Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dr. Hanjo Täubig
Tobias Lieber

Sommersemester 2011
Übungsblatt 5
14. Juni 2011

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: In der jeweiligen Tutorübung

Hausaufgabe 1

Implementieren Sie in der Klasse `UIsmsArray` den MergeSort-Algorithmus, in der Funktion `sort`, der die Elemente in dem Feld `A` sortiert.
Verwenden Sie für Ihre Implementierung die auf der Übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die Klasse `UIsmsArray`.
Achten Sie bei der Abgabe Ihrer Aufgabe darauf, dass Ihre Klasse `UIsmsArray` heißt und auf den Rechnern der Rayhalle (`rayhalle.informatik.tu-muenchen.de`) mit der bereitgestellten Datei `main_m` kompiliert werden kann. Anderenfalls kann eine Korrektur nicht garantiert werden. Achten Sie darauf, dass Ihr Quelltext ausreichend kommentiert ist.
Schicken Sie die Lösung per Email mit dem Betreff `[GAD] Gruppe <Gruppennummer>` an Ihren Tutor.

Lösungsvorschlag

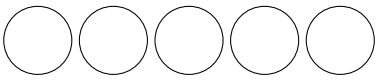
Siehe Übungswebseite.

Hausaufgabe 2

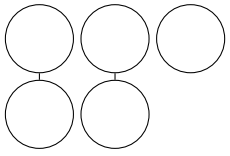
Geben Sie ein Verfahren an, um 5 Elemente mit 7 Vergleichen zu sortieren.

Lösungsvorschlag

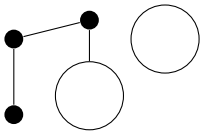
Wir beginnen mit 5 ungeordneten Elementen:



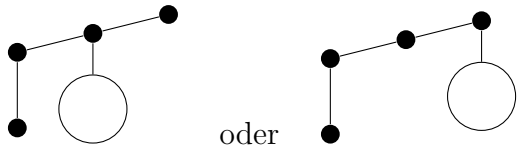
Mittels zwei Vergleichen stellen wir eine Ordnung unter jeweils zwei Elementen her:



Vergleichen wir die beiden größten Elemente so erhalten wir eine Ordnung unter drei Elementen:



Mit zwei weiteren Vergleichen können wir das bisher noch unverglichene Element sicher in die drei korrekt geordneten Elemente einfügen (Binäre Suche). Wir erhalten Eine der beiden Situationen:



In beiden Fällen können wir mit zwei Vergleichen das „weiße“ Element durch die noch zwei verbleibenden Vergleiche korrekt einsortieren.

Aufgabe 1

Veranschaulichen Sie das Vorgehen des RadixSort-Algorithmus anhand der folgenden Wörter:

`alt, hort, kalt, bar, sport, spalt, ort, stall, spart, speer, bart`

Benutzen Sie die lexikographische Sortierung (Dabei sei das Leerzeichen ␣ kleiner als jeder andere Buchstabe). Es reicht aus, wenn sie als Alphabet die Buchstaben ␣, a, b, e, h, k, l, o, p, r, s, t verwenden und daher die Schlüssel 0,...,11 verwenden.

Lösungsvorschlag

- Ergänzen wir die Wörter, wie Zahlen, vorne mit dem kleinsten lexikoraphischen Zeichen erhalten wir folgendes Sortierung:

  Wir Sortieren nach dem letzten Zeichen.

␣	a	b	e	h	k	l	o	p	r	s	t
						stall			bar		alt
									speer		hort
											kalt
											sport
											spalt
											ort
											spart
											bart

  Wir erhalten daraus das Zwischenresultat: `stall, bar, speer, alt, hort, kalt, sport, spalt, ort, spart, bart`.

  Sortieren nach dem vorletzten Zeichen:

␣	a	b	e	h	k	l	o	p	r	s	t
	bar		speer			stall			hort		
						alt			sport		
						kalt			ort		
						spalt			spart		
									bart		

Nach dem zweiten Zeichen: `bar, speer, stall, alt, kalt, spalt, hort, sport, ort, spart, bart`.

Sortieren nach dem drittletzten Zeichen:

␣	a	b	e	h	k	l	o	p	r	s	t
	stall	bar	speer				hort				
	alt						sport				
	kalt						ort				
	spalt										
	spart										
	bart										

Nach dem dritten Zeichen: `stall, alt, kalt, spalt, spart, bart, bar, speer, hort, sport, ort`.

Sortieren nach den 2. Zeichen:

␣	a	b	e	h	k	l	o	p	r	s	t
alt	bart			hort	kalt			spalt			stall
bar								spart			
ort								speer			
								sport			

Vor dem letzten Sortieren: `alt, bar, ort, bart, hort, kalt, spalt, spart, speer, sport, stall`.

Sortieren nach dem ersten Zeichen:

␣	a	b	e	h	k	l	o	p	r	s	t
alt										spalt	
bar										spart	
ort										speer	
bart										sport	
hort										stall	
kalt											

Ergibt: `alt, bar, ort, bart, hort, kalt, spalt, spart, speer, sport, stall`.

- Ergänzen wir die Wörter hinten mit dem kleinsten Zeichen, erhalten wir die korrekte lexikographische Sortierung:

  Sortieren nach dem letzten Zeichen:

␣	a	b	e	h	k	l	o	p	r	s	t
alt						stall			speer		sport
hort											spalt
kalt											spart
bar											
ort											
bart											

Wir erhalten das Zwischenergebnis `alt, hort, kalt, bar, ort, bart, stall, speer, sport, spalt, spart`.

Sortieren nach dem vorletzten Zeichen:

␣	a	b	e	h	k	l	o	p	r	s	t
alt			speer			stall			sport		hort
bar						spalt			spart		kalt
ort											bart

Daraus erhalten wir das Zwischenresultat: `alt, bar, ort, speer, stall, spalt, sport, spart, hort, kalt, bart`.

Sortieren nach dem dritten Zeichen:

␣	a	b	e	h	k	l	o	p	r	s	t
stall			speer			kalt	sport		bar		alt
spalt									hort		ort
spart									bart		

Daraus erhalten wir das Zwischenresultat: `stall, spalt, spart, speer, kalt, sport, bar, hort, bart, alt, ort`.

Sortieren nach dem zweiten Zeichen:

␣	a	b	e	h	k	l	o	p	r	s	t
	kalt					alt	hort	spalt	ort		stall
	bar							spart			
	bart							speer			
								sport			

Daraus erhalten wir das Zwischenresultat: `kalt, bar, bart, alt, hort, spalt, spart, speer, sport ,ort, stall`.

Sortieren nach dem ersten Zeichen:

␣	a	b	e	h	k	l	o	p	r	s	t
	alt	bar		hort	kalt		ort			spalt	
		bart								spart	
										speer	
										sport	
										stall	

Daraus erhalten wir das Ergebnis: `alt, bar, bart, hort, kalt, ort, spalt, spart, speer, sport, stall`.

Aufgabe 2

Wir haben uns in der Vorlesung beim Beweis, dass die Laufzeit von MergeSort in $\mathcal{O}(n \log n)$ liegt, auf das Mastertheorem berufen.
Zeigen Sie ohne Benutzung des Mastertheorems, dass die geschlossende Darstellung der Rekursionsformel

$$\begin{aligned} T(n) &= T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \Theta(n) \\ T(1) &= \Theta(1) \end{aligned}$$

in $\mathcal{O}(n \log n)$ liegt. Sie dürfen vereinfachend annehemen, dass n eine Zweierpotenz ist.

Lösungsvorschlag

Wie in der Aufgabe angegeben, gehen wir davon aus, dass n von der Form 2^m ist. Wir berechnen die Laufzeit von $T(n)$ durch widerholtes anwenden der Definition.

$$\begin{aligned} T(n) = cn + 2T(\frac{n}{2}) = cn + 2\left(c\frac{n}{2} + 2T(\frac{n}{4})\right) = cn + cn + 4T(\frac{n}{4}) = \\ cn + cn + \cdots + cn + 2^m T(\frac{n}{2^m}) = cn + cn + \cdots + cn + 2^m T(1) = \\ cn + cn + \cdots + cn + nc = cn(\log n + 1) \in \mathcal{O}(n \log n) \end{aligned}$$

Die letzte Zeile folgt, da $m = \log n$.

Aufgabe 3 (9 Punkte)

Wir wollen in dieser Aufgabe den `QuickSort` Algorithmus näher analysieren.

a) Geben Sie eine Familie von Eingabearrays an, so dass der `QuickSort` Algorithmus für jedes n Laufzeit $\Omega(n^2)$ benötigt.

b) Zeigen Sie: Wird in dem Algorithmus als Pivot-Element das k-größte Element ausgewählt (wobei $k \in \mathbb{N}$ eine beliebige aber feste Konstante ist), so liegt die Worst-Case Laufzeit des modifizierten Algorithmus immer noch bei $\Omega(n^2)$.

   Ist $|A| < k$, so wird das Pivot-Element wie im Algorithmus der Vorlesung gewählt.

c) Beweisen Sie: Wählen wir als Pivot-Element das $\lfloor \frac{9}{10} n \rfloor$-größte Element aus `A` ($|A| = n$), so ist die Worst-Case Laufzeit $\mathcal{O}(n \log n)$ (wenn die Selektion des Pivot Elements in $\mathcal{O}(n)$ realisiert ist).

Lösungsvorschlag

a) Eine Familie von Eingabearrays, für die der QuickSort Algorithmus für jedes $n \in \mathbb{N}$ $\Omega(n^2)$ Zeit benötigt, ist die Familie der sortierten Arrays.

b) Da als Pivot-Element immer das k-größte Element ausgewählt wird, wird der rekursive Aufruf `quickSort(l,i-1)` genau $\lfloor \frac{n}{k} \rfloor$-Mal durchgeführt. In jedem rekursiven Aufruf

müssen alle $r - l$ Elemente mit dem Pivot-Element verglichen werden. Es müssen also schon durch den ersten rekursiven Aufruf mindestens

$$\sum_{j=1}^{\lfloor \frac{n}{k} \rfloor} kj \geq k \sum_{j=1}^{\frac{n}{k}-1} j = \frac{n(\frac{n}{k}-1)}{2} = \frac{n^2}{2k} - \frac{n}{2} \in \Omega(n^2)$$

Vergleiche bzw. Rechenoperationen durchgeführt werden.

c) In jedem rekursiven Schritt ist die Anzahl der Rechenoperationen durch das Vergleichen des gesamten Arrays mit dem, bzw. den Pivot-Element/en beschränkt (es gibt mehrere Pivot-Elemente ab der zweiten Stufe der Rekursion). Die Anzahl der Rechenoperationen ist also durch die Tiefe i der rekursiven Aufrufe und durch die Anzahl der zu sortierenden Elemente beschränkt ($\mathcal{O}(ni)$).

Sei n_i die Anzahl der Elemente die noch zu sortieren sind nach dem i-ten (ersten) rekursiven Aufruf (bzw. r_i und l_i die rechten und linken Grenzen). Die Selektion des Pivot-Elementes wird so oft durchgeführt, bis sich die Rekursion trivialisiert, also $r_i \leq l_i + 1$ gilt. Wir sind also an dem größten i interessiert so dass $r_i \geq l_i + 1$ gilt(denn so lange diese Gleichung noch gilt muss noch rekursiv abgestiegen werden).

Dies ist äquivalent mit der Forderung

$$r_i - l_i \geq 1 \Leftrightarrow n_i \geq 1. \tag{1}$$

Da in jedem Rekursionsschritt (wir betrachten nur den ersten Rekursionsschritt) die Eingabegröße um $\frac{9}{10}$ verringert wird, ist Ungleichung (1) äquivalent zu der Frage, für welches i die Ungleichung $\left(\frac{9}{10}\right)^i n \geq 1$ erfüllt ist. Daraus ergibt sich:

$$\left(\frac{9}{10}\right)^i n \geq 1 \Leftrightarrow n \geq \left(\frac{10}{9}\right)^i \Leftrightarrow i \leq \log_{\frac{10}{9}} n \Leftrightarrow i \in \mathcal{O}(\log n)$$

Insgesamt ergibt sich also für diese Variante des QuickSort Algorithmus eine Laufzeitschranke von: $\mathcal{O}(n \log n)$.