RWTH Aachen **SS 2018**
Lehrgebiet Theoretische Informatik Blatt 8
Rossmanith–Dreier–Kuinke 18.6.2018

Übung zur Vorlesung Algorithmen und Datenstrukturen

Aufgabe T16
Gegeben seien die Funktionen $f(n)$ und $g(n)$. Beweisen oder widerlegen Sie:

a) $O(f) \cdot O(g) = O(f \cdot g)$

b) Falls $g = O(f)$ und $h = O(f)$ dann gilt auch $g = O(h)$

c) $\sqrt{2n(n+2)(n-1)} = \Theta(n^{3/2})$

Lösungsvorschlag:
a)
Die Aussage ist wahr. Sei $f_* = O(f)$ und $g_* = O(g)$, dann exitieren $c_1, c_2 \in \mathbf{R}^+$ und $n_1, n_2 \in \mathbf{N}$, so dass für alle $n > n_1$, $|f_*(n)| \leq c_1|f(n)|$ und für alle $n > n_2$, $|g_*(n)| \leq c_2|g(n)|$ gilt. Sei $n_0 = \max(n_1, n_2)$ und $c_0 = c_1 \cdot c_2$. Dann gilt für alle $n > n_0$, $|f_*(n)| \cdot |g_*(n)| \leq c_1|f(n)| \cdot c_2|g(n)| = c_0(|f(n) \cdot g(n)|) = O(f \cdot g)$ gilt.

b)
Die Aussage ist falsch. Es gilt z.b. $n^2 = O(n^3)$ und $n = O(n^3)$ aber nicht $n^2 = O(n)$.

c)
Die Aussage ist wahr. Wir schätzen die gegebene Funktion zuerst durch eine obere Schranke ab. Für $n \geq 2$ gilt

$$\sqrt{2n(n+2)(n-1)} \leq \sqrt{2n \cdot 2n \cdot n} = \sqrt{4n^3} = 2n^{3/2}.$$

Für $n \geq 2$ finden wir außerdem folgende untere Schranke.

$$\sqrt{2n(n+2)(n-1)} \geq \sqrt{2n \cdot n \cdot n/2} = \sqrt{n^3} = n^{3/2}.$$

Damit existieren Konstanten $c_1 = 1$, $c_2 = 2$ und $N = 2$, sodaß

$$c_1 \cdot n^{3/2} \leq \sqrt{2n(n+2)(n-1)} \leq c_2 \cdot n^{3/2}$$

für alle $n \geq N$.

Aufgabe T17

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?

Lösungsvorschlag:

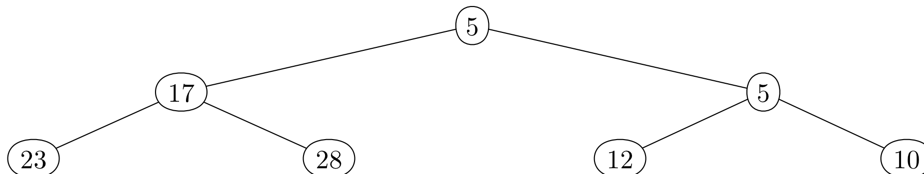
Das oberste Element des Stacks wird sofort am Anfang der Schleife entfernt, interessant ist also das erste der beiden eingefügten Intervalle. Bringen wir den Algorithmus dazu, als erstes Element ein sehr kleines Intervall (etwa der Größe eins) auf den Stack zu legen und dann gezwungenermaßen ein sehr großes Intervall als zweites, so benötigt Quicksort $\Omega(n)$ Iterationen der äußeren Schleife—eine Beispielinstanz dafür wäre schlicht ein bereits aufsteigend sortiertes Array. Da nun in jeder Iteration ein Element auf dem Stack verbleibt, erreicht der Stack eine Größe von $\Omega(n)$.

Dieses Verhalten kann verhindert werden, indem man immer das kürzere Intervall oben auf den Stack legt: so kann der Stack maximal eine Größe von $O(\log n)$ erreichen.

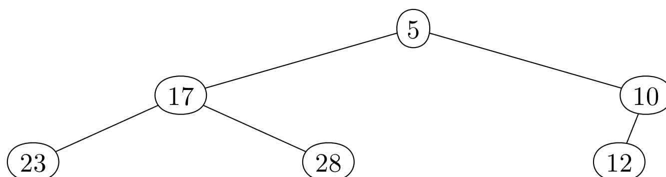
Nehmen wir o.b.d.A. an, die Eingabe bestehe aus einer beliebigen Permutation der Elemente $1 \dots n$ und alle diese Permutationen haben die gleiche Wahrscheinlichkeit. Ein einfaches Zählargument zeigt, daß die anfängliche Vertauschung Permutationen auf den Elementen $2 \dots n$ erzeugt, die wieder alle gleich wahrscheinlich sind: für jede der $(n-1)!$ vielen Permutationen können n verschiedene Permutation der Elemente $1 \dots n$ erzeugt werden, indem ein Element durch die 1 ersetzt und anschließend den anderen vorangestellt wird. Da dies niemals zwei gleiche Permutationen erzeugt, erhalten wir $n(n-1)! = n!$ viele Permutationen auf n Elementen— umgekehrt ist damit die Wahrscheinlichkeit, eine konkrete Permutation der Elemente $2 \dots n$ zu erhalten, $n\frac{1}{n!} = \frac{1}{(n-1)!}$

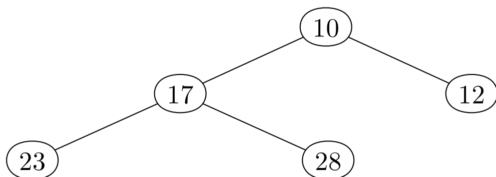
Aufgabe H15 (8 Punkte)
Fügen Sie die Zahlen 23, 12, 5, 17, 28, 10 und 5 in einen anfangs leeren Min-Heap ein (die kleineren Zahlen sind oben). Wie sieht dieser aus?
Entfernen Sie jetzt nacheinander dreimal die kleinste Zahl aus dem Heap. Wie sieht er nach jeder der drei Operationen aus?

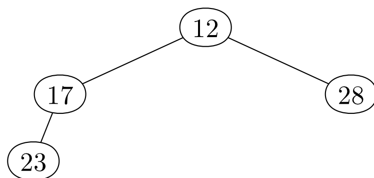
Lösungsvorschlag:
Der entstehende Heap sieht so aus:



Die weiteren Heaps nach jeweiligem Entfernen des kleinsten Elements sind:







Aufgabe H16 (8 Punkte)
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?

Lösungsvorschlag:

a) Gesucht ist die Anzahl der Index-Paare (i, j) mit $i < j$, so daß der Wert an Stelle i größer ist als der Wert an Stelle j. Für die gegebene Zahlenfolge sind das genau die Arrayindex Paare:

$$\begin{aligned} &\{(1,2),(1,3),(1,4),(1,5),(1,6),(1,8),(2,3),(2,5),(2,6),(2,8)\} \\ \cup\ &\{(3,6),(3,8),(4,5),(4,6),(4,8),(5,6),(5,8),(6,8),(7,8)\}, \end{aligned}$$

also hat sie 19 Inversionen.

b) Die Folge hat 6 Läufe:

- 8
- 6
- $3, 7$
- 4
- $2, 20$
- -45

c)
- 8 wird als Pivotelement gewählt
- Vertauschen von 20 und -45: $8, 6, 3, 7, 4, 2, -45, 20$
- Vertauschen von -45 und 20: $8, 6, 3, 7, 4, 2, 20, -45$
- Abbruch der Schleife: $l > r$ (l zeigt auf -45, r auf 20)
- Wert an Stelle l rückt nach vorne, Wert an Stelle r rückt an Stelle l und das Pivotelement rückt an Stelle r: $-45, 6, 3, 7, 4, 2, 8, 20$

d) In der letzten Mischphase sind die linke und die rechte Teilfolge bereits sortiert: $3, 6, 7, 8, -45, 2, 4, 20$. Gemischt werden sollen $3, 6, 7, 8$ und $-45, 2, 4, 20$.

- $3 > -45 \Rightarrow b[0] = -45$
- $3 > 2 \Rightarrow b[1] = 2$
- $3 \leq 4 \Rightarrow b[2] = 3$
- $6 > 4 \Rightarrow b[3] = 4$
- $6 \leq 20 \Rightarrow b[4] = 6$
- $7 \leq 20 \Rightarrow b[5] = 7$
- $8 \leq 20 \Rightarrow b[6] = 8$
- linker Counter hat linke Teilfolge verlassen $\Rightarrow b[7] = 20$