Praktikum 6: Algorithmen & Laufzeitmessung

Lernziele

- Implementierung einfacher Sortierverfahren
- Anwenden von Laufzeitmessungen

1 Sortierverfahren - Zeitmessung

Sie sollen ein einfaches Sortierverfahren Ihrer Wahl implementieren, um einen `vector` mit Zahlenwerten zu sortieren. Hierzu reicht eine Implementierung von beispielsweise BubbleSort völlig aus. Anschließend verwenden Sie eine Zeitmessung, um die Performance Ihres Sortierverfahrens gegenüber dem Sortierverfahren aus der Standardbibliothek zu vergleichen.

Ihnen steht mit der Datei `rahmen.cpp`[1] ein Rahmen zur Verfügung, der Ihnen bereits die Zeitmessung abnimmt.

1.1 Funktion `fill`

Implementieren Sie die Funktion `fill` die einen als Referenz übergebenen `vector` mit zufälligen Werten zwischen 0 und 10.000 füllen soll.

1.2 Eigene Sortierfunktion

Schreiben Sie eine Funktion, die einen als Referenz übergebenen `vector` sortiert.
Implementieren Sie dazu **selbst** ein beliebiges Sortierverfahren, benutzen Sie also **keine** Funktion aus der Standardbibliothek!

Eine (unvollständige) Liste von Verfahren, die Sie verwenden können:

- BubbleSort `https://de.wikipedia.org/wiki/Bubblesort`
- MergeSort `https://de.wikipedia.org/wiki/Mergesort`
- SelectionSort `https://de.wikipedia.org/wiki/Selectionsort`
- InsertionSort `https://de.wikipedia.org/wiki/Insertionsort`

Hinweis: Die „Zeitmessung“ über den Aufruf von `clock()` liefert so wie im Rahmen verwendet Werte in der Einheit „CPU ticks“. Diese sind nicht zwangsläufig zwischen zwei Systemen vergleichbar! Die C++11 Variante misst hingegen in Sekunden.

1.3 `sort`-Methode der Standardbibliothek

Die Standardbibliothek stellt im Header `algorithm` eine Sortierfunktion zur Verfügung. In den zur Verfügung gestellten Rahmenprogrammen ist der Aufruf bereits enthalten. Für weitere Informationen können Sie eine Referenz zu Rate ziehen, beispielsweise `http://www.cplusplus.com/reference/algorithm/sort/?kw=sort`

1.4 Zeitmessung

Machen Sie Versuche mit den Vektorgrößen 10.000, 100.000, 1.000.000 und 10.000.000 und notieren Sie die Ergebniszeiten. Je nach Implementierung und gewähltem Sortierverfahren kann es passieren, dass diese Zeiten sehr groß werden. Beachten Sie dies bei der Zeitmessung.

Stellen Sie die gemessenen Zeiten in einem geeigneten Diagramm dar.

[1] Bzw. `rahmenc11.cpp, welche eine andere Art der Zeitmessung aus C++11 verwendet.`

2 Fortlaufende Aufgabe: Musikbibliothek

Als letzten Ausbauschritt der Musikbibliothek sollen Sie eine Sortierung der einzelnen Lieder nach Titel implementieren. Wie Sie dies implementieren bleibt Ihnen überlassen.
Eine Möglichkeit wäre, dass Sie bereits beim Einfügen die korrekte Stelle des neu einzufügenden Liedes ermitteln und das Lied dann an dieser Stelle in das dynamische Array einfügen.

Hinweis: Im Moodle-Forum sind zwei Quellcodestücke veröffentlicht. Eine davon sollte eigentlich auf allen Systemen lauffähig sein und ermöglicht es Ihnen, `.mp3`-Dateien aus einem Verzeichnis einzulesen. Dies erspart Ihnen auf jeden Fall, bei jedem Testdurchlauf eine Anzahl an Testdaten einzufügen. Versuchen Sie daher, den Quellcodeschnipsel, in Ihr Programm einzufügen, um sich unnötige Arbeit zu sparen.

2.1 Sortierung

Innerhalb des dynamischen Arrays sollen die Lieder nach Liedtitel sortiert werden. Die Ausgabe erfolgt demnach logischerweise ebenfalls sortiert.

Wie Sie die Sortierung letzlich gestalten, bleibe Ihnen überlassen. Versuchen Sie jedoch, Ihre Anwendung so effizient wie möglich zu gestalten.

2.2 Suchen nach Titeln (Menüpunkt 5)

Während der Menüpunkt 2 (Details eines Titels anzeigen) den Benutzer nach einer ID (meist identisch mit Arrayindex eines Liedes) fragt, soll Menüpunkt 5 den Benutzer nach einem `Titel` fragen.

Führen Sie anschließend eine **binäre Suche** in Ihrer Musikbibliothek durch, ob ein Lied mit dem gesuchten Titel in Ihrer Bibliothek vorhanden ist oder nicht, und zeigen Sie gegebenenfalls alle Informationen über den gesuchten Titel an. Im Negativfall (Lied ist nicht vorhanden) informieren Sie den Benutzer entsprechend.