Inf ProgTech Übung: Sortierverfahren

Klaus Kusche

Wir wollen alle wesentlichen *Sortierverfahren grafisch veranschaulichen*
(diese Übung benötigt daher eine Entwicklungsumgebung mit wxWidgets, z.B. CodeLite, oder unsere Linux-Umgebung) und betreffend Komplexität statistisch *vergleichen*.

Ich stelle dazu ein Rahmenprogramm zur Verfügung, das sich um die Datendarstellung, das GUI, das Mitzählen der Operationen usw. kümmert.

In diesem Programm *fehlen die eigentlichen Sortier-Funktionen*.
Implementiere diese (siehe Kommentare im Source).

Hinweise:

- Als *Grundoperationen* zum Zugriff auf die zu sortierenden Daten stehen dir *nur* **`Comp(i, j)`** (*Vergleiche* das Element Index **`i`** mit dem Element Index **`j`**, Ergebnis wie bei **`strcmp <0`**, **`==0`**, **`>0`**) und **`Swap(i, j)`** (*Vertausche* das Element Index **`i`** mit dem Element Index **`j`**) zur Verfügung.

  *Du darfst **keine** einzelnen Zuweisungen von Arrayelementen machen,*
  keine Arrayelemente temporär außerhalb des Arrays speichern,
  und keine Vergleiche eines Array-Elementes mit einem nicht-Array-Element machen.

- Der Index der Elemente im Array geht von **`0`** bis **`(arrSize - 1)`** .

  Achtung:
  Der Heapsort verwendet in seiner Logik traditionell Indices von **`1`** bis **`arrSize`**!

Anmerkungen:

- Der **Heapsort** ist *am wenigsten wichtig* und nur für die,
  die zusätzliche Übungs-Herausforderungen suchen.

  Quicksort, Shellsort und ein oder zwei einfache Verfahren wären wichtig.

- Du darfst und sollst dir auf meinen Webseiten und am Internet
  die Algorithmen zusammensuchen.

  Beachte aber, dass gerade beim **Quicksort** *nicht jeder weit verbreitete Algorithmus zur Partitionierung* des Arrays mit unseren eingeschränkten Zugriffsfunktionen (ohne temporäre Elemente außerhalb des Arrays) realisierbar ist.
  Du musst einen suchen, der das *Pivot-Element nicht außerhalb* des Arrays ablegt!

  Tipp: Es gibt z.B. Partitionierungs-Algorithmen,
  - die das Pivot-Element zuerst ganz rechts im Array speichern
    (bzw. mit dem Element ganz rechts vertauschen),
  - dann die restlichen Elemente davor (nur durch Vertauschen) partitionieren
  - und schließlich das Pivot-Element an die Grenze zwischen "kleinem" und "großem" Teil zurücktauschen.

  Der Partitionierungs-Algorithmus von meinem Sortier-Beispielcode ist ungeeignet (weil bei ihm das Pivot-Element sowohl in einer Variable gespeichert wird als auch im zu partitionierenden Bereich selbst verbleiben muss, damit alle Schleifen terminieren), der auf Wikipedia lässt sich entsprechend umbauen.

- Generell hat man bei **Quicksort** *mehrere Möglichkeiten*:

  Auswahl des *Partitionswertes*:
  - Element in der *Mitte* des Arrays
  - Wertmäßig mittleres Element aus *erstem, mittlerem und letztem* Array-Element (best of 3)

  Partitionsalgorithmus:
  - Algorithmus mit *zwei gegenläufigen Indices*, die sich in der Mitte treffen (3 Schleifen, *schwieriger*, aber deutlich *effizienter*)
  - Algorithmus mit *zwei Indices von links nach rechts* (1 Schleife, *einfacher*, aber macht meist sehr *viel mehr Vertauschungen*)

  Für beides hat der Prototyp meiner Quicksort-Funktion einen *Parameter*:
  - **`simple`** schaltet auf den *einfacheren* Partitionsalgorithmus um
  - **`opt`** schaltet den *"best of 3"*-Partitionswert ein

  Der *Normalfall* ist bei mir: Mittleres Array-Element, besserer Partitionsalgorithmus. Es wäre schön, wenn Du diesen Fall zum Laufen bringst.

- Außerdem hat mein **Quicksort** einen *Parameter für eine Verzögerung* pro rekursivem Aufruf, damit man den Ablauf in Ruhe verfolgen kann.

- Auch beim **Bubblesort** freue ich mich, wenn du *Optimierungen* einbaust (z.B. *Position der letzten Vertauschung* merken).