Otto-von-Guericke Universität Magdeburg
Institut für Mathematische Optimierung
Sommersemester 2016
Prof. Gennadiy Averkov

2. Übung zur Vorlesung

Algorithmische Mathematik II

(Abgabe: spätestens Dienstag, 19.04.2016, 15:15 Uhr, d.h. vor der Vorlesung)

1. Aufgabe (Votieraufgabe) **4 Punkte**

Sie möchten die 1000 teuersten Produkte aus einer ungeordneten Preisliste mit 10^7 verschieden Produkten finden. Ihnen stehen zwei Sortiervorgehensweisen zur Verfügung:

- **Variante 1**: Wiederhole 1000 Mal die *sequenzielle Suche*. Bei der sequenziellen Suche wird ein bestimmtes Element in einer Liste oder einem Array gesucht. Man geht dazu die Liste Element für Element durch, bis man es gefunden hat. *Hinweis*: Die sequentielle Suche hat die lineare Laufzeit $\mathcal{O}(n)$.
- **Variante 2**: Konvertiere die Liste in einen Array (mit der gleichen Komplexität $\mathcal{O}(n)$ wie bei der Suche), sortiere dann den Array (Laufzeit $\mathcal{O}(n \log n)$ und rufe die ersten 1000 Produkte ab.

Welches Schema würden Sie bevorzugen, angenommen die Suche von 100 Produkten in einer unsortierten Liste benötigt 0.1 Millisekunden (ms) und das Sortieren von 100 Produkten braucht ebenso 0.1 ms? Zeit für das Abrufen von Daten kann vernachlässigt werden (wie in der 2. Variante).

2. Aufgabe **5 Punkte**

Die Zeitkomplexität eines Sortier-Algorithmus sei uns gegeben durch die folgenden rekursiven Gleichungen (c seine eine positive Konstante):

$$T(n) = 3\,T(n/3) + 2c\,n;$$
$$T(1) = 0$$

Lösen Sie die Gleichungen um eine explizite Formel für $T(n)$ zu erhalten. Dabei können Sie $n = 3^m$ annehmen. Spezifizieren Sie zusätzlich die Laufzeit des Algorithmus in $\mathcal{O}$-Notation.

3. Aufgabe (Programmieraufgabe) **4 Punkte**

Implementieren Sie den Algorithmus *Sortieren durch Einfügen* wie in der Vorlesung kennengelernt mit der Programmiersprache `C++`.

Hinweis: Die Implementierung ist zwar Geschmackssache, jedoch ist sie vermutlich einfacher, wenn die *Version ohne Funktionen*, Pseudocode 4.7 der Vorlesung, umgesetzt wird.

Testen Sie ihre Implementierung anhand des folgenden Arrays: [3 9 2 8 58 5 4 31 6].

4. Aufgabe (Programmier- und Votieraufgabe) **5 Punkte**

Bus 1 fährt täglich von Stadt A in die Stadt B und hat die Abfahrtszeit AB1 und die Ankunftszeit AN1. Bus 2 fährt täglich von Stadt B in die Stadt C und hat die Abfahrtszeit AB2 und die Ankunftszeit AN2.
In einem Zeitpunkt Z entscheidet eine Person P, die sich in Stadt A befindet, in die Stadt C zu fahren. Dabei benutzt die Person P die Busse 1 und 2 und versucht schnellstmöglich die Stadt C zu erreichen. Berechnen Sie die Reisezeit der Person P mittels eines `C++`-Programms.
Liegt der Zeitpunkt AN der Ankunft eines Busses zeitlich früher als der Zeitpunkt AB der Abfahrt, so bedeutet es, dass der Bus im Zeitpunkt AB abfährt und im Zeitpunkt AN des darauffolgenden Tages ankommt. Umsteigezeiten können vernachlässigt werden. Die Zeitpunkte AB1, AB1, AB2, AN2 und Z sind im Format STUNDEN MINUTEN gegeben, mit $0 \leq$ STUNDEN ≤ 23 und $0 \leq$ MINUTEN ≤ 59. Die Daten AB1, AN1, AB2, AN2 und Z sollen in dieser Reihenfolge aus der Standarteingabe gelesen werden. Die Reisezeit muss im Format STUNDEN MINUTEN, mit $0 \leq$ STUNDEN und $0 \leq$ MINUTEN≤ 59, in die Standartausgabe geschrieben werden.

Beispieleingabe:
17 0 5 30
4 25 12 50
6 23

Beispielausgabe: 54 27

Erklärung: Die Person P entscheidet um $6:23$ Uhr nach C zu fahren. Bus 1 fährt um $17:00$ Uhr ab und kommt um $5:30$ Uhr des darauffolgenden Tages an. Bus 2 fährt um $4:25$ Uhr ab und kommt um $12:50$ Uhr des gleichen Tages an. Daher soll die Person P von $6:32$ Uhr bis $17:00$ Uhr in A warten. Dann fährt die Person P mit Bus 1 und kommt um $5:30$ Uhr des nächsten Tages in B an. Dann wartet die Person P in B bis $4:25$ Uhr des nächsten Tages, fährt anschließend mit Bus 2, und kommt um $12:50$ in C an. Der Reiseweg lässt sich schematisch so darstellen:

$6:23 \rightarrow 17:00 \rightarrow$ Mitternacht $\rightarrow 5:30 \rightarrow$ Mitternacht $\rightarrow 4:25 \rightarrow 12:50$

Die Reisezeit beträgt daher 54 Stunden und 27 Minuten. Folgende Reisezeiten sollen mit dem Programm getestet werden (inklusive der Beispielreise).

erste Reise:	**zweite Reise:**	**dritte Reise:**
7 0 8 0	7 0 8 0	7 0 8 0
9 0 10 0	9 0 3 0	9 0 3 9
6 0	6 0	6 0

Hinweis: Es wird empfohlen Funktionen in dieser Aufgabe zu verwenden und mit einem Bonuspunkt belohnt.