Erschienen in: *Informatik mit Kopf, Herz und Hand, Praxisbeiträge zur INFOS 2011, 14. GI Fachtagung Informatik und Schule* (2011), S. 280–288

Zeitverhalten von Sortierverfahren – Experimentelles Arbeiten mit linearer Regression

Christian Wach

Technische Universität Darmstadt
Didaktik der Informatik
Hochschulstr. 10
64289 Darmstadt
wach@cs.tu-darmstadt.de

October 19, 2011

Die formale Analyse der Zeitkomplexität von Algorithmen ist aus verschiedenen Gründen ein schwieriges Thema für Schüler und Lehrer. Dieser Artikel beschreibt in Anlehnung an Fothes [Fot03] Vorschläge zur experimentellen Analyse der Zeitkomplexität eine weiterführende experimentelle Analyse mit Hilfe von Tabellenkalkulationen.

Mit Hilfe eines Frameworks werden Messwerte systematisch durch die Schüler erfasst und in eine Tabellenkalkulation importiert. Mit Hilfe der Tabellenkalkulation werden verschiedene Einflussfaktoren und der genaue funktionale Zusammenhang zwischen Problemgröße und durchgeführten Operationen untersucht. Durch Regressionsrechnung werden verschiedene Funktionen an die Messwerte angepasst und auf ihre Genauigkeit untersucht. Ein so bestimmter funktionaler Zusammenhang kann im weiteren Unterrichtsgeschehen durch eine formale Betrachtung ergänzt werden.

1 Einleitung

Such- und Sortieralgorithmen sind auch heute noch fester Bestandteil der Informatikcurricula an Schulen und Universitäten. Das Themengebiet Suchen und Sortieren eignet

sich nicht nur zur Einführung der algorithmischen Grundstrukturen und der Datenstruktur Array, sondern bietet auch zahlreiche Anknüpfungspunkte an weitere fundamentale Ideen der Algorithmisierung [Sch93]. So wird die Evaluation von Algorithmen erforderlich, wenn durch Schüler unterschiedliche Algorithmen zur Lösung eines Problems vorgestellt werden. Löst der Algorithmus das Problem korrekt? Wie effizient löst er es?

Die Analyse der Zeitkomplexität von Algorithmen gilt als schwieriges Thema. Eine exakte formale Analyse ist meist nicht möglich, da die mathematischen Werkzeuge zur Analyse fehlen. Faustformeln, Abschätzungen und Beweise werden nur rezipiert. Eine explorative Erarbeitung durch die Schüler ist auf formaler Ebene nicht möglich.

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-i-1) = \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i = && (1)\\
&= (n-1)^2 - \frac{(n-2)(n-1)}{2} = \frac{2(n-1)^2 - (n-1)(n-2)}{2} && (2)\\
&= \frac{(n-1)(2(n-1)-(n-2))}{2} = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n && (3)
\end{aligned}
$$

Figure 1: Exakte Bestimmung der Anzahl der Vergleiche bei Selectionsort

So endet eine formale Analyse des Selectionsort mit der Bestimmung der durchgeführten Vergleiche in Abhängigkeit der Anzahl der Elemente im Array (s. Abb. 1). Die Schleifen werden in Summen überführt und mit Hilfe der Gaußschen Summenformel werden diese vereinfacht. Die Summation wird dann meist nur sprachlich beschrieben oder durch Abschätzung der inneren Schleife vermieden. Aus dieser formalen Betrachtung werden oft Regeln für die Zeitkomplexität von verschachtelten Schleifen abgeleitet, die dann von den Schülern angewendet werden.

Doch schon bei sehr einfachen Sortieralgorithmen können diese Regeln nicht mehr angewendet werden. Gnomesort (s. Abb. 2), der als bidirektionaler Bubblesort oder Insertionsort betrachtet werden kann, besteht nur aus einer Schleife. Dass diese *eine* Schleife zu *quadratischer* Laufzeit führt ist nicht nur überraschend, sondern auch deutlich schwieriger formal zu zeigen.

Viele für die Analyse der Zeitkomplexität wesentliche Fragen können durch die Schüler selbst experimentell erarbeitet werden [Fot03]. So muss zunächst die *Problemgröße n* bestimmt werden. Experimente zeigen sehr schnell, dass die Anzahl der zu sortierenden Elemente einen wesentlichen Einfluss auf die Laufzeit hat. Dieser Einfluss auf die Laufzeit des Algorithmus wird im Anschluss genauer analysiert.

Hierzu muss zunächst der Begriff der Laufzeit präzisiert werden. Der intuitive Zugang den Algorithmus auf einem Computer auszuführen und die benötigte Zeit zu stoppen führt zu einigen Problemen:

1. Die gemessene Zeit schwankt bei konstanter Eingabe auf einer Maschine.

2. Meßergebnisse von unterschiedlichen Computern sind nicht vergleichbar.

Die Anzahl der durchgeführten Operationen als Maß der Zeitkomplexität vermeidet diese Probleme. Können *besonders zeitkritische Operationen* identifiziert werden, kann die Zeitkomplexitätsbestimmung auf diese Operationen begrenzt werden. Oft ist unklar, welche Operationen als zeitkritisch einzustufen sind. Das Sortieren von Strings, als Stellvertreter für reale Daten, erleichtert diese Betrachtung. Lese- und Schreiboperationen von Strings, sowie deren Vergleiche sind somit deutlich als zeitkritische Operationen identifizierbar.

Die Anzahl der benötigten zeitkritischen Operationen ist nicht nur von der Problemgröße abhängig. Die Vorsortierung des Arrays hat teilweise einen signifikanten Einfluss auf die Zeitkomplexität des Algorithmus. Dieser Einfluss kann auch experimentell untersucht werden [Fot03].

Im diesem Beitrag werden Messwerte systematisch erfasst und auf einen funktionalen Zusammenhang untersucht. Die Bestimmung eines funktionalen Zusammenhangs aus Meßergebnissen kann den Schülern bereits aus einem computergestützten Mathematik- oder Physikunterricht bekannt sein. Hierzu werden verschiedene lineare Modelle mit linearer Regression an die Messwerte angepasst und untersucht. Da die lineare Regression in vielen Tabellenkalkulationen (z. B. Microsoft Excel 2010) integriert ist, kann diese als Blackbox verwendet werden.

2 Erfassung von Messwerten

Zur einheitlichen und unkomplizierten Erstellung von Messreihen sollte den Schülern ein Programmiergerüst oder auch Framework vorgegeben werden. Das Framework sollte die Generierung von Testdaten, die Durchführung und Protokollierung der Testläufe und den Export der Daten als kommaseparierte Werte (CSV) ermöglichen. Schüler erweitern das Framework lediglich um eigene Implementierungen von Sortierverfahren und erstellen mit dem Framewort Messreihen, die im Anschluß analysiert werden.

Ein mögliches Programmiergerüst in Java steht mit **Sortkomp** [Wac11] zur Verfügung. Um ein Sortierverfahren dem Framework hinzuzufügen, muss lediglich die abstrakte Klasse `SortierAlgorithmus` beerbt werden. In der erbenden Klasse muss die Methode `sortiereArray` implementiert werden. In dieser Methode können die in der Superklasse definierten Methoden zur Operationszählung verwendet werden. Zusätzlich stehen auch Methoden zur Vertauschung und zum Vergleich zur Verfügung, die die benötigten Operationen automatisch zählen. Eine Beispielimplementierung des Gnomesort Algorithmus finden Sie in Abb. 2. In den Zeilen 6 und 9 wurden die Hilfsmethoden der Superklasse verwendet.

So können mit geringem Aufwand verschiedenste Messreihen für unterschiedliche Sortierverfahren erzeugt werden und direkt in eine Tabellenkalkulation importiert werden.

Bei der Benutzung von Sortkomp sind einige Besonderheiten zu beachten:

1. Sortkomp unterstützt ausschließlich das Sortieren von String Arrays. So werden Daten von Zählvariablen klar unterschieden und Datenzugriffe und -vergleiche sind

```
1  public class GnomeSort extends SortierAlgorithmus {
2    @Override
3    public void sortiereArray(String[] arr) {
4      int position = 0;
5      while (position < arr.length - 1) {
6        if (vergleicheKleinerGleich(arr[position], arr[position + 1]))
7          position++;
8        else {
9          vertausche(arr, position, position + 1);
10         if (position == 0)
11           position++;
12         else
13           position--;
14     }}}}
```

Figure 2: Implementierung von Gnomesort im Sortkomp Framework [Wac11]

die zeitlich dominierenden Operationen.

2. Die zufälligen Eingabedaten werden nur einmal mit der maximalen Arraygröße generiert. Testläufe mit kleinerer Problemgröße n verwenden die ersten n Elemente des generierten Arrays. Dies erklärt die monoton steigenden Kurvenverläufe bei den Testläufen ohne Vorsortierung.

Für die weitere Auswertung wurden Testreihen für Selectionsort, Bubblesort, Gnomesort und Quicksort erstellt. Benötigte Vergleiche, Vertauschungen, Lese- und Schreiboperationen wurden für Arraylängen von 1000 - 9900 in Hunderterschritten erfasst. Für jedes Verfahren wurde je eine Testreihe für unsortierte, aufsteigend sortierte und absteigend sortierte Daten erfasst. Bei der Implementierung des Quicksort wurde das Pivotelement zufällig gewählt.

3 Auswertung der Messergebnisse mit einer Tabellenkalkulation

Jede Testreihe ist als separate CSV Datei abgespeichert und kann direkt in Excel oder einer anderen Tabellenkalkulation geöffnet werden. In jeder Zeile sind Problemgröße n und die Anzahl der Vergleiche, Vertauschungen, Lese- und Schreiboperationen erfasst.

Vor weiterführenden Analysen können diese Daten zunächst grafisch veranschaulicht werden. In den Abb. 3 und 4 wurde die Anzahl der erfolgten Vertauschungen und Vergleiche in Abhängigkeit von der Problemgröße für alle erfassten Messreihen angegeben[1]. Der Einfluss der Vorsortierung kann bereits mit den Diagrammen untersucht werden. Auch der Einfluss der Wahl des Pivotelementes bei der Durchführung des Quicksorts kann grafisch untersucht werden(vgl. [Fot03]).

[1]Die Abbildungen wurden aus Satzgründen nicht mit Excel, sondern mit dem ggplot2 [Wic09] Paket in R erstellt. Die Diagramme können grundsätzlich auch mit Excel erzeugt werden.

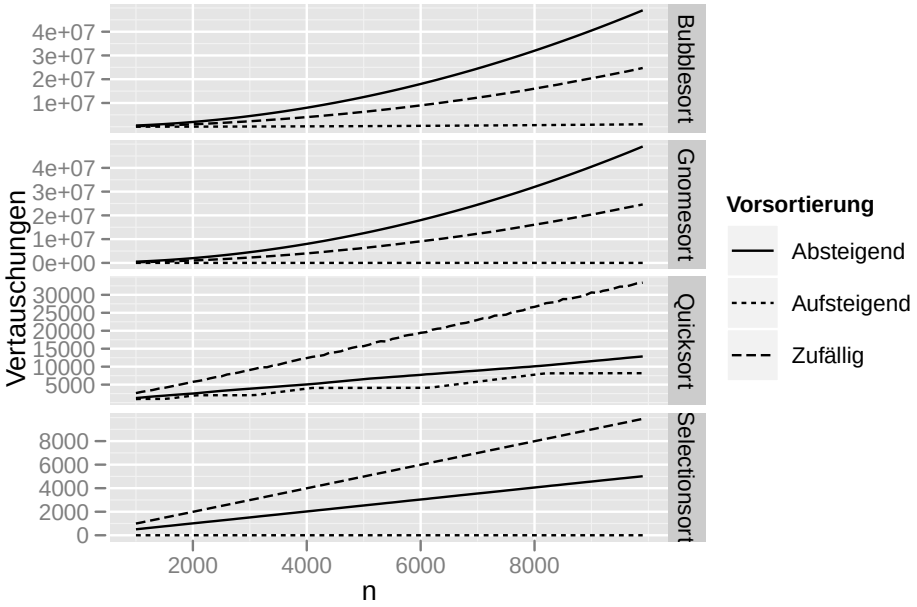

Figure 3: Vertauschungen in Abhängigkeit von der Problemgröße

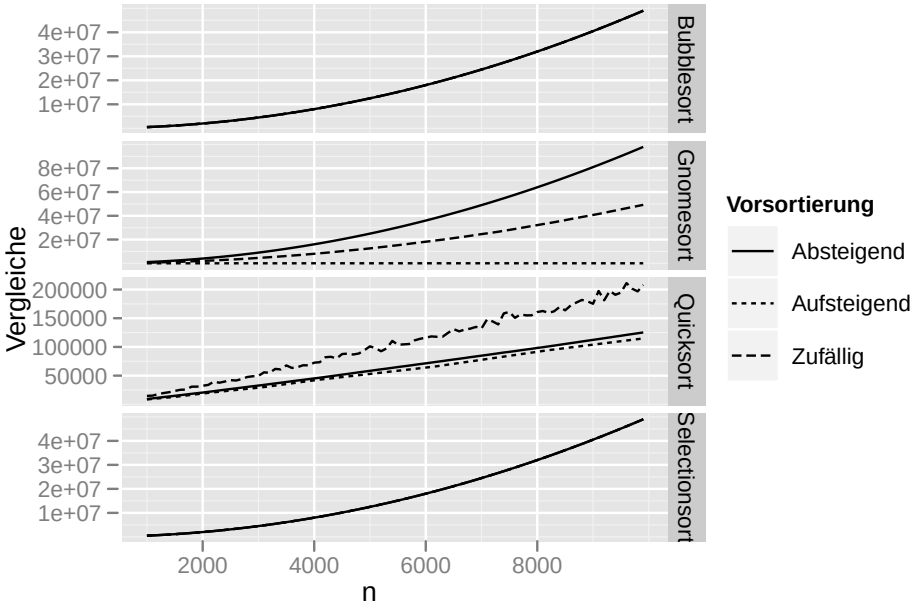

Figure 4: Vergleiche in Abhängigkeit von der Problemgröße

Aus den Kurvenverläufen in den Plots können erste Hypothesen zum funktionalen Zusammenhang aufgestellt werden. So kann z. B. linearer Zusammenhang $V(n) = an+b$ oder quadratischer Zusammenhang: $V(n) = an^2+bn+c$ unterstellt und auf Plausibilität untersucht werden. Die unterschiedliche Skalierung der beiden Achsen erschwert hierbei das Wiedererkennen bereits bekannter Funktionen.

Doch wie können die verschiedenen Hypothesen zum funktionalen Zusammenhang auf Plausibilität geprüft werden? Mit Hilfe der linearen Regression können lineare Modelle, also Modelle bei denen die zu bestimmenden Regressionskoeffizienten nur linear eingehen, an die beobachteten Daten angepasst werden. Ein vermuteter exponentieller Zusammenhang, lässt sich durch Logarithmieren in ein lineares Modell überführen. Die Regression liefert immer ein Ergebnis, ob dieses Ergebnis eine zufriedenstellende Beschreibung des funktionalen Zusammenhangs ist, muss geprüft werden.

Eine einfache, wenn auch mathematisch nicht exakte, Form der Überprüfung ist der Test der Prognosegüte des Modells. Hierzu wird ein Modell zunächst an einen Teil der Beobachtungen angepasst. Mit dem so entstehenden Modell wird der Wert für die übrigen Werte prognostiziert. Der Fehler der Schätzung für jedes n entspricht der Differenz des berechneten Wertes mit dem beobachteten Wert. Wenn in den berechneten Fehlern ein systematischer Trend vorliegt, ist davon auszugehen, dass das tatsächliche Wachstum asymptotisch über- oder unterschätzt wurde. Dies kann auch graphisch in Tabellenkalkulationen geprüft werden.

Zudem muss auch der Aspekt einer Überanpassung beachtet werden. So besteht immer die Möglichkeit, dass ein zu komplexes Modell an die Daten angepasst wurde. Additive Konstanten in der Modellfunktion führen beispielsweise bei den hier betrachteten Algorithmen zu einer Überanpassung. Eine solche Konstante entspräche einem Initialisierungsaufwand, der unabhängig von der Problemgröße ist.

Eine mathematisch exakte Überprüfung der Überanpassung ist im Schulunterricht, wegen der erforderlichen Statistikkenntnisse, nicht möglich. Allerdings können Schüler mit Hilfe von Plausibilitätsüberlegungen recht einfach prüfen, ob eine Überanpassung vorliegen könnte. Wenn sich ein einfacheres Modell (also eines mit weniger zu bestimmenden Koeffizienten) finden lässt, dass den funktionalen Zusammenhang ebenso gut beschreibt, ist dieses dem komplexeren Modell vorzuziehen. Ein Anhaltspunkt für Modellvereinfachungen können Koeffizienten sein, die numerisch entweder sehr klein oder sehr groß sind.

Lineare Regression wird in Excel von der `RGP`-Funktion zur Verfügung gestellt. Bei `RGP` handelt es sich um eine *Matrixfunktion*, die mehrere Ergebnisse (die angepassten Koeffizienten) zurückgibt. Weitere Informationen zu `RGP` und Matrixfunktionen finden sich in der Hilfe.

Zur Regressionsrechnung ist es zweckmäßig ein neues Arbeitsblatt zu erstellen und die nötigen Werte aus den Messwerten zu kopieren. Vorschläge zur Organisation des Arbeitsplattes und Beispielrechnungen finden sich in den folgenden Abschnitten.

4 Analyse Bubblesort

Nach Betrachtung des Diagramms in Abb. 4 lässt sich zwischen der Problemgröße n und der Anzahl der Vergleiche ein quadratischer Zusammenhang vermuten. Um diese Vermutung zu überprüfen, passt man das lineare Modell $V(n) = an^2+bn$ an die Beobachtungen an.

In ein neues Arbeitsblatt der Tabellenkalkulation[2] werden zunächst die Beobachtungen übernommen. In Spalte A werden die Vertauschungen übernommen, in Spalte B die zugehörige Problemgröße n. In der ersten Zeile steht jeweils die Legende in den Zeilen 2-91 die Beobachtungen.

Da der Einfluss von n^2 untersucht werden soll, muss hierzu eine neue Spalte C angelegt werden. Mit der Formel `C2=B2*B2` und der Ausfüllen Funktion können die Daten vervollständigt werden. Es ist hilfreich zur späteren Analyse einige Spalten neben den Daten freizulassen. Das Modell kann nun an die Daten angepasst werden. Nach Eingabe der Matrixfunktion `RGP(A2:A32;B2:C32;FALSCH)` bestimmt Microsoft Excel die Koeffizienten 0.5 und -0.5 aus den ersten 30 Beobachtungen. Es ist darauf zu achten, dass Excel die Koeffizienten in umgekehrter Reihenfolge ausgibt!

Mit dem angepassten Modell $V(n) = 0.5n^2 - 0.5n$ können nun die erwarteten Werte (`D2=0.5*C2-0.5*B2`) und der Schätzfehler (`E2=D2-A2`) berechnet werden. Die gefundene Funktion beschreibt den funktionalen Zusammenhang *perfekt*. Dieser *Sonderfall* der Regression sollte im Unterricht thematisiert werden. Dies kann z. B. im Vergleich zu der Anzahl der Vertauschungen bei zufälligen Eingaben geschehen.

Abb. 5 zeigt die Schätzfehler für die drei Modelle $y = an, y = an^2, y = an^2 + bn$. Die Funktionen wurden nur an die ersten 30 Werte ($n \leq 4000$) angepasst. Sehr gut erkennbar ist der systematische Fehler der ersten beiden Modelle.

5 Analyse Quicksort

Die Analyse des Quicksort Algorithmus erfolgt ebenso nach dem in Abschnitt 3 beschriebenen Schema. Nach Betrachtung der Diagramme Abb. 4 wird meist ein linearer Zusammenhang vermutet (vgl. [Fot03]).

Experimente mit verschiedenen Modellen ergeben u.a. die folgenden Funktionen:

1. $V_1(n) = 1.23n \log n$
2. $V_2(n) = 11.16n - \mathbf{478} \log n$
3. $V_3(n) = 9.80n$
4. $V_4(n) = 0.\mathbf{000584}n^2 + 8.05n$

Die Schätzfehler sind in Abb. 6 abgebildet. Die Funktionen wurden erneut nur an die ersten 30 Werte ($n \leq 4000$) angepasst und mit allen beobachteten Werten getestet.

[2] Ein Beispieldokument für die o.g. beschriebene lineare Regression steht auf der Sortkomp Projektseite [Wac11] zum Download bereit.

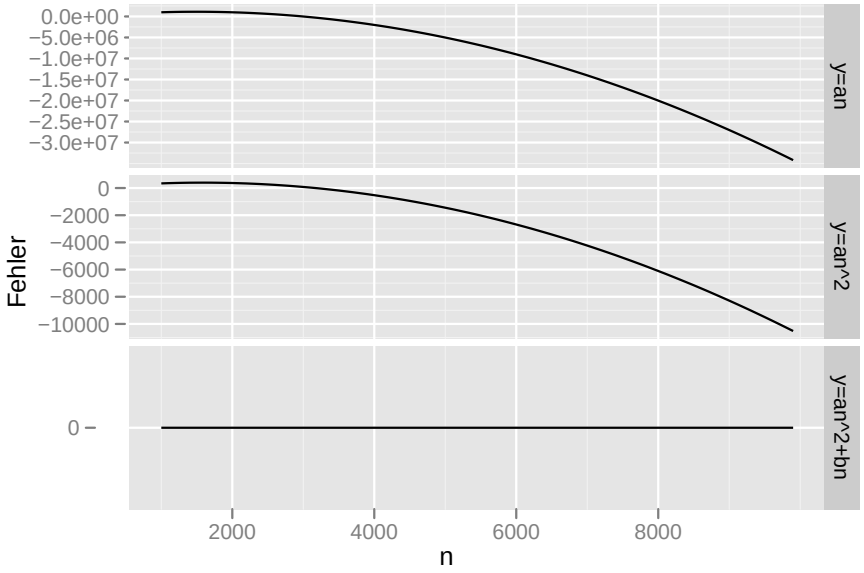

Figure 5: Bubblesort: Fehler bei der Schätzung der Anzahl der Vergleiche mit verschiedenen Modellen

Auffällig sind die Koeffizienten in V_2 und V_4. Bei V_4 ist eine Überanpassung offensichtlich: Für $n \leq 4000$ ist die quadratische Funktion präziser, als eine einfache Gerade, allerdings verschlechtert sich dadurch die Genauigkeit für große n leicht.

Ein erkennbarer Unterschied zwischen den Modellen V_1 und V_2 zeigt sich erst bei großen n. Durch das geringe Wachstum des Logarithmus werden Modellfehler erst sehr spät erkennbar. Um logarithmisches Wachstum zu erkennen, müssen genügend Messwerte in unterschiedlichen Größenordnungen vorliegen.

6 Zusammenfassung und Ausblick

Durch Experimente können wesentliche Inhalte des Themengebietes Zeitkomplexitätsbestimmung durch Schüler explorativ erarbeitet werden. Die Erfassung und der Export von Messwerten für verschiedene Sortierverfahren kann durch die Nutzung eines Frameworks ohne zusätzlichen Zeitaufwand durchgeführt werden. Lediglich die Sortieralgorithmen müssen von den Schülern implementiert werden.

Messwerte werden in eine Tabellenkalkulation exportiert und können dort von den Schülern weiter analysiert werden. Hierzu bieten sich zunächst Diagramme der Daten an. In einem weiteren Schritt der funktionale Zusammenhang der Funktionsgröße zur Operationsanzahl genauer untersucht werden. Schüler passen hierzu verschiedene Funktionsmodelle an die erfassten Daten an und bewerten diese auf Plausibilität. Die hierzu nötige lineare Regression ist in der Tabellenkalkulation bereits implementiert und kann als Black Box genutzt werden.

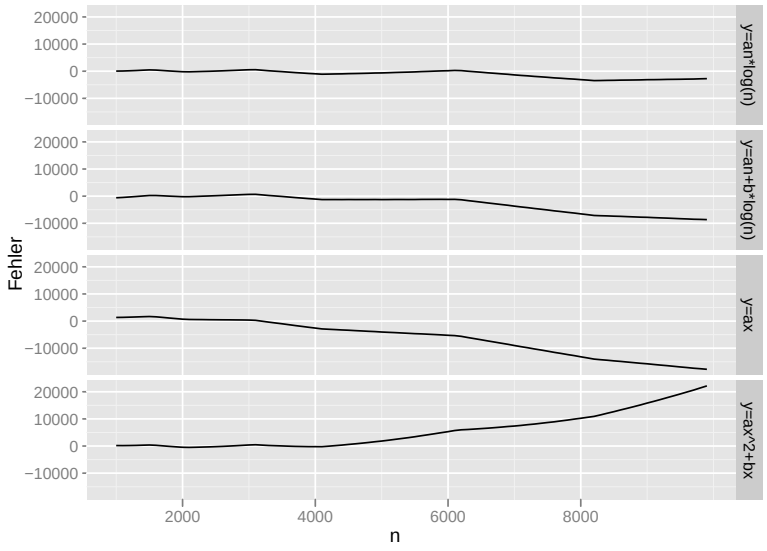

Figure 6: Quicksort: Fehler bei der Schätzung der Anzahl der Vergleiche mit verschiedenen Modellen

Nach einer Erarbeitung durch die Schüler können die gewonnenen Ergebnisse punktuell durch eine formale Analyse im Unterricht verifiziert werden.

Der hier vorgestellte Ansatz zur Analyse der Zeitkomplexität von Algorithmen wird in den Didaktik Übungen für Studierende des Lehramts an der TU Darmstadt erfolgreich angewendet. Studierende, die nicht zeitgleich Mathematik als zweites Fach studieren, haben erfahrungsgemäß sehr große Schwierigkeiten bei der formalen Analyse der Zeitkomplexität von Algorithmen. Ein handlungsorientierter experimenteller Ansatz führt zu ersten selbstgewonnenen Erkenntnissen, die dann um weitere formale Aspekte ergänzt werden. So wird beispielsweise bei der Bestimmung der Zeitkomplexität der rekursiven Bestimmung der Fibonacci Zahlen durch Regressionsrechnung eine obere und untere asymptotische Schranke bestimmt. Diese werden im im Anschluss durch vollständiger Induktion bewiesen.

References

[Fot03] Fothe, Michael: Zeitverhalten von Sortierverfahren - Beispiele für experimentelles Arbeiten im Informatikunterricht. In: *INFOS'03*, 2003, 111-120

[Sch93] Schwill, Andreas: Fundamentale Ideen der Informatik. In: *Zentralblatt für Didaktik der Mathematik 1*, 1993, S. 20–31

[Wac11] Wach, Christian: *Sortkomp – Ein Java Framework zur experimentellen Bestimmung der Zeitkomplexität von Sortieralgorithmen.* 2011. – "`http://code.google.com/p/sortkomp/`"

[Wic09] Wickham, Hadley: *ggplot2: elegant graphics for data analysis*. Springer New York, 2009 http://had.co.nz/ggplot2/book. – ISBN 978–0–387–98140–6