RWTH Aachen **SS 2016**
Lehrgebiet Theoretische Informatik Blatt 6
Rossmanith–Hensel–Kuinke–Sánchez 30. Mai 2016

Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T17

Gegeben ist folgende Zahlenfolge: 8, 6, 3, 7, 4, 2, 20, -45

a) Wieviele Inversionen hat sie?

b) Wieviele Läufe hat sie?

c) Was macht Quicksort in der ersten Partitionierungsphase daraus?

d) Was macht Mergesort in der letzten Mischphase?

Aufgabe T18

In der Vorlesung wurde Quicksort auch iterativ mit Hilfe eines expliziten Stacks implementiert. Da Speicherverbrauch immer ein wichtiger Faktor ist, sind wir an der maximalen Höhe dieses Stacks interessiert: Finden Sie ein Beispiel, in welchem die gegebene Quicksort-Implementation $\Omega(n)$ Paare gleichzeitig im Stack speichert. Überlegen Sie dann, wie der Algorithmus abgeändert werden kann, um diesen schmerzhaften Speicherverbrauch deutlich zu senken.

Aus Effizienzgründen bestimmt die vorliegende Implementation zunächst das minimale Element des Eingabearrays und vertauscht es mit dem ersten Element desselben, die verbleibenden Elemente werden dann wie gehabt sortiert.

```
public void quicksort() {
  Stack⟨Pair⟩ stack = new Stack⟨Pair⟩();
  stack.push(new Pair(1, a.length − 1));
  int min = 0;
  for(int i = 1; i < a.length; i++)
    if(a[i] < a[min]) min = i;
  int t = a[0]; a[0] = a[min]; a[min] = t;
  while(!stack.isEmpty()) {
    Pair p = stack.pop();
    int l = p.first(), r = p.second();
    int i = l − 1, j = r, pivot = j;
    do {
      do { i++; } while(a[i] < a[pivot]);
      do { j−−; } while(a[j] > a[pivot]);
      t = a[i]; a[i] = a[j]; a[j] = t;
    } while(i < j);
    a[j] = a[i]; a[i] = a[r]; a[r] = t;
    if(r − i > 1) stack.push(new Pair(i + 1, r));
    if(i − l > 1) stack.push(new Pair(l, i − 1));
  }
}
```

Die Analyse von Quicksort setzt jedoch voraus, daß jede mögliche Permutation der zu sortierenden Schlüssel gleich wahrscheinlich ist. Sind nach der obigen Veränderung der Eingabe alle Permutationen der verbleibenden Elemente immer noch gleich wahrscheinlich?

Aufgabe T19

Die **while**-Schleife des Insertionsorts wird dann betreten, wenn $a[i]$ nicht das größte Element im Unterarray $a[0], ..., a[i]$ ist.

procedure *insertionsort*(n) :
  for $i = 1, ..., n-1$ **do**
    $j := i$;
    while $j \geq 1$ **and** $a[j-1] > a[j]$ **do**
      vertausche $a[j-1]$ *und* $a[j]$;
      $j := j - 1$
    od
  od

Nehmen Sie an das Array sei mit verschiedenen Zahlen in zufälliger Reihenfolge gefüllt. Für wieviele Durchgänge der **for**-Schleife wird die **while**-Schleife im Erwartungswert mindestens einmal betreten?

Aufgabe H16 (2+3+3 Punkte)

```
for(int i = 0; i < n; i++) {
    int m = n - 1;
    for(int j = i + 1; j < n; j++) {
        if(a[j] < a[m]) {
            m = j;
        }
    }
    int t = a[i]; a[i] = a[m]; a[m] = t;
}
```

a) Obiges Programm soll ein Array $a[0],\ldots,a[n-1]$ sortieren. Versuchen Sie zu beschreiben, wie der Algorithmus funktionieren sollte.

b) Leider enthält das Programm einen kleinen Fehler. Geben Sie eine Eingabe an, auf welcher das Programm versagt.

c) Finden Sie den Fehler und korrigieren Sie ihn mit möglichst wenigen Änderungen.

Aufgabe H17 (4+0+3+6 Punkte)

Wir untersuchen in dieser Aufgabe, wie man binäre Suchbäume zum Sortieren verwenden kann.

a) Entwerfen Sie einen Algorithmus, der in $O(n)$ Schritten die n Schlüssel aus einem binären Suchbaum in aufsteigender Reihenfolge ausgeben kann.

b) Beschreiben Sie, wie man mit binären Suchbäumen möglichst effizient sortieren kann.

c) Wie schnell ist Ihr Verfahren aus b) im worst-case mit gewöhnlichen binären Suchbäumen, mit AVL-Bäumen und mit Splay-Bäumen?

d) Beantworten Sie Frage c) auch für den Spezialfall, daß die Eingabe bereits sortiert bzw. umgekehrt sortiert ist.