# 2 Praktikum 2: Sortierverfahren / Hashing

## 2.1 Teilaufgabe 1 Sortieren

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 sind. Sie dürfen aber auch gerne eine komplett eigene Lösung erstellen, bzw. die Vorlage Ihren Wünschen gemäß anpassen, solange Ihre Syntax mit den Test-Cases übereinstimmt.

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*:

   - Insertion Sort
   - Heapsort
   - Mergesort
     Verwenden Sie hier bitte für die Merge-Methode den Algorithmus aus der Vorlesung
   - Quicksort
   - Shellsort mit der **Hibbard Folge** ($H_i = 2H_{i-1} + 1$ und $H_1 = 1$)
   - Shellsort mit der **Abstandsfolge** ($H_i = 2^i$)
   - CountingSort
   - RadixSort

2. Messen Sie anschließend die Ausführungszeiten in Abhängigkeit der Problemgröße $n$ für InsertionSort, Heapsort, Mergesort, Quicksort, Shellsort, CountingSort und RadixSort.

   Nutzen Sie dafür die vorgegebene Funktion `benchmark_all`. Es wird bei der Problemgröße $n = 1000$ begonnen und in jedem Schritt die Problemgröße um 10000 erhöht, bis Sie bei 1000000 angekommen ist. In jedem Schritt werden 10 zufällige Eingaben abhängig der Problemgröße generiert, von allen Algorithmen sortiert und die mittlere Laufzeit der 10 Sortierfolgen für jeden Algorithmus berechnet. Nach jedem Schritt wird die mittlere Ausführungszeit für diese Problemgröße in eine Textdatei geschrieben.

3. Stellen Sie ihre Messergebnisse unter Zuhilfenahme von MATLAB, Octave oder GNUPLOT grafisch dar. Entsprechen die Messergebnisse den Erwartungen (z.B. bzgl. O-Notation)? Achten Sie bei den Plots auf aussagekräftige Achsenbeschriftungen und eine vernünftige Legende. Integrieren Sie eine mat. Fkt., die ihre Laufzeit möglichst gut approximiert.

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

Hinweise zur Zeitmessung finden Sie ab 2.3.

## 2.2 Teilaufgabe 2

Implementieren Sie einen einfachen lokalen Passwortmanager, der Webseiten-Zugänge speichert. Dabei soll eine selbst entwickelte Hash-Tabelle verwendet werden, um die Einträge zu speichern.
Entwickeln Sie die folgenden Funktionalitäten der Klasse MyHashTable:

- `rehash`
  Verdoppelung der Tabellengröße auf die nächste Primzahl. Rehashing aller bestehenden Elemente.
- `insert`
  - Prüfung, ob der Belegungsfaktor `maxLoadFactor` überschritten ist. Falls ja, wird ein Rehashing durchgeführt.
  - Berechnung des Hash-Index $x$ aus dem Website-Namen mit Hilfe der Funktion `hash(int)`.
  - Einfügen in die Hash-Tabelle unter Berücksichtigung der gewählten Sondierungs-Methode
    Lineares Sondieren:
    $$h_i(x) = (x + i)\%M \tag{1}$$
    Quadratisches Sondieren:
    $$h_i(x) = (x + i^2)\%M \tag{2}$$
    Doppeltes Hashing:
    $$h_i(x) = (x + i(R - x\%R))\%M \tag{3}$$
- `login`
  Prüfung, ob zu den 3 vom Nutzer eingegebenen Parametern (`site, user, hashed`) ein zugehöriger Eintrag in der Hashtabelle existiert.
- `listEntries`
  Ausgabe aller gespeicherten Website-Benutzername Kombinationen auf der Konsole.

## 2.3 Lösungshinweise

### 2.3.1 Allgemeine Hinweise zur Zeitmessung

- Kompilieren Sie Ihr Projekt vor der Messung unbedingt im *RELEASE* Modus (Visual Studio) und verwenden Sie das Compilerflag */Ox* in Visual Studio oder *-O2* bzw *-O3* bei Verwendung des gcc, 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, E-Mail-Client, Antivirus, etc.....), da diese das Ergebnis ebenfalls drastisch verfälschen können!

### 2.3.2 Format der TXT-Dateien

Die Messungen erzeugen für jeden gemessenen Algorithmus eine eigene Textdatei. Die Laufzeiten werden tabulatorgetrennt spaltenweise abgespeichert, 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
...
```

### 2.3.3 Beispiele zum Plotten mit MATLAB / GNUPLOT / Octave

- MATLAB
  Laden Sie aus ILIAS das MATLAB Live-Script und speichern Sie dieses im selben Ordner, wo sich Ihre Messungen befinden und führen Sie dieses Aus. Es sollte ein Graph ähnlich von 2.3.4 entstehen. Erweitern Sie das Skript, sodass alle Text-Dateien geladen und die Messwerte geplottet werden.
- 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

### 2.3.4 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):

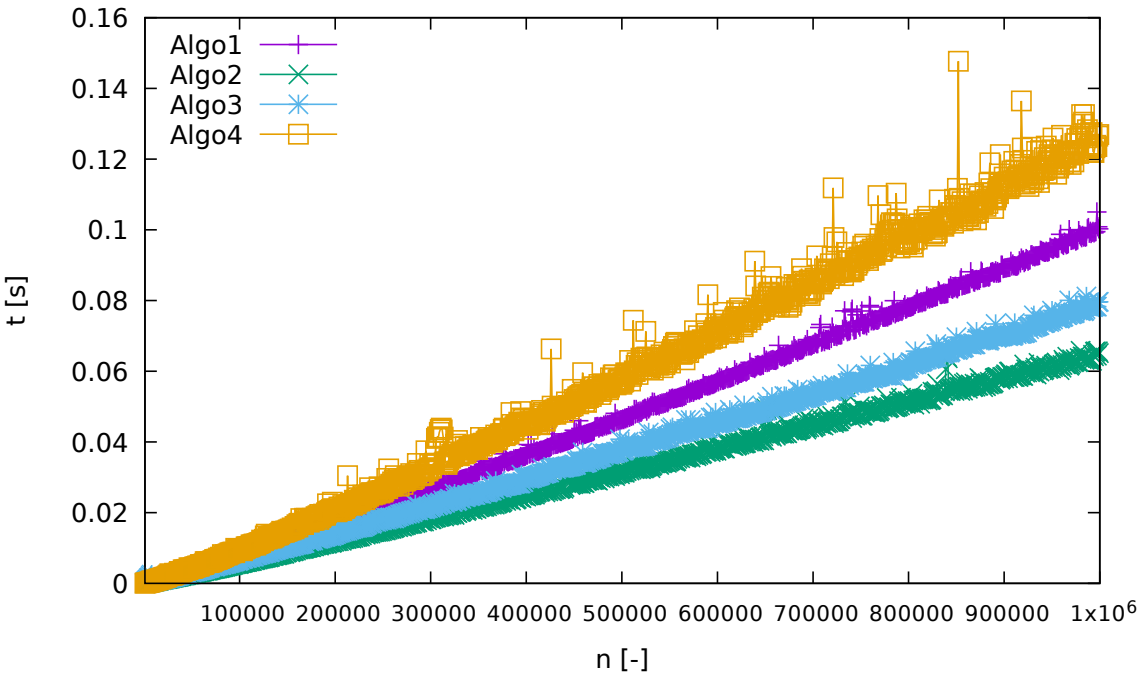


Abbildung 1: Laufzeitvergleich der Sortieralgorithmen