DAP2 Praktikum für ETIT und IKT, SoSe 2021, Langaufgabe L1

Fällig am 10.05. um 14:00

Die Regeln für Programmieren und die Punktevergabe gelten wie bei Aufgabe K1.

Vorbemerkung

Betrachten Sie den nachfolgend den als Beispiel aufgeführten `BubbleSort` – Algorithmus. Er ist als Funktionstemplate realisiert. Er sortiert den Vektor `a` aufsteigend.

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <assert.h>
#include <iomanip>
#include <vector>
#include <sstream>
#include <algorithm>

using namespace std;

template < class T > void BubbleSort(vector<T> &a) {

	assert(a.size()>0);

	for (size_t i = a.size() - 1; i > 0; i--)
		for (size_t j = 0; j < i; j++)
			if (a[j] > a[j+1])
				swap(a[j],a[j+1]);

	for (size_t i=0; i<a.size()-1; i++) assert(a[i]<=a[i+1]); // really sort-
ed?
}
```

Langaufgabe 2.1 (2 Punkte)

- Schreiben Sie eine Funktion

  ```
  vector <double> GenerateNumbers(string Selection, size_t n)
  ```

  Diese Funktion erzeugt ein Zahlenfeld der Länge `n`. Je nach Wert von `Selection` werden bei `random` zufällige, unterschiedliche Fließkommazahlen die zwischen *0* und *1* liegen, bei `up` aufsteigende Zahlen *0,1,2,3…*, bei `down` absteigende Zahlen *0,-1,-2,…*, und bei `constant` konstante Zahlen *17,17,17,…* im Zahlenfeld zurückgeliefert.

- Schreiben Sie unter Nutzung der obigen Templates ein Sortierprogramm `mysort`, dass Fließkommazahlen sortiert und folgende Eigenschaften hat:

  - Der Aufruf erfolgt mit

    ```
    mysort n random|up|down|constant [ -o ] [ -t ]
    ```

- `n` ist die Länge des zu sortierenden Feldes (das Feld ist vom Typ `double`) danach muss einer der Strings `random up down` oder `constant` angegeben werden um die Art des zur sortierenden Zahlenmaterials vorzugeben.
- Der optionale Schalter `-o` führt dazu, dass nach der Sortierung zweispaltig das Originalfeld (links) und das zu sortierte Feld (rechts) zeilenweise ausgegeben werden.
- Der optionale Schalter `-t` bewirkt, dass die Zeitmessung für den reinen Sortiervorgang aktiviert wird. Es erfolgt am Ende die Ausgabe der Laufzeit.
- Die Reihenfolge der Schalter soll beliebig sein.
- Beispiele für Aufrufe von `mysort` sind:
  - `mysort 13 up -t -o`
  - `mysort 25 random`
  - `mysort 9 up -o`

Langaufgabe 2.2 (2 Punkte)

- Schreiben Sie ein Funktionstemplate

```
template < class T > bool IsSortedAndNothingIsLost
      (vector <T> &Before, vector <T> &After)
```

  das überprüft, ob `After` aufsteigend sortiert ist und zudem sich alle Elemente aus `Before` in `After` wieder finden lassen. Damit soll sicher überprüft werden, ob die Sortierung funktioniert und zudem keine Werte dupliziert bzw. verworfen werden. Diese Testroutine muss nicht laufzeitoptimiert sein.

- Ergänzen Sie unter Nutzung des obigen Templates `mysort` um den optionalen Schalter `-c` . Dieser schaltet den Test `IsSortedAndNothingIsLost` ein. Wird ein Fehler gefunden, so wird der Text `Check failed` ausgeben und der *exit code* entsprechend auf `1` gesetzt. Bei Erfolg wird `Success` ausgegeben und der *exit code* bleibt `0`.

Langaufgabe 2.3 (2 Punkte)

- Schreiben Sie eine Funktion

```
template <class T> void InsertionSort(vector<T> &a)
```

  die das Feld `a` aufsteigend mit dem Algorithmus `InsertionSort` aus der Vorlesung sortiert.

- Erweitern Sie das `mysort` um einen weiteren optionalen Schalter `-i` zur Auswahl des Sortieralgorithmus `InsertionSort`.

Langaufgabe 2.4 (3 Punkte)

Implementieren Sie die Funktion als Template

```
template <class T> void MergeSort(vector<T> &a)
```

die das Zahlenfeld `a` aufsteigend mit Hilfe von `MergeSort` sortiert. `vector<T>::pushback()` darf nicht benutzt werden. Die Funktion `MergeSort` soll iterativ implementiert werden (d.h. keine rekursiven Aufrufe) und unnötiges Zurückkopieren am Ende von Merge vermeiden. Letzteres kann man erreichen, indem man immer abwechselnd bei Merge die Rolle des zu sortierenden Arrays des Hilfsarrays tauscht.

- Erweitern Sie das `mysort` um einen weiteren optionalen Schalter `-m` zur Auswahl des Sortieralgorithmus `MergeSort`.

Langaufgabe 2.5 (1 Punkt)

- Nutzen Sie Matlab (im Retina-Pool vefügbar), um einen Plot über die jeweilige Laufzeit zu erzeugen. Gehen Sie dazu folgendermaßen vor:
  - Starten Sie Matlab
  - Nehmen Sie für das Beispiel an, Sie hätten für die Größen des Zahlenfeldes 100,1000 und 10000 benutzt und dafür die Laufzeiten $2{,}1 \times 10^{-3}$, $4{,}5 \times 10^{-2}$ und $1{,}9 \times 10^{-1}$ Sekunden erhalten.
  - Geben Sie in Matlab ein:

```
figure; n=[100 1000 10000]; t=[2.1e-3 4.5e-2 1.9e-1]; plot(n,t);
```

- Wählen Sie sinnvolle Größen für `n`. Erhöhen Sie `n` schrittweise, bis Sie Laufzeiten von ca. 1 min erhalten, und messen dann weiter bei kleineren `n`. Bei den Zeitmessungen sollten Sie denselben Aufruf zweimal hintereinander durchführen, und den kleineren Wert wählen. Bei der Zeitmessung dürfen die Schalter `-c` und `-o` nicht aktiviert sein.
- Machen Sie Plots der Laufzeit abhängig von `n` für `BubbleSort InsertionSort` und `MergeSort` jeweils für `random,up,down,constant` (in Summe also 12 Plots).

Hinweise:

- Verwenden Sie den STL-Container `vector` aus `#include <vector>` zum Abspeichern der Zufallszahlen vom Typ `double`.
- Beachten Sie, dass `vector` unter bestimmten Umständen Exeptions erzeugen kann! Sehen Sie hierfür eine sinnvolle Fehlermeldung vor!
- Verwenden sie `assert` um Invarianten an geeigneten Stellen zu überprüfen.
- Wie üblich in diesem Praktikum sind C-Funktionen wie `atoi` oder `atof` zum Auslesen von `argv` nicht erlaubt. Verwenden Sie `isstringstream`.
- Zur Erzeugung von Zufallszahlen kann die Funktion `rand()` benutzt werden. Diese gibt eine gleichverteilte Zufallszahl im Bereich `0` bis `RAND_MAX` zurück. Zu beachten ist, dass die Funktion `rand()` für jeden Programmaufruf immer die gleiche Sequenz von Zufallszahlen generiert. Man kann dies korrigieren, indem man einen Startwert des Zufallszahlengenerators, mit der Funktion `srand(unsigned int seed)` setzt. Für den gleichen Wert von `seed` wird aber natürlich wieder die gleiche Sequenz von Zufallszahlen erzeugt. Benutzen Sie daher die aktuelle Systemzeit um damit `seed` zu initialisieren:

```
#include <cstdlib>
#include <ctime>
```

```
...

srand(time(0));
```

Programmaufrufe sollten so aussehen (Wenn Ihre Implementierung anders aussehende Ausgaben erzeugt, gibt es Punktabzug)**:**

```
mysort 0 up
First Parameter n must be positive integer
mysort -1 up
First Parameter n must be positive integer

mysort
mysort n Direction [ -o ] [ -c ] [ -t ] [ -m | -i ]

mysort 1 wurst
Second Parameter must be chosen from random|up|down|constant

mysort 1 up q
Wrong Switch, use only one of -o|-c|-t|-m|-i

mysort 1 random

mysort 1 random -o
0.335358283639 0.335358283639

mysort 4 random -o -c
Success
0.558449699338 0.112394452147
0.112394452147 0.381985771182
0.381985771182 0.452287898144
0.452287898144 0.558449699338

mysort 4 random -o -c -t -m
Success
0.143099529735 0.143099529735
0.743549953561 0.154496426300
0.910161077003 0.743549953561
0.154496426300 0.910161077003
--- It took 1.00000000e-06 seconds to compute ---

mysort 4 random -o -c -t -i
Success
0.087689542718 0.087689542718
0.292177320594 0.178376465653
0.241934595277 0.241934595277
0.178376465653 0.292177320594
--- It took 2.00000000e-06 seconds to compute ---
```