23. November 2014

Team: 02, Daniel Hofmeister, Alex Mantel

Aufgabenaufteilung:

1. Daniel Hofmeister: Skizze, q_sort
2. Alex Mantel:

Quellenangaben:

- Praktikumsaufgabe 3

Bearbeitungszeitraum:

- Daniel Hofmeister: 8h
- Alex Mantel:

Aktueller Stand: Komplett fertig

Fragen an: **alex.mantel@haw-hamburg.de**
daniel.hofmeister@haw-hamburg.de

Änderungen der Skizze:

Aufgabe 4: Implementierung des Quicksorts

2. quicksort(Array,leftPivot)

Ein Array aus Zahlen soll mit diesem Aufruf sortiert werden, aufsteigend von links nach rechts. Der verwendet Algorithmus soll hierbei der quickSort und die Lösung rekursiv gestaltet sein.
Als Pivot des Algorithmus soll bei diesem einfachen Algorithmus das erste (linkste) Element des Arrays verwendet werden.
Der Algorithmus funktioniert, indem alle Elemente des ursprünglichen Arrays mit dem Pivot verglichen werden und in zwei neue Arrays namens „Left“ und „Right“ gesteckt werden, je nachdem ob das grade angeschaute Element kleiner (→ Left) oder größer (→ Rechts) ist.
„Left“ und „Right“ werden anschließend rekursiv in den Aufruf „quicksort“ geworfen um nach dem gleichem Prinzip sie aufzuteilen.
Ist die Größe eines Arrays 11 oder wengier, soll mit dem vorhandenem Suchalgorithmus „selectionsort“ genau diese Elemente sortiert und zurückgegeben werden.
Das „leftPivot“ im Aufruf gibt an, wie der Pivot gewählt wird um den gleichen Aufruf für verschiedene Strategien der Pivotauswahl zu erlauben.

23. November 2014

3. quicksort(Array,randomPivot)

Wie bei (2.) soll mit dem quicksort Verfahren das Array sortiert werden und auch hier soll auf den selectionsort umgeschaltet werden, wenn ein Teilarray nur 11 oder weniger Elemente enthält.
Die Strategie der Pivotwahl ist hier jedoch ein zufälliges Element aus dem Array nehmen. Der Nachteil ist, dass die Bestimmung des Pivots etwas mehr Zeit kostet als einfach das erste Element zu nehmen, der Algorithmus wird jedoch wesentlich effektiver im Fall von vorsortierten Arrays.
Würde man bei einem sortierten Array das erste Element nehmen, so wäre die Komplexität maximal für den Algorithmus $O(n^2)$, bei einem hier zufällig bestimmten Pivot wäre sie nurnoch $O(n \log n)$.

Für beide Algorithmen (2+3) gilt:
Das jeweils zu sortierende Array soll aus einer Datei (zahlen.dat) ausgelesen werden, das nach Anwendung des Algorithmus sortierte Array soll selbst in eine Datei (sortiert.dat) ausgegeben werden.

Zusätzlich soll eine Zeitmessung eingebaut werden, um die Dauer einer einzelnen Sortierung zu bestimmen. Eine zweite Kopie des Codes soll am Ende erstellt werden und diese mit Zählern für Vergleiche und Verschiebung der Elemente versehen.

Abschließend:
Die Algorithmen sind 100-mal auszuführen, wenn möglich mit jeweils unterschiedlichen Zahlen: 80-mal Zufallszahlen und jeweils 10-mal "best case" bzw. "worst case" Zahlen. Aus den Zeitmessungen ist dann per Mittelwert anzugeben:

1. Anzahl eingelesener Elemente
2. Name des Algorithmus
3. Benötigte Zeit, sowie maximal und minimal benötigte Zeit.
4. Anzahl Vergleiche, sowie maximale Anzahl und minimale Anzahl an Vergleichen.
5. Anzahl Verschiebungen, sowie maximale Anzahl und minimale Anzahl an Verschiebungen.

Zusätzlich:

Es wird ebenso eine Ausgabe in .csv erzeugt für jeden „batch“ der 100 Aufrufe mit der man einen Sortieralgorithmus statistisch bewerten könnte. Weiterhin wurde ein Progression-Zähler eingebaut, um den Anwender des Haupt-aufrufes bei großen Sortieraufgaben zu informieren wie viele Iterationen bereits durchlaufen wurden.