Bernhard Dietrich

Datenstrukturen und Algorithmen: Blatt 1

Bernhard Dietrich (6256800)
Lars Fernhomberg (6256030)
Sebastian Kniesburges (6257120)
Marcus Köthenbürger (6258550)

Aufgabe 1

Wir betrachten die Sortieralgorithmen INSERTION-SORT und MERGE-SORT. Der Algorithmus INSERTION-SORT benötige zur Sortierung von n Zahlen $8n^2$ Vergleiche, der Algorithmus MERGE-SORT benötige zur Sortierung von n Zahlen $64n\log(n)$ Vergleiche. Weiter sei Computer A in der Lage pro Sekunde 10^9 Vergleiche durchzuführen, während Computer B pro Sekunde nur $5\cdot 10^6$ Vergleiche durchführen kann. Wie lange braucht man mit INSERTION-SORT auf Computer A bzw. mit MERGE-SORT auf Computer B, um 10^7 Zahlen zu sortieren?

Für $n = 10^7$ folgt, dass INSERTION-SORT insgesamt $8n^2 = 8\cdot 10^{14}$ Vergleiche benötigt und MERGE-SORT $64n\cdot\log(n) = 64\cdot 10^7\cdot\log(10^7)$ Vergleiche benötigt (gemäß Konvention gilt: $\log(n) = \log_2(n)$).

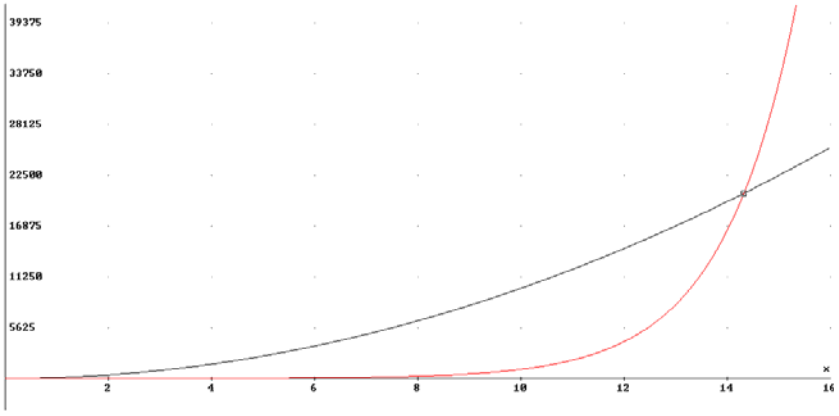
Es folgt somit für INSERTION-SORT auf Computer A: $\dfrac{8\cdot 10^{14}\,\mathrm{V}}{10^9\,\frac{\mathrm{V}}{\mathrm{s}}} = 800\,000\mathrm{s}$

Für MERGE-SORT auf Computer B gilt: $\dfrac{64\cdot 10^7\cdot\log(10^7)\,\mathrm{V}}{5\cdot 10^6\,\frac{\mathrm{V}}{\mathrm{s}}} \approx 2976,45\mathrm{s}$

Aufgabe 2

Was ist der kleinste Wert für n, so dass ein Algorithmus mit Laufzeit $100n^2$ schneller ist als ein Algorithmus, dessen Laufzeit 2^n beträgt?

n	$100n^2$		2^n
5	2500	>	32
10	10 000	>	1024
15	**22500**	**<**	**32768**
12	1400	>	4096
14	196	>	16384



Der Graph der Funtktion $100n^2$ befindet sich bis ungefähr zur Stelle $x = 15$ über dem Graphen der Funktion 2^n (wenn man den Fall $n = 0$ vernachlässigt).

Aufgabe 3
Wir betrachten das Suchproblem:
Eingabe: Eine Folge von n Zahlen $(a_1,...,a_n)$, die in einem Array A gespeichert ist, und eine weitere Zahl v.
Ausgabe: Ein Index i, so dass $v = A[i]$ oder ein spezieller Wert NIL, falls v nicht in der Folge A auftaucht.

Schreiben Sie Pseudocode für die so genannte *linerare Suche*. Die lineare Suche durchläuft das Array einmal von links nach rechts, um die Zahl v im Array A zu lokalisieren.

```
1     rückgabe ← NIL
2     for i ← 1 to length[A]
3          do if A[i] = v then
4               do rückgabe ← i
5     return rückgabe
```

Aufgabe 4
Wir betrachten das Minimumsuchproblem:
Eingabe: Eine Folge von n Zahlen $(a_1,...,a_n)$, die in einem Array A gespeichert ist.
Ausgabe: Ein Index i, so dass $A[i] \leq A[j]$ für alle $j = 1,...,n$ gilt.

1. Beschreiben Sie in Pseudocode einen Algorithmus für die Minimumsuche.
   ```
   1     kleinste ← 1
   2     for i ← 2 to length[A]
   3          do if A[i] < A[kleinste] then
   4               do kleinste ← i
   5     return kleinste
   ```

2. Formulieren Sie eine Schleifeninvariante für Ihren Algorithmus und zeigen Sie mit Hilfe dieser Invariante die Korrektheit Ihres Algorithmus.
   - Invariante: Vor jedem Durchlauf der Schleife enthält die Variable „`kleinste`“ den Index des kleinsten Elements, welches bis zu diesem Zeitpunkt gefunden wurde.
   - Initalisierung: Wird die Schleife zum ersten mal durchlaufen, ist das erste Element der Folge das bis zu diesem Zeitpunkt kleinste gefundene Element, da nur eine 1-elementige Menge (mit dem ersten Element betrachtet wurde).
   - Erhaltung: Wird während des Schleifendurchlaufs ein Element „entdeckt“, welches kleiner ist, als das bis zu diesem Zeitpunkt kleinste Element, so wird der Index des neuen Minimums in der Variable „`kleinste`“ gespeichert (vgl. Zeile 4 & 5)
   - Terminierung: Nach dem letzten Schleifendurchlauf enthält die Variable „`kleinste`“ den Index des kleinsten Elements des Arrays. Somit arbeitet der Algorithmus korrekt.