4 Praktikum 4: Sortierverfahren / Hashing

Ihre Vorgesetzten bei der Firma "Data Fuse Inc." sind begeistert von Ihren Fähigkeiten! Da die Verarbeitungsgeschwindigkeit der enormen Datenmengen weiter optimiert werden muss wurden Sie beauftragt, ein Framework zur Messung von Laufzeiten zu entwickeln. Mit Hilfe dieses Programms sollen Sie anschließend die Ausführungsdauer verschiedener Sortieralgorithmen in Abhängigkeit einer Problemgröße n untersuchen und auswerten. Da exakte Zeitmessungen in C/C++ nicht trivial sind, brauchen Sie dies nicht selber zu implementieren. Stattdessen sollen Sie OpenMP nutzen, um die Zeiten zu messen (Vorteil: einheitlich und einfache Nutzung). Im zweiten Teil der Aufgabe soll eine einfache Hashtabelle implementiert werden.

4.1 Teilaufgabe 4.1

Für alle Aufgaben gilt, dass Sie hierzu die Vorlage aus ILIAS benutzen können, in der bereits der Programmrumpf sowie ein Benchmarkaufruf für einen Sortieralgorithmus exemplarisch implementiert ist. Sie dürfen aber auch gerne eine komplett eigene Lösung erstellen, bzw die Vorlage Ihren Wünschen gemäß anpassen.

1. Vervollständigen Sie die Sortieralgorithmenbibliothek, bestehend aus der Header-Datei *sorting.h* und implementieren Sie die folgenden Algorithmen in der zugehörigen cpp-Datei *sorting.cpp* sowie im eigenen Namespace *sorting*:
   - Heapsort
   - Mergesort
   - Quicksort
   - Shellsort mit der **Hibbard Folge** ($H_i = 2H_{i-1} + 1$)
2. Erstellen bzw. vervollständigen Sie das Hauptprogramm. Im Hauptprogramm sollen die zu messenden Sortieralgorithmen mit einer entsprechenden Problemgröße n aufgerufen und die Ergebnisse der Zeitenmessungen in Textdateien geschrieben werden.
3. Messen Sie anschließend die Ausführungszeiten in Abhängigkeit der Problemgröße n für:
   - Heapsort, $n = 1000 : 1000 : 1000000$
   - Mergesort, $n = 1000 : 1000 : 1000000$
   - Quicksort, $n = 1000 : 1000 : 1000000$
   - Shellsort, $n = 1000 : 1000 : 1000000$

Wobei $n = 1000 : 1000 : 1000000$ z.B. bedeutet, dass Sie bei der Problemgröße $n = 1000$ beginnen sollen und die Problemgröße in jedem Schritt um 1000 erhöhen bis Sie bei 1000000 angekommen sind. Nach jedem Schritt wird die Ausführungszeit für diese Problemgröße in eine Textdatei geschrieben. Messen Sie **nicht** die Gesamtlaufzeit! Initialisieren Sie Ihre Datenstrukturen vor jeder Messung neu mit Zufallszahlen (Integer).

4. Stellen Sie ihre Messergebnisse unter Zuhilfennahme von MATLAB, Octave oder GNUPLOT grafisch dar (Beispiele: siehe 4.3.5)! Entsprechen die Messergebnisse den Erwartungen (z.B. bzgl. O-Notation)? Achten Sie bei den Plots auf aussagekräftige Achsenbeschriftungen und eine vernünftige Legende!

5. Beachten Sie unbedingt die Lösungshinweise (4.3) und **planen Sie genügend Zeit für die Messungen ein**!

6. Alle Beispiele (Textausgaben, Codevorlagen, Plots,...) dienen der Illustration und dürfen gerne entsprechend Ihren eigenen Vorstellungen angepasst werden.

4.2 Teilaufgabe 4.2

Implementieren Sie eine Hash-Tabelle als Array (ohne Re-Hashing), in dem Sie die Vorlage `hashtable.h` bzw. `hashtable.cpp` vervollständigen. Implementieren Sie folgende Features:

1. Konstruktor:
   Beim Erzeugen einer Klasseninstanz soll ein entsprechend dem übergebenen Parameter dimensionierter `vector<int>` dynamisch auf dem Heap alloziert werden. Initial erhalten alle Werte des Vektors den Wert -1. Die Größe der Hashtabelle wird mit dem übergebenen Wert initialisiert, der Kollisionszähler und die Anzahl der gespeicherten Elemente sollten mit 0 initialisiert werden.

2. Destruktor:
   Stellen Sie sicher, dass aller zur Laufzeit dynamisch allozierter Speicher bei Zerstörung des Objektes wieder freigegen wird.

3. Berechnung des Hashindex:
   Implementieren Sie die Methode `HashTable::hashValue(int item)`, die den Hash-Index berechnet. Übergeben Sie der Methode den Schlüssel. Tritt eine Kollision auf, so soll der Kollisionszähler erhöht werden. Zur Kollisionsvermeidung soll quadratisches Sondieren als Strategie verwendet werden (N ist die Größe der Hashtabelle):

$$h_i(x) = (h(x) + i * i)\%N) \tag{1}$$

4. Einfügen in die Hashtabelle:
   Implementieren Sie die Funktion `HashTable::insert(int item)` zum Einfügen von Elementen. Bevor ein Item in die Tabelle eingefügt werden kann ist zunächst sein Hashindex zu berechnen. Bei erfolgreichem Einfügen soll der Zähler für die Anzahl der Elemente erhöht werden.

5. Nachdem die Unittests erfolgreich durchgelaufen sind, erzeugen Sie in Ihrem Hauptprogramm eine Hashtabelle der Größe 1000 und fügen Sie automatisch 200 Zufallszahlen ein, die im Wertebereich von 1000 bis 1500 liegen und geben Sie die Anzahl der Kollisionen auf der Konsole aus.

4.3 Lösungshinweise

4.3.1 Allgemeine Hinweise zur Zeitmessung

- Kompilieren Sie Ihr Projekt vor der Messung unbedingt im *RELEASE* Modus und verwenden Sie das Compilerflag *-O3* um eine maximale Performance zu erhalten.
- Deaktivieren Sie alle unnötigen Konsolenausgaben für die Messungen, da diese sehr viel Zeit kosten.
- Beenden Sie alle anderen Anwendungen (Browser, EMail-Client, Antivirus, etc.....), da diese das Ergebnis ebenfalls drastisch verfälschen können!
- Achten Sie ebenfalls darauf, dass Sie nur die reine Sortier-/Ausführungszeit messen, und nicht zB das Erzeugen der Zufallszahlen mitmessen

4.3.2 OpenMP Compiler-Einstellungen

Damit Sie OpenMP zur Zeitmessung nutzen können sind entsprechende Compilereinstellungen erforderlich. Falls Ihre IDE nicht hier gelistet ist müssen Sie die für Sie erforderlichen Einstellungen per Selbstrecherche herausfinden.

Linux g++ Aktivierung von OpenMP unter Linux durch Compilerflag **-fopenmp** und linken gegen OpenMP Library **-lgomp**. Beispielaufruf für g++:

```
g++ main.cpp sorting.cpp hashtable.cpp unit_tests.cpp -o praktikum4 -fopenmp -lgomp
```

Windows Visual Studio Aktivierung von OpenMP unter Visual Studio:
https://msdn.microsoft.com/de-de/library/fw509c3b.aspx

```
Öffnen Sie das Dialogfeld Eigenschaftenseiten des Projekts.
Erweitern Sie den Knoten Konfigurationseigenschaften.
Erweitern Sie den Knoten C/C++.
Wählen Sie die Eigenschaftenseite Sprache aus.
Ändern Sie die Eigenschaft OpenMP-Unterstützung.
```

Windows DevC++ Aktivierung von OpenMP unter DevC++:

```
Tools -> Compiler Options
Add the following commands when calling the compiler: -fopenmp
Add the following commands when calling the linker: -lgomp
```

4.3.3 Format der TXT-Dateien

Erzeugen Sie für jeden gemessenen Algorithmus eine eigene Textdatei. Die Messungen sollten tabulatorgetrennt spaltenweise abgespeichert werden, damit sie möglichst einfach geplottet werden können. Ein Auszug aus der Datei "quicksort.txt"könnte dann beispielsweise wie folgt aussehen:

- 1.Spalte: Problemgröße n
- 2.Spalte: Berechnungsdauer in s

Beispiel:

```
...
986000   6.3498632997e-02
987000   6.3852430001e-02
988000   6.3209023996e-02
...
```

4.3.4 Beispiele zum Plotten mit MATLAB / GNUPLOT / Octave

- MATLAB
  Erzeugen Sie im selben Ordner, indem sich Ihre Messungen befinden, eine M-Skript-Datei mit einem beliebigen Namen, z.B. *make_plots.m*:

```
clear;clc;close all;

fid=fopen('quicksort.txt');
data=textscan(fid,'%d %f');
fclose(fid);
x=data{1};
quicksort_y=data{2};

fid=fopen('mergesort.txt');
data=textscan(fid,'%d %f')
fclose(fid);
mergesort_y=data{2};

fid=fopen('heapsort.txt');
data=textscan(fid,'%d %f');
fclose(fid);
heapsort_y=data{2};

fid=fopen('shellsort.txt');
```

```
data=textscan(fid,'%d %f');
fclose(fid);
shellsort_y=data{2};


figure;
title('sorting algorithms');
xlabel('n [-]');
ylabel('t [s]');
hold on;
plot(x,quicksort_y);
plot(x,mergesort_y);
plot(x,heapsort_y);
plot(x,shellsort_y);
legend('quicksort','mergesort','heapsort','shellsort','Location','northwest');
hold off;
```

Führen Sie das Skript anschließend aus:

```
>> make_plots
```

- GNUPLOT
  Erzeugen Sie im selben Ordner, indem sich Ihre Messungen befinden, eine Datei mit einem beliebigen Namen, z.B. *plots.gnu*:

```
reset
set autoscale x
set autoscale y
set xlabel "n [-]"
set ylabel "t [s]"
set key top left

plot \
"quicksort.txt" with linespoints title 'Quicksort',\
"mergesort.txt" with linespoints title 'Mergesort',\
"shellsort.txt" with linespoints title 'Shellsort',\
"heapsort.txt" with linespoints title 'Heapsort',\
```

Starten Sie nun Gnuplot, wechseln Sie in das korrekte Verzeichnis, und fuehren Sie das Skript wie folgt aus:

```
$ cd 'pfad-zum-gnuplot-skript'
$ load "plots.gnu"
```

Weiterführende Befehle zu GNUPLOT findet man z.B. hier:
http://gnuplot.sourceforge.net/docs_4.0/gpcard.pdf

4.3.5 Beispielplots

Die Plots sollten, natürlich in Abhängigkeit der verwendeten CPU, in etwa so aussehen (in den Abbildungen wurden die Legenden anonymisiert um die Ergebnisse nicht vorweg zu nehmen):

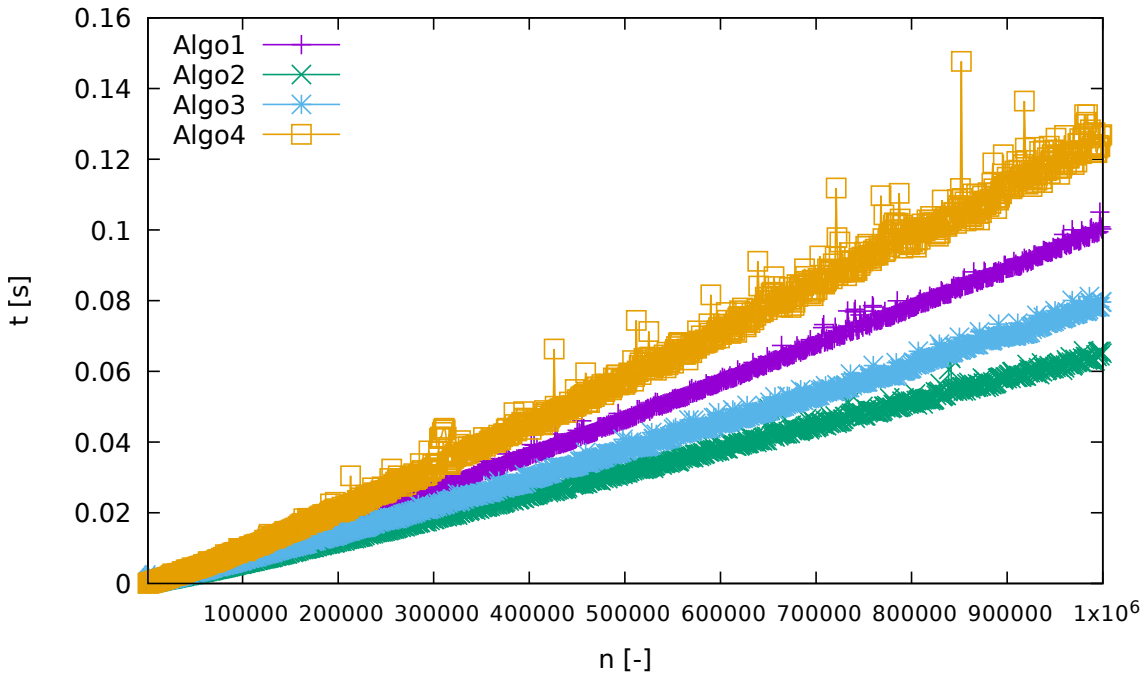

Abbildung 1: Laufzeitvergleich der Sortieralgorithmen