Master-Theorem II

Das Master-Theorem II geben wir hier ohne Beweis an:

Es sei $r > 0$ und die Zahlen α_i seien positiv (für alle i) und erfüllen $\sum_{i=1}^{r} \alpha_i < 1$.

Wenn die rekursive Funktion t die Ungleichung

$$t(n) \leq \left(\sum_{i=1}^{r} t(\lceil \alpha_i \cdot n \rceil) \right) + c \cdot n$$

für ein $c > 0$ erfüllt, dann ist $t(n) \in O(n)$.

Sortieren

Sortieren ist eine der häufigsten und wichtigsten Aufgaben, die von Algorithmen auszuführen sind.

Die üblichen Sortieralgorithmen sind grundsätzlich *vergleichsbasiert*, d.h. dass einerseits die Grundoperation, die zum Erfolg führt, der Vergleich zweier Elemente ist – aber auch, dass zur Messung des Aufwands nur die Anzahl der Vergleichsoperationen heranzuziehen ist.

Für vergleichsbasierte Sortieralgorithmen gibt es eine beweisbare untere Schranke für den Aufwand:

Jeder vergleichsbasierte Sortieralgorithmus, der n Elemente sortiert, braucht im worst case $\Omega(n \log n)$ Vergleiche.

Beweis der unteren Schranke

Die entscheidende Beobachtung ist folgende:

Zu Beginn eines Ablaufs jedes beliebigen Sortieralgorithmus, der eine Folge von n Elementen sortieren soll, gibt es $n!$ mögliche Vorsortierungen, die der Algorithmus unterscheiden können muss.

Jeder Vergleich teilt die Menge der möglichen Vorsortierungen in zwei Teile auf, von denen durch das Vergleichsergebnis einer – im worst case bei ungleicher Verteilung der größere – zur weiteren Verarbeitung ausgewählt wird.

Also kann der Algorithmus im worst case frühestens nach $\log(n!)$ vielen Vergleichen die komplette und korrekte Sortierung finden.
Dass $\log(n!)$ in $\Omega(n \log n)$ ist ein bekannter Sachverhalt.

Wir werden das im 2. Teil der Vorlesung (Diskrete Strukturen) auch noch beweisen.

Quicksort

Den Quicksort-Algorithmus haben wir schon vorgestellt.
Die zentrale Idee: Wähle ein *Pivot-Element* aus dem zu sortierenden Array, teile in zwei Teile, die rekursiv sortiert werden.
Einer dieser Teile beinhaltet die Elemente, die kleiner als das Pivot-Element sind, der andere die, die größer sind.

Bei geschickter Implementierung landet das Pivot-Element jeweils direkt an seinem Platz, die übrigen $n-1$ Elemente werden rekursiv weiter verarbeitet, jedes in einem der beiden Teile.

Dass die worst-case Laufzeit des Quicksort-Algorithmus in jedem Fall quadratisch ist, haben wir uns schon überlegt.
Hieran ändert auch eine geschickte Strategie zur Pivot-Wahl nichts – obwohl in der Praxis die „Median-aus-drei“ Strategie sehr gut funktioniert.

Quicksort-Varianten

Die Median-aus-drei Variante funktioniert so:
Wenn für den Teil $a[links, \ldots, rechts]$ ein Pivot-Element festzulegen ist, wählt man aus drei Elementen – etwa $a[links]$, $a[rechts]$ und $a[(links + rechts)/2]$ – das der Größe nach mittlere.

Es ist ein einfaches Rechenbeispiel, nachzuweisen, dass diese Strategie im worst case nach wie vor eine quadratische Anzahl Vergleiche benötigt.

Um ein „gutes“ Pivot-Element in jedem Schritt zu erzwingen, muss man im noch zu bearbeitenden Teil-Array $a[links, \ldots, rechts]$ eine komplette Medianberechnung durchführen. Diese ist allerdings vergleichsweise aufwändig. Daher findet diese Variante in der Praxis kaum Verwendung.

Quicksort: average case Analyse

Wie kann man den Quicksort-Algorithmus in seinem Durchschnittsverhalten analysieren?
Wir gehen von einer Standardvariante aus, die bei einem Aufruf mit dem Teil-Array $a[\ell, \ldots, r]$ (hier steht ℓ für *links* und r für *rechts*) ein zufälliges Pivot-Element auswählt, wobei Gleichverteilung unter den in Frage kommenden Elementen vorausgesetzt wird.

Bei jeder Aufteilung von m Elementen ist eines von ihnen das Pivot-Element, alle anderen (also $m-1$ Elemente) werden mit dem Pivot verglichen. Andere Vergleiche gibt es nicht.

Nun stellt sich die Frage, wie viele Vergleiche insgesamt im Ablauf der Quicksort-Prozedur auf dem Gesamtarray $a[1, \ldots, n]$ durchschnittlich ausgeführt werden müssen.