Datenstrukturen und Algorithmen: Blatt 5

Bernhard Dietrich (6256800)
Lars Fernhomberg (6256030)
Sebastian Kniesburges (6257120)
Marcus Köthenbürger (6258550)

Übungsgruppe 11
Mittwoch, 11:00-13:00 Uhr in D1.303
Matthias Ernst

Aufgabe	1	2	3	Σ	Korrektor
Punkte					
von	4	4	12	20	

Aufgabe 1

1. Was ist in O-Notation die Laufzeit von Quicksort, wenn n identische Zahlen sortiert werden sollen?
   Durch den in der Vorlesung besprochenen Partitionsalgorithmus (ohne Optimierungen) wird bei n identischen Zahlen immer nur das letzte Element abgespalten, so dass der Partitionierungsalgorithmus anschließend auf ein Element und auf $n-1$ Elemente angewandt wird.
   Es ergibt sich somit
   $$n+(n-1)+(n-2)+(n-3)+...+3+2+1=\sum_{i=1}^{n} i=\frac{n(n+1)}{2}=\frac{n^2+n}{2}$$
   Somit wird deutlich, dass der Algorithmus eine quadratische Laufzeit $O(n^2)$ hat, wenn n identische Zahlen sortiert werden sollen.
2. Was ist in O-Notation die Laufzeit von Quicksort, wenn die Eingabezahlen absteigend sortiert sind?
   Bei einer absteigenden Sortierung spaltet der in der Vorlesung besprochene Partitionsalgorithmus (ohne Optimierungen) ebenfalls nur ein, in diesem Fall das erste, Element ab, so dass ebenfalls eine quadratische Laufzeit $O(n^2)$ resultiert (vgl. mit Teilaufgabe 1).

Aufgabe 2

Angenommen die Funktion PARTITION teilt Teilarrays der Größe n immer in zwei Teilarrays auf, von denen eins $(1-\alpha)n$ Elemente und das andere αn Elemente enthält, hierbei ist $\frac{1}{2} \le \alpha < 1$. Zeigen Sie, dass dann jedes Blatt im Rekursionsbaum für Quicksort höchstens Tiefe $-\frac{\log(n)}{\log(\alpha)}$ und mindestens $-\frac{\log(n)}{\log(1-\alpha)}$ besitzt. In der gesamten Aufgabe dürfen Auf- bzw. Abrundungen ignoriert werden.

Der Maximum-Fall tritt ein, wenn die Funktion PARTITION die Arrays in einen Teilbaum mit lediglich einem Element und in einen Teilbaum mit $a-1$ Elementen, wobei a die Anzahl Elemente des an die Funktion PARTITION übergebenen Teilarrays bezeichnet, teilt.
…
Der Mindest-Fall tritt ein, wenn der Baum optimal verteilt ist und somit jeder Teilbaum gleich groß und die Blätter alle in der gleichen Tiefe sind. Für einen solchen binären Baum haben

wir in der Vorlesung ermittelt, dass der Baum eine Tiefe von $\log(n)$ hat. Da außerdem $\log(1-\alpha) = \log\left(1-\frac{1}{2}\right) = \log\left(\frac{1}{2}\right) = -1$ folgt somit $-\frac{\log(n)}{\log(1-\alpha)}$.

Aufgabe 3

Betrachten Sie den folgenden alternativen Algorithmus zur Partitonierung eines Teilarrays, wie er von Quicksort benutzt wird.

HOARE-PARTITION(A,p,r)

```
1     x ← A[p]
2     i ← p − 1
3     j ← r + 1
4     while i < j
5           do repeat j ← j - 1
6                 until A[j] ≤ x
7              repeat i ← i + 1
8                 until A[i] ≥ x
9              if i < j
10                then A[i] ↔ A[j]
11    return j
```

Zeigen Sie

1. Der Algorithmus greift nie auf Elemente $A[i]$, $A[j]$ außerhalb des Teilarrays $A[p...r]$ zu.

   Die Variable i wird mit $p-1$ und die Variable j mit $r+1$ initialisiert.
   Die Variable j wird zwar mit $r+1$ initialisiert, fällt allerdings bereits während des ersten Schleifendurchlaufs und vor dem ersten Zugriff auf $A[j]$ auf r, sodass die obere Schranke nicht überschritten wird. Das j kann im weiteren Verlauf des Algorithmus nur solange fallen, bis es auf ein Element trifft, welches kleiner oder gleich x ist.
   Da $x = A[p]$ stoppt das herunterzählen von j spätestens bei $j = p$. Wenn $j = p$ gilt, ist die Abbruchbedingung der while-Schleife erfüllt, da die Variable i auf jeden Fall mindestens den Wert p hat (siehe weiter unten) und somit $i < j$ nicht mehr erfüllt ist.
   Die Variable i wird bereits beim ersten Durchlauf und vor Zugriff auf das Elemente $A[i]$ auf den Wert $i+1 = p$ gesetzt (und somit kann j auch nie unter p fallen), sodass keine Elemente als $A[p]$ angesprochen werden können. Wenn $A[p]$ das größte Objekt im Array ist, durchläuft die erste „repeat-Schleife“, welche die Variable j steuert, das Array mit dem Resultat $j = i$. Da somit die Bedingung der while-Schleife nicht mehr erfüllt ist und diese abbricht, kann die Variable i in diesem Fall nicht mehr weiter wachsen und verbleibt auf dem Wert der unteren Schranke. Sollte die while-Schleife mehrmals durchlaufen werden, so bedeutet dies, dass es Elemente zwischen j und r gibt, die größer bzw. gleich dem Element $A[p]$ sind, so dass die Variable i nicht weiter als bis zu diesen Elementen laufen kann, da die Bedingung der zweiten Repeat-Schleife hier den Schleifendurchlauf abbricht. Wenn nun der Fall $i > j$ eintritt wird die komplette Schleife terminiert.
2. Für den Wert j, den Algorithmus HOARE-PARTITION ausgibt, gilt $p \le j \le r-1$.

   Wie bereits in Teilaufgabe 1 angesprochen, muss $p \le j$ gelten, da, sofern $x = A[p]$ das größte Element im Teilarray ist, die erste repeat-until Schleife bis zum

Wert $j = p$ zählt und dort – da ihre Abbruchbedingung erfüllt ist – abbricht.
Wenn x nicht das größte Element im Teilarray ist, ist j größer als p .
$j \leq r - 1$ folgt aus …

3. Am Ende des Algorithmus HOARE-PARTITION und für den Ausgabewert j gilt: Alle Elemente in $A[p...j]$ sind höchstens so groß wie die Elemente in $A[j+1...r]$.
   Die erste repeat-until-Schleife durchsucht das gegebene Array absteigend vom letzten Element an nach Elementen, deren Wert kleiner oder gleich dem Wert des ersten Element des gegebenen Arrays ist und bricht ab, wenn ein solches Element gefunden wurde. Die zweite repeat-until-Schleife verfährt ähnlich und sucht aufsteigend vom ersten Element des anfänglichen Arrays an nach Elementen, deren Wert größer oder gleich dem Wert des ersten Elements des anfänglichen Arrays ist und bricht ebenfalls ab, wenn dieser Fall eintritt. Nach beiden repeat-until-Schleifen ist nun ein Element im Teilarray $A[p...i]$ bekannt, dessen Wert kleiner oder gleich dem Wert des ersten Elements des anfänglichen Arrays ist und im Teilarray $A[j...r]$ ist ein Element bekannt, dessen Wert größer als der Wert des ersten Elements des anfänglichen Arrays ist. Wenn $i < j$ ist, werden nun diese beiden Elemente vertauscht, so dass im Teilarray $A[p...i]$ nur Elemente vorhanden sind, deren Werte kleiner oder gleich dem Wert des ersten Elements des anfänglichen Arrays sind und im Teilarray $A[j...r]$ nur Elemente vorhanden sind, deren Werte größer oder gleich dem Wert des ersten Elements des anfänglichen Arrays sind. Ist nun $i > j$ so bedeutet dies, dass alle Elemente einmal mit dem ersten Element des anfänglichen Arrays verglichen wurden und somit die Werte der Elemente $A[p...j]$ höchstens so groß wie die Werte der Elemente $A[j+1...r]$ sind.