Rauf und runter

19. November 1999

Sortieralgorithmen und Permutationen. In dieser Übungsstunde wollen wir Sortieralgorithmen und Teilfolgen von Permutationen betrachten. Sortieralgorithmen erhalten als Eingabe eine gewisse Anzahl von Objekten in beliebiger Reihenfolge. Die Objekte können miteinander verglichen werden, und das Ziel des Sortieralgorithmus ist es, die Objekte in eine sortierte Reihenfolge zu bringen. Wenn wir beim Sortieren keine anderen „Tricks“ zulassen, können wir o.B.d.A.[1] annehmen, daß es sich bei den „Objekten“ um die Zahlen von 1 bis n handelt, die in einer beliebigen Reihenfolge gegeben sind. Der mathematische Begriff für eine solche Folge ist der Begriff der *Permutation*.

Definition 1 (Permutation) *Eine* Permutation π *ist eine bijektive Abbildung der Zahlen von* 1 *bis* n *in sich selbst:*

$$\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}.$$

Aus der Definition folgt unter anderem sofort, daß jede Permutation durch die Folge

$$\pi(1), \pi(2), \ldots \pi(n)$$

ihrer Werte eindeutig bestimmt ist. Es gibt somit genau $n!$ Permutationen der Zahlen von 1 bis n.

Quicksort ist nicht immer gut. Kommen wir zurück zu den Sortieralgorithmen. Das „Quick“ im Namen läßt annehmen, daß der Quicksort-Algorithmus[2] ein „schneller“ Algorithmus ist. Obwohl auch andere „optimale“ Sortieralgorithmen bekannt sind, wie zum Beispiel der sogenannte Mergesort, ist der Quicksort in vielen Fällen erheblich schneller. Allerdings hat der Quicksort den Nachteil, daß er sich auf gewissen Eingaben besonders schlecht verhält — solche besonders schlechten Eingaben gibt es für den Mergesort hingegen nicht. Der Standard-Quicksort verhält sich zum Beispiel auf bereits sortierten Eingaben besonders schlecht. Dieses Verhalten ist in den Abbildungen dargestellt: Wir wissen, daß „gute“ Sortieralgorithmen ungefähr $n \log n$ Operationen brauchen, um n Elemente zu sortieren. Man kann zeigen, daß der Quicksort im „Durchschnitt“ $n \log n$ Vergleiche braucht. Dies ist in der Abbildung 1 zu erkennen, wo die Laufzeit von Quicksort auf zufällig ausgewählten Permutationen dargestellt ist. In der Abbildung 2 ist das Verhalten von Quicksort auf bereits aufsteigend sortierten Eingaben dargestellt. Hier sieht man ein Verhalten, daß weit von dem eines optimalen Algorithmus abweicht: Quicksort hat

[1] o.B.d.A. steht für „ohne Beschränkung der Allgemeinheit“ und ist eine in der Mathematik häufig verwendete Abkürzung. Im Englischen wird oft die Abkürzung „w.l.o.g.“ für „without loss of generality“ verwendet.

[2] Quicksort ist einer der am besten analysierten Algorithmen, Er wurde 1962 von C. A. R. Hoare entwickelt.

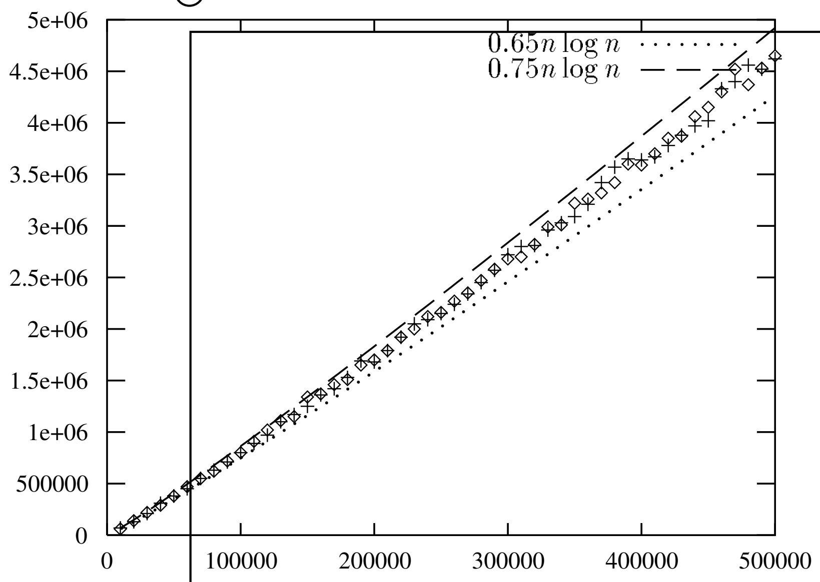


Abbildung 1: Laufzeiten von Quicksort auf zufällig gewählten Permutation

auf bereits sortierten Eingaben der Länge n eine Laufzeit von n^2. (Zum Vergleich der Laufzeiten sind die Laufzeiten für zufällige Permutation ebenfalls in der Abbildung dargestellt — es sind die Punkte „ganz unten“. Der Quellcode des zur Messung verwendeten C-Programms befindet sich am Ende des Übungsblattes.

Um die `qsort()`-Funktion der C-Bibliothek auszutricksen, müssen die Eingaben eine leicht andere Form haben, als beim Standard-Quicksort[3]: die Zahlen müssen nicht auf- oder absteigend sortiert sein, sondern die erste Hälfte der Zahlen aus der Permutation muß aufsteigend und die zweite Hälfte der Zahlen muß absteigend sortiert sein.

Auf- und absteigende Teilfolgen. Die obigen Beobachtungen sollen uns als Motivation dienen, bestimmte Teilfolgen von Permutation zu betrachten. Insbesondere möchten wir untersuchen, wielange die längste auf- bzw. absteigende Teilfolge einer Permutation ist.

Definition 2 *Eine* aufsteigende Teilfolge *einer Permutation π ist eine aufsteigende Folge von Indizes $i_1 < i_2 < \cdots < i_k$, so daß $\pi(i_1) < \pi(i_2) < \cdots < \pi(i_k)$.*

Mit den Methoden, die wir in der letzten Übungsstunde kennengelernt haben, können wir eine erste Abschätzung für die Länge der längsten auf- bzw. absteigenden Teilfolge finden.

[3]der Grund ist, daß `qsort()` den sogenannten Median-of-three-Quicksort verwendet

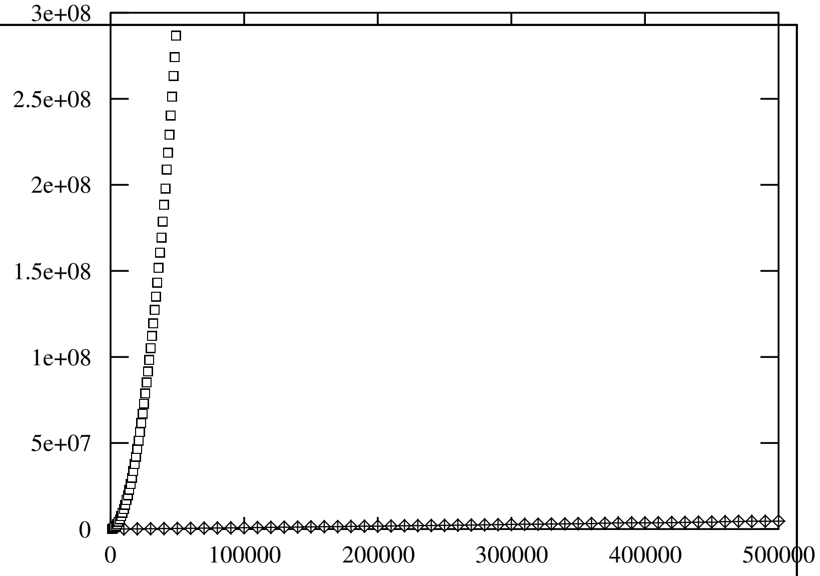
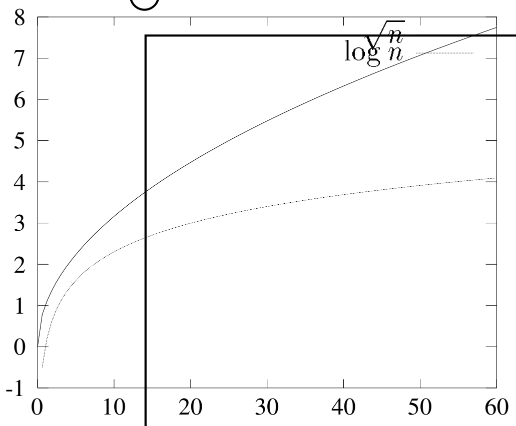


Abbildung 2: Laufzeiten von Quicksort auf bereits aufsteigend sortierten Eingaben

Satz 3 *Jede Permutation* π *der Zahlen von* 1 *bis* n *enthält eine aufsteigende oder eine absteigende Teilfolge der Länge* $\log n$.

Beweis. Diesen Satz können wir mit den Methoden beweisen, die wir in der letzten Übungsstunde kennengelernt haben. Wir konstruieren aus einer Permutation π der Zahlen von 1 bis n: Als Knoten nehmen wir die Zahlen von 1 bis n und fügen zwischen jedem Paar von Knoten i und j ($i < j$) eine blaue Kante, wenn $\pi(i) < \pi(j)$ und eine rote Kante wenn $\pi(i) > \pi(j)$. Aus einer aufsteigenden Teilfolge der Länge k erhalten wir mit dieser Konstruktion einen vollständigen blauen Teilgraph, aus einer absteigenden Folge der Länge k einen vollständigen roten Teilgraph. Aus der letzten Übungsstunde wissen wir, daß $R(s, s) \leq 4^2$ ist, d.h. daß wir in jedem Graph mit 4^s Knoten auf jeden Fall einen roten oder blauen vollständigen Teilgraph mit s Knoten finden. Wenn wir jetzt $s := \log n$ setzen, erhalten wir $R(\log n, \log n) \leq 4^{\log n}$. Jeder Graph mit n Knoten enthält also eine auf- oder absteigende Teilfolge der Länge $\log_4 n$. □

In der Abbildung auf der rechten Seite sind die Funktionen $\sqrt{n}$ und $\log n$ dargestellt. Man sieht sofort, daß $\log n$ viel langsamer wächst als $\sqrt{n}$. Im nächsten Satz werden wir zeigen, daß das obige Resultat stark verbessert werden kann, wir können nämlich zeigen, daß jede Permutation der Zahlen von 1 bis n sogar eine auf- oder absteigende Teilfoge der Länge $\sqrt{n}$ enthält.



Satz 4 *Jede Permutation π der Zahlen von 1 bis n enthält eine aufsteigende oder eine absteigende Teilfolge der Länge $\geq \sqrt{n}$.*

Beweis. Wir stellen die Permutation als Folge ihrer Werte $\pi(1), \pi(2), \ldots, \pi(n)$ dar und ordnen jedem der Elemente $\pi(i)$ der Folge ein Paar von Zahlen dar, dessen erster Eintrag die Länge der längsten aufsteigenden Folge ist, die mit $\pi(i)$ endet und dessen zweiter Eintrag die Länge der längsten aufsteigenden Folge ist, die mit $\pi(i)$ anfängt. Man überlegt sich leicht, daß es keine zwei Elemente der Folge geben kann, der dasselbe Paar zugeordnet ist. Wenn die längste auf- oder absteigende Teilfolge weniger als $\sqrt{n}$ Elemente enthalten würde, gäbe es insgesamt weniger als n Möglichkeiten für die Paare, und mindestens ein Paar müßte somit mehrmals verwendet werden. *Widerspruch!* $\square$

Übungen.

1. In Abbildung 1 ist das Verhalten von Quicksort auf zufälligen Permutationen dargestellt, wobei „zufällig“ bedeutet, daß aus den $n!$ Permutation jede mit gleich großer Wahrscheinlichkeit ausgewählt werden kann. Beschreiben Sie einen Algorithmus, der eine zufällige Permutation erzeugt!

2. Überlegen Sie sich einen Algorithmus, der in polynomieller Zeit (polynomiell in n) die längste aufsteigende Teilfolge einer Folge von n Zahlen (d.h. einer Permutation) bestimmt!

Bibliographie. Weitere Eigenschaften von Permutationen werden unter anderem in [SF96] untersucht. Dort findet man auch viele weiterführende Informationen zur Analyse von Algorithmen.

Literatur

[SF96] Robert Sedgewick and Philippe Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley Publishing Company, 1996.