programmierung

WS 2012/13

Tutorium 1: Sortieralgorithmen

Besprechung: 20.12.2012

Ende der Abgabefrist: Keine Abgabe.

Hinweise:

- Im Tutorium besprochene Aufgaben sind prüfungsrelevant für Nebenfachstudierende mit 60 ECTS.
- Eine Abgabe und Korrektur des Tutoriums findet **nicht** statt.

Aufgabe 1-1 *Komplexität von Sortieralgorithmen* **0 Punkte**

- **SimpleSort – Sortieren durch einfaches Vertauschen**
  Gegeben ist ein (unsortiertes) Array a der Länge n. Jedes Element an Position i von a (mit $i = 0, \ldots, n-1$) wird nun nacheinander mit allen Elementen der Positionen $j = i+1, i+2, \ldots, n-1$ verglichen. Falls das Element an Position j kleiner ist als das Element an Position i, werden die beiden Elemente vertauscht.

  Implementieren Sie in einer Klasse `SimpleSort` eine Methode
  `public static void sort(int[] a),`
  die ein übergebenes Array vom Typ `int[]` nach obigem Verfahren sortiert.

- **CountSort – Sortieren durch Abzählen**
  Zum Sortieren eines Arrays a der Länge n kann man sich folgende Beobachtung zu Nutze machen: In einem sortierten Array kann die Position eines Elements durch Abzählen aller kleineren Elemente des Arrays ermittelt werden. Gibt es in einem Array $i-1$ Elemente, die kleiner sind als ein Element e, so steht dieses Element e an Position i des sortierten Arrays. Trotz dieser simplen Idee erfordert die Möglichkeit des Vorkommens gleicher Elemente ein trickreiches Vorgehen.

  Implementieren Sie in einer Klasse `CountSort` eine Methode
  `public static void sort(int[] a),`
  die ein übergebenes Array vom Typ `int[]` nach obigem Prinzip sortiert.

- **SelectionSort – Sortieren durch direktes Auswählen**
  Gegeben ist ein (unsortiertes) Array a der Länge n. Für jede Position $i = 0, \ldots, n-2$ von a wird nun nacheinander das kleinste Element im Teilarray $a_i, \ldots, a_{n-1}$ gesucht und mit dem Element an Position i vertauscht.

  Implementieren Sie in einer Klasse `SelectionSort` eine Methode
  `public static void sort(int[] a),`
  die ein übergebenes Array vom Typ `int[]` nach obigem Verfahren sortiert.

- **BubbleSort – Sortieren durch direktes Austauschen**
  Gegeben ist ein (unsortiertes) Array a der Länge n. Für $i = 0, \ldots, n-1$ werden für $j = n-1, \ldots, i+1$ paarweise die Elemente mit den Positionen $j-1$ und j miteinander verglichen. Falls das Element an Position j kleiner ist als das Element an Position $j-1$, werden die Elemente vertauscht.

  Implementieren Sie in einer Klasse `BubbleSort` eine Methode
  `public static void` `sort(`**`int`**`[] a)`,
  die ein übergebenes Array vom Typ **`int`**`[]` nach obigem Verfahren sortiert.

Die Klasse `Laufzeit` ermöglicht eine empirische Untersuchung des Laufzeitverhaltens der implementierten Sortierverfahren. Sie können für die Konstanten `START`, `STOP` und `SCHRITT` auch mit anderen Werten experimentieren. Der Algorithmus QuickSort ist in bereits in Java integriert. Was beobachten Sie?

Die Klasse `Laufzeit` finden Sie auf der Webseite zur Vorlesung.

Aufgabe 1-2 *Suchverfahren* **0 Punkte**

Gegeben ein sortiertes Integer-Array.

(a) Implementieren Sie in einer Klasse `LineareSuche` eine Funktion `public static int suche(int[] array, int element)`, die das Element mit dem Wert `element` sucht und im Erfolgsfall den entsprechenden Index im Array zurückgibt. Die Suche soll so implementiert sein, dass sämtliche Elemente des Arrays in aufsteigender Reihenfolge betrachtet werden. Die Suche kann abgebrochen werden, sobald das gesuchte Element gefunden wurde. Sollte das Element nicht enthalten sein, geben Sie -1 zurück.

(b) Implementieren Sie in einer Klasse `BinaereSuche` eine Funktion `public static int suche(int[] array, int element)`, die das Element mit dem Wert `element` sucht und im Erfolgsfall den entsprechenden Index im Array zurückgibt. Die binäre Suche wurde bereits in der Vorlesung besprochen; die Implementierung kann sich an der Vorlesung orientieren. Sollte das Element nicht enthalten sein, geben Sie -1 zurück.

(c) Die Klasse `LaufzeitSuche` ermöglicht eine empirische Untersuchung des Laufzeitverhaltens der implementierten Suchverfahren; sie kann auf der Vorlesungswebsite heruntergeladen werden. Sie können für die Konstanten `START`, `STOP` und `SCHRITT` auch mit anderen Werten experimentieren. Was beobachten Sie?