Zeitverhalten von Sortierverfahren - Beispiele für experimentelles Arbeiten im Informatikunterricht

Michael Fothe
Landesfachberater für Informatik
Albert-Schweitzer-Gymnasium Erfurt
fothe@asgspez.ef.th.schule.de

LOOK! (H. Zemanek)

Zusammenfassung: In dem Beitrag werden verschiedene Wege zum Ermitteln des Zeitverhaltens von Sortierverfahren dargestellt. Dabei wird besonderer Wert auf den Einsatz des Computers und auf Plausibilitätsbetrachtungen gelegt.

1 Einleitung

Das Thema „Sortieren und Suchen" ist fundamentaler Bestandteil des Informatikunterrichts. Die Einführung algorithmischer Grundstrukturen kann exemplarisch an Sortier- und Suchverfahren erfolgen. Die Schülerinnen und Schüler lernen dabei Algorithmen kennen, die in der kommerziellen Datenverarbeitung eine wichtige Rolle spielen (vgl. ORDER BY- und WHERE-Klausel in der Datenbanksprache SQL). Sortier- und Suchverfahren können im Unterricht angewandt werden - z. B. beim Thematisieren kryptologischer Verfahren. Jedes Sortier- und Suchverfahren besitzt spezifische Eigenschaften, aus denen sich (besonders geeignete) Einsatzgebiete des Verfahrens ableiten lassen. Eine wesentliche Eigenschaft ist der funktionale Zusammenhang zwischen der Problemgröße - also der Anzahl n der Elemente a_1, a_2, a_3,..., a_n - und der für das Sortieren oder Suchen benötigten Zeit t. Hat man nur wenige Elemente, so reicht ein Algorithmus mit schlechter Zeitkomplexität. Diese Algorithmen sind meist einfach zu verstehen und zu implementieren. Bei einer großen Anzahl von Elementen verwendet man Algorithmen mit guter Zeitkomplexität. Diese haben den Nachteil, vergleichsweise kompliziert zu sein. Der Beitrag zeigt anhand von vier Sortierverfahren, wie das Ermitteln der Zeitkomplexität im Unterricht in Angriff genommen werden kann. Nach Möglichkeit erfolgt dabei der Einsatz des Computers als Werkzeug zum Experimentieren. Zu diesem Zweck wurden Oberon-Programme entwickelt (vgl. [Fo01]), auf die nachfolgend Bezug genommen wird (www.lfk-informatik.de). Das Thema „Zeitkomplexität von Algorithmen" ist seit 20 Jahren Gegenstand des Informatikunterrichts. Dennoch gilt dieses Thema - aus Lehrersicht - als schwer zu unterrichten und - aus Schülersicht - als schwer zu verstehen. Der Beitrag soll helfen, die Situation zu verbessern. Aus Gründen der Konzentration auf das Wesentliche werden nachfolgend stets nur Zahlen sortiert.

2 Naives Sortieren

Ein naheliegendes Sortierverfahren ist das naive Sortieren (vgl. [CM90]). Bei diesem Verfahren werden alle möglichen Reihenfolgen (Permutationen) der zu sortierenden Zahlen erzeugt und jedes Mal wird überprüft, ob die Zahlen in der sortierten Reihenfolge vorliegen. Ist dies der Fall, so wird das Erzeugen und Überprüfen beendet. Zum Beispiel kann nach dieser Methode eine Liste in einem Prolog-Programm sortiert werden. Clocksin & Mellish stellen fest: „Dies ist natürlich keine sehr effiziente Methode zum Sortieren einer Liste." (S. 179) Eine glatte Übertreibung! Man ist geneigt, von Slowsort zu sprechen, denn es gibt n! Permutationen von n verschiedenen Zahlen, von denen genau eine die gesuchte ist. Im Mittel sind daher n!/2 Permutationen zu erzeugen und zu überprüfen. Ein Gefühl für das Zeitverhalten lässt sich zum Beispiel durch Abarbeiten eines Prolog- oder Oberon-Programms gewinnen. Die Rechenzeit steigt sehr schnell an. Schon für kleine n dauert es ewig, bis das Ergebnis feststeht.

3 Sortieren durch Auswählen

```
PROCEDURE Auswahl*;
VAR r, s, t, x: INTEGER;
BEGIN
  FOR r := 1 TO n-1 DO
    t := r;  x := a[r];
    FOR s := r+1 TO n DO
      IF a[s] < x THEN
        t := s;  x := a[s]
      END
    END;
    a[t] := a[r];  a[r] := x
  END
END Auswahl;
```

Die Oberon-Prozedur Auswahl realisiert das Sortieren durch Auswählen (zum Sortieren von n Zahlen in aufsteigender Reihenfolge). Beim Untersuchen des Zeitverhaltens erfolgen als Erstes Zeitmessungen (vgl. Abb. 1). Dabei werden drei Eigenschaften der gegebenen Zahlen unterschieden: Die Zahlen liegen bereits aufsteigend sortiert vor. Die Zahlen sind Zufallszahlen. Die Zahlen liegen falsch herum, also absteigend sortiert vor. Aus den Messergebnissen lässt sich die Vermutung ableiten, dass Sortieren durch Auswählen eine quadratische Zeitkomplexität besitzt. Die Vermutung soll durch Analyse des Quelltextes bestätigt werden. Modellartig werden dabei nur die zeitaufwändigen Operationen betrachtet, also die Operationen, bei denen (mindestens) ein Zugriff auf das Array a erfolgt. Das sind der Vergleich a[s] < x und die Bewegungen x := a[r], x := a[s], a[t] := a[r] und a[r] := x. (Weitere Vereinfachung: Für jede dieser Operationen wird jeweils die gleiche Rechenzeit angenommen.) Die schnelleren INTEGER-Wertzuweisungen wie t := s und das Hochzählen der Laufvariablen bleiben unberücksichtigt. Die Analyse des Quelltextes soll computerunterstützt erfolgen. In dem Oberon-Programm zum Sortieren durch Auswählen wurden zu diesem Zweck zusätzliche Ausgabeanweisungen aufgenommen. Nach jedem Vergleich bzw. nach jeder Bewegung gibt das Programm ein Zeichen aus. Jedes Mal, wenn eine Zahl an die richtige Position gebracht wurde, wird eine neue Zeile begonnen. Veränderliche Parameter sind die Festlegung, ob Vergleiche oder Bewegungen ausgegeben werden sollen, die Anzahl zu sortierender

Zahlen und die Eigenschaften dieser Zahlen.

Im Einzelnen wurden die folgenden Experimente durchgeführt:

1. Sortieren von 10 Zahlen mit unterschiedlichen Eigenschaften, dann 20 und 50 Zahlen: Die Lernenden erkennen, dass die Anzahl der Vergleiche unabhängig von den Eigenschaften der gegebenen Zahlen ist. Diese Aussage wird anhand des Quelltextes bestätigt.

2. Sortieren von 10, 20 und 50 Zahlen: Die Zeichen sind jeweils als Dreiecksfigur angeordnet (vgl. Abb. 2). Aus der Größe der Dreiecke ermitteln die Schülerinnen und Schüler die Formel $V = (n^2-n) / 2$ für die Anzahl V der Vergleiche (vgl. Abb. 2).

3. Sortieren von 10, 20 und 50 Zahlen, die jeweils aufsteigend sortiert sind: Die Lernenden ermitteln die Formel $B = 3 * (n-1)$ für die Anzahl B der Bewegungen. Die Formel wird anhand des Quelltextes bestätigt.

4. Sortieren von 10, 20 und 50 Zahlen, die jeweils absteigend sortiert sind: Die Lernenden ermitteln die Formel $B \approx 3 * (n-1) + n^2 / 4$ durch Zerlegen der Figur in ein Rechteck und ein Dreieck (vgl. Abb. 2). Die Trichterform entsteht, weil in jedem Durchlauf der inneren For-Anweisung zwei Elemente an die richtigen Positionen gebracht werden.

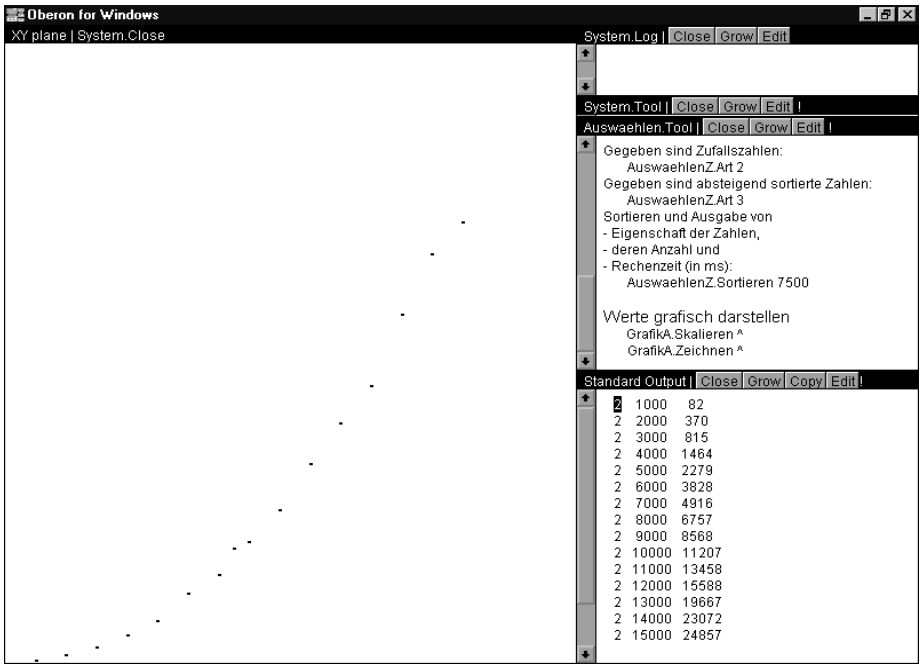

Abb. 1: Zeitmessungen beim Sortieren durch Auswählen

5. Sortieren von 10, 20 und 50 Zufallszahlen: Die Schülerinnen und Schüler vermuten, dass die Anzahl der Bewegungen zwischen den beiden Fällen aufsteigend sortiert / absteigend sortiert liegt. Sie können beobachten, dass die Anzahl der Bewegungen beim Sortieren von Zufallszahlen schneller als linear ansteigt. Genauere Erkenntnisse werden durch Experimente nicht gewonnen. Die mittlere Anzahl an Bewegungen ist schwierig zu bestimmen. Das theoretisch gewonnene Ergebnis ist $B \approx n * (\ln n + g)$, wobei g die EULERsche Konstante 0,577216 ist (vgl. [Wi83]).

```
gegeben: 20 Zahlen absteigend sortiert

Ausgabe der Vergleiche              Ausgabe der Bewegungen

*******************                 **********************
*******************                 **********************
*****************                   ******************
*****************                   ****************
***************                     **************
**************                      ************
************                        **********
***********                         ********
**********                          ******
*********                           ****
********                            ***
*******                             ***
******                              ***
*****                               ***
****                                ***
***                                 ***
**                                  ***
*                                   ***

190 Vergleiche                      157 Bewegungen
```

Abb. 2: Ausgabe der ausgeführten Operationen beim Sortieren durch Auswählen

Aus den Experimenten lassen sich die folgenden Erkenntnisse ableiten:

- Die Anzahl der Vergleiche wächst stets quadratisch mit der Problemgröße.

- Sind die gegebenen Zahlen bereits aufsteigend sortiert, so wächst die Anzahl der Bewegungen linear. Sind die gegebenen Zahlen absteigend sortiert, so wächst die Anzahl der Bewegungen quadratisch. Werden Zufallszahlen sortiert, so wächst die Anzahl der Bewegungen zwischen linear und quadratisch.

- Die minimale Anzahl an Operationen ist dann auszuführen, wenn die gegebenen Zahlen bereits aufsteigend sortiert sind. Die maximale Anzahl an Operationen ist vermutlich dann auszuführen, wenn die gegebenen Zahlen absteigend sortiert sind.

4 Quicksort

Die Oberon-Prozedur Sortieren sortiert n Zahlen mit dem Verfahren Quicksort. Der Algorithmus wurde gründlich untersucht (vgl. [OW93]). Bei der Untersuchung wurden mathematische Kenntnisse und Fähigkeiten in einem Umfang verwendet, der für den Unterricht an Hochschulen, jedoch nicht oder nur eingeschränkt an Schulen vorausgesetzt werden kann. Den Weg über Rekursionsgleichungen und deren Lösung, wie er in einem Schulbuch vorgeschlagen wird, betrachtet der Autor daher als nicht realistisch (vgl. [Ba93]). In einem anderen Schulbuch (vgl. [KW92]) und in einer Handreichung für Lehrer (vgl. [St93]) wird die Zeitkomplexität von Quicksort mitgeteilt und es werden die Ergebnisse von Zeitmessungen im Vergleich zu anderen Sortierverfahren angegeben. Die Auswirkungen der Wahl des Trennelementes werden nicht systematisch untersucht.

```
PROCEDURE Sortieren*;
  PROCEDURE Sort(l, r: INTEGER);
  VAR i, j, x, w: INTEGER;
  BEGIN
    i := l;  j := r;
    x := a[(l + r) DIV 2];
    REPEAT
      WHILE a[i] < x DO INC(i) END;
      WHILE x < a[j] DO DEC(j) END;
      IF i <= j THEN
        w := a[i];  a[i] := a[j];  a[j] := w;
        INC(i);  DEC(j)
      END
    UNTIL i > j;
    IF l < j THEN Sort(l, j) END;
    IF i < r THEN Sort(i, r) END
  END Sort;
BEGIN
  Sort(1, n)
END Sortieren;
```

Nachfolgend wird ein Weg dargestellt, der aus drei Schritten besteht.

Im ersten Schritt werden mit einem Oberon-Programm 10.000 REAL-Zahlen sortiert und die benötigte Zeit wird gemessen. Dabei werden zwölf Fälle unterschieden, und zwar drei mögliche Eigenschaften der gegebenen Zahlen:

1 - Die Zahlen liegen bereits aufsteigend sortiert vor.
2 - Die Zahlen sind Zufallszahlen.
3 - Die Zahlen liegen falsch herum, also absteigend sortiert vor.

und vier Möglichkeiten für die Wahl des Trennelementes:

1 - Das erste Element der Teilfolge.
2 - Das mittlere Element der Teilfolge.
3 - Das letzte Element der Teilfolge.
4 - Ein zufälliges Element der Teilfolge.

Nach jeder Zeitmessung werden vier Zahlen ausgegeben: Eigenschaften der gegebenen Zahlen, Wahl des Trennelementes, Anzahl an Zahlen, benötigte Zeit in ms (vgl. Abb. 3).

```
1    1    10000     6834
1    2    10000       18
1    3    10000     6927
1    4    10000       21
2    1    10000       41
2    2    10000       39
2    3    10000       39
2    4    10000       27
3    1    10000     6767
3    2    10000        9
3    3    10000     6372
3    4    10000       28
```

Abb. 3: Zeitmessungen beim Sortieren von 10.000 Zahlen mit Quicksort

Aus den Messergebnissen lässt sich ableiten, dass Quicksort langsam ist, wenn die gegebenen Zahlen aufsteigend oder absteigend sortiert sind und wenn als Trennelement stets das erste oder das letzte Element einer Teilfolge genommen wird. Ansonsten ist Quicksort schnell. Häufig staunen die Schülerinnen und Schüler über das Ergebnis der Zeitmessungen und über die Rolle des Zufalls. Mit diesem Computerexperiment wurden bereits wesentliche Eigenschaften von Quicksort festgestellt.

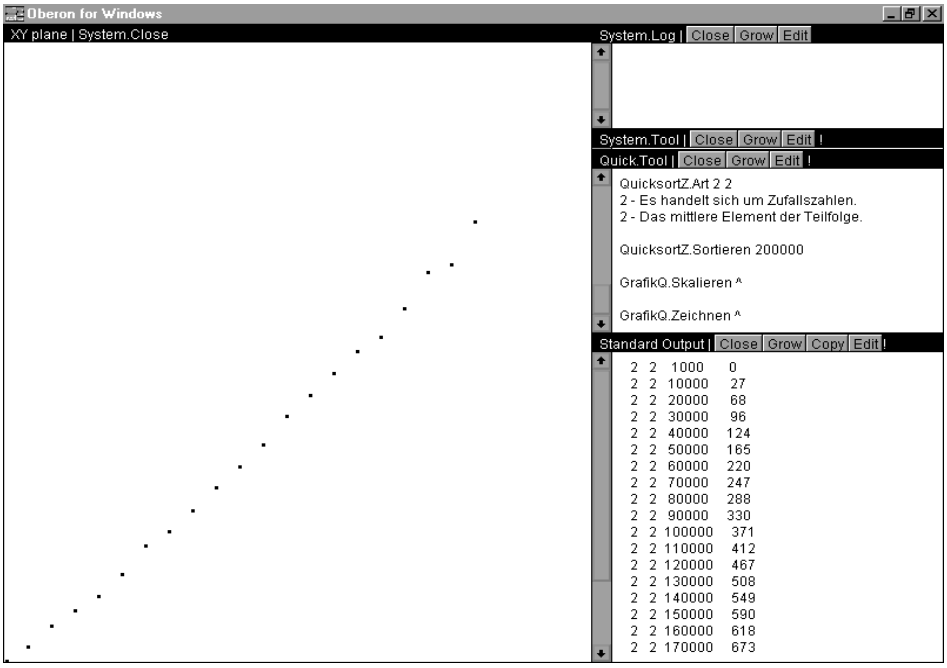

Abb. 4: Zeitmessungen bei Quicksort

Im zweiten Schritt soll der funktionale Zusammenhang zwischen der Problemgröße und der Zeit mit Hilfe von Zeitmessungen ermittelt werden. Das Vorgehen wurde bereits beim Sortieren durch Auswählen beschrieben. Die Schülerinnen und Schüler lesen jedoch meist lineares Zeitverhalten aus der grafischen Darstellung heraus (vgl. Abb. 4), was verständlich wird, wenn man den Graph der Funktion y = x * ld x betrachtet. Diese Funktion hat die Eigenschaft, dass sich beim Verdoppeln des x-Wertes der y-Wert auf das (2 * (1 + 1 / ld x))-fache vergrößert. Auch die rekursiven Aufrufe der Prozedur Sort haben Einfluss auf die Zeit, die zum Sortieren benötigt wird. Die Anzahl an Aufrufen der Prozedur - und damit der Zeitaufwand zum internen Verwalten des Rekursionsstapels - steigt linear mit der Problemgröße an.

Im dritten Schritt wird das Zeitverhalten von Quicksort im besten und schlechtesten Fall mit Hilfe einer Plausibilitätsbetrachtung verdeutlicht (vgl. [Fo98]). Für das Erschließen des Zeitverhaltens eignet sich ein einfaches Modell, das nachfolgend zuerst für den besten Fall beschrieben wird. Wir nehmen an, dass k Zahlen zu sortieren sind. Für das Abarbeiten der Repeat-Anweisung werden näherungsweise k Zeiteinheiten benötigt. (Jede Zahl wird mit dem Trennelement verglichen. Manchmal werden zusätzlich zwei Zahlen getauscht.) Dann werden die linke und die rechte Teilfolge in gleicher Weise bearbeitet. Im besten Fall wird die Teilfolge beim Sortieren stets in zwei gleich große Teilfolgen zerlegt. In Abb. 5 wird die Situation für 16 Zahlen, die zu sortieren sind, dargestellt. Das Zeichen x steht für eine Zahl und damit nach den Erläuterungen für eine Zeiteinheit. Die Zeichen x lassen sich zu einem Rechteck zusammenschieben. Es passt alles gut zusammen. Die Fläche als Maß für die benötigte Zeit zum Sortieren von 16 Zahlen hat den Inhalt 16 * 4. Durch weitere Beispiele (z. B. 32 und 8 Zahlen) lässt sich der Zusammenhang, dass die benötigte Zeit proportional zu n * ld n ist, finden.

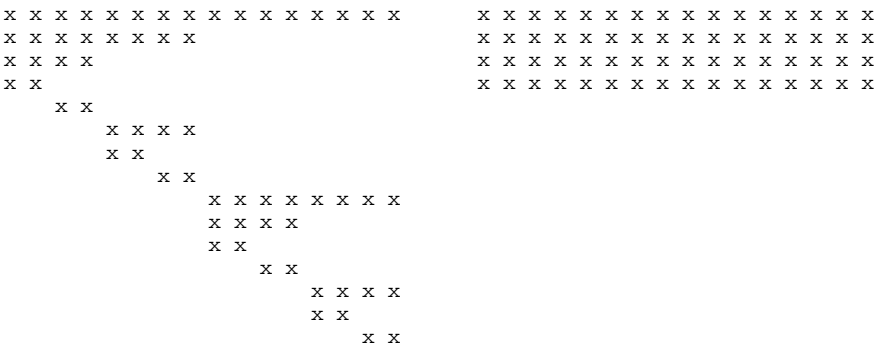

Abb. 5: Quicksort im besten Fall

Der schlechteste Fall liegt zum Beispiel dann vor, wenn als Trennelement stets die größte Zahl einer Teilfolge genommen wird. Bei 16 Zahlen entsteht ein Dreieck, das

den Flächeninhalt 135 besitzt (vgl. Abb. 6). Allgemein ist der Flächeninhalt $(n^2+n-2)/2$. Damit ist die quadratische Zeitkomplexität plausibel gemacht.

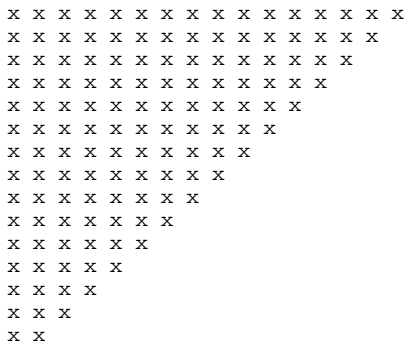

Abb. 6: Quicksort im schlechtesten Fall

Abschließend werden die Ursachen für die Messergebnisse aus dem ersten Schritt diskutiert. Die Lernenden erkennen u. a., dass auch beim Sortieren von Zufallszahlen und Auswählen von zufälligen Trennelementen der schlechteste Fall eintreten kann, dass dies jedoch äußerst unwahrscheinlich ist.

Im Modell werden die Vergleiche präzise berücksichtigt. Die Bewegungen dagegen werden sehr großzügig betrachtet. Diese Vereinfachung ist für eine Plausibilitäts-betrachtung akzeptabel. Zur Orientierung wurden für unterschiedliche Anzahlen von REAL-Zufallszahlen die verschiedenen Arten von Operationen gezählt (als Trennelement wurde stets das mittlere Element einer Teilfolge genommen). Auf jeweils 5 bis 6 Vergleiche kam dabei ein Tausch zweier Zahlen (vgl. Abb. 7). Ein Tausch ist aus drei Bewegungen zusammengesetzt.

Zahlen	Vergleiche	Bewegungen	Tausche zweier Zahlen	Aufrufe der Prozedur Sort
20.000	371.583	234.688	72.279	17.851
50.000	1.036.438	630.078	195.219	44.421
100.000	2.344.248	1.317.376	409.464	88.984
150.000	3.573.756	2.028.153	631.559	133.476

Abb. 7: Anzahl an verschiedenen Operationen bei Quicksort

5 Spagetti-Computer

Der Funktionalität eines Analogcomputers liegt eine physikalische Größe zu Grunde. Beim Spagetti-Computer ist dies die Länge (vgl. [De84]). Für jede Zahl wird eine Spagetti-Stange entsprechender Länge hergestellt. Zum Beispiel ist für die Zahl 85 eine Spagetti-Stange der Länge 85 mm herzustellen. Zum Sortieren wird das Bündel an Spagetti-Stangen auf einer Ebene kräftig aufgestoßen und dann werden die Spagetti-Stangen der Länge nach von oben nach unten weggenommen und gemessen. Die Zahlen werden aufgeschrieben. Dieser Analogcomputer besitzt lineare Zeitkomplexität, denn bei der doppelten Anzahl an Zahlen verdoppelt sich die Zeit für das Herstellen der Spagetti-Stangen (Vorbereitungsphase) und das Wegnehmen und Messen der Länge (Nachbereitungsphase). Der Augenblick für das Aufstoßen des Bündels kann vernachlässigt werden (Analogphase). Das Sortieren mit dem Spagetti-Computer wird im Unterricht in einem Gedankenexperiment unter idealen Annahmen durchgespielt. Die praktische Ausführbarkeit ist natürlich nur für wenige Zahlen gegeben. Der interessante Ansatz scheitert leider bei einer größeren Anzahl von zu sortierenden Zahlen.

6 Resümee

Die Untersuchungen zeigen, dass wichtige Aussagen zur Zeitkomplexität von Algorithmen über Experimente mit dem Computer gewonnen werden können. Diese Methode besitzt jedoch Grenzen. Mit Plausibilitätsbetrachtungen und Gedankenexperimenten kann das Verständnis der Lernenden vertieft werden. Die Beispiele verdeutlichen die Möglichkeit, das Thema „Zeitkomplexität von Algorithmen" weder zu theoretisch noch zu vereinfachend im Informatikunterricht zu behandeln. Das in diesem Beitrag vorgestellte Vorgehen thematisiere ich seit einigen Jahren in der Thüringer Lehrerfortbildung. Mehrere Informatiklehrerinnen und -lehrer teilten mir ihre Erfahrungen bei der Umsetzung der Ideen in ihrem Unterricht mit. Dies war hilfreich bei der Abfassung dieses Beitrags.

Literaturverzeichnis

[Ba93] Baumann, R.: Informatik für die Sekundarstufe II. Band 2. Ernst Klett Schulbuchverlag Stuttgart, 1993.

[CM90] Clocksin, W. F.; Mellish, C. S.: Programmieren in Prolog. Springer-Verlag Berlin, Heidelberg, New York, 1990.

[De84] Dewdney, A. K.: Computer-Kurzweil. Spektrum der Wissenschaft Heft 9/1984, S. 8-13.

[Fo98] Fothe, M.: Ergebnisse der Didaktischen Werkstatt Dezember 1998. Homepage der Thüringer Landesfachkommission Informatik (www.lfk-

informatik.de).

[Fo01] Fothe, M.: Problemlösen mit OBERON. Konzeption und Einsatz eines elektronischen Lehrbuchs. LOG IN 21 (2001) Heft 2, S. 33-40.

[KW92] Kreutzer, K.-H.; Wiedemann, A.: Informatik für Einsteiger. Ehrenwirth Verlag München, 1992.

[OW93] Ottmann, T.; Widmayer, P.: Algorithmen und Datenstrukturen. 2. Aufl. BI-Wissenschaftsverlag Mannheim, Leipzig, Wien, Zürich, 1993.

[St93] Stegmaier, E. L.: Suchen, Sortieren, Aufwand. Pädagogisches Zentrum Bad Kreuznach, 1993.

[Wi83] Wirth, N.: Algorithmen und Datenstrukturen. Pascal-Version. 3. Aufl. Teubner Stuttgart, 1983.

[Ze88] Zemanek, H.: Der Geist in der Flasche: Warum der Computer nicht ausschaut. Informationstechnik - it Heft 1/1988, S. 3-10.