Datenstrukturen und Algorithmen, SS 2019
Prof. Dr. P. Rossmanith
S. Dollase, J. Dreier, H. Lotze

Datum: 16.5.2019

Übungsblatt mit Lösungen 06

Aufgabe T20

Beweisen Sie, dass es keinen vergleichsbasierten Sortieralgorithmus gibt, welcher ein beliebiges Array der Größe fünf mit nur sechs Vergleichen sortieren kann.

Lösungsvorschlag

In der Vorlesung wurde bewiesen, dass wir mindestens $\log(5!) = \log(120) \approx 6.9$ Vergleiche brauchen.

Aufgabe T21

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

a) Wie viele Inversionen hat diese?

b) Wie viele Läufe hat diese? Ein Lauf ist eine maximale sortierte Teilfolge.

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 ist das genau

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

Also hat die Zahlenfolge 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 $\Rightarrow b[7] = 20$

Aufgabe T22

Erfinden Sie ein Sortierverfahren, dass ein Array der Größe n in linearer Zeit sortieren kann unter der Annahme, dass das Array ausschließlich ganze Zahlen zwischen -1000 und 1000 enthält. Ist ihr Verfahren In-place?

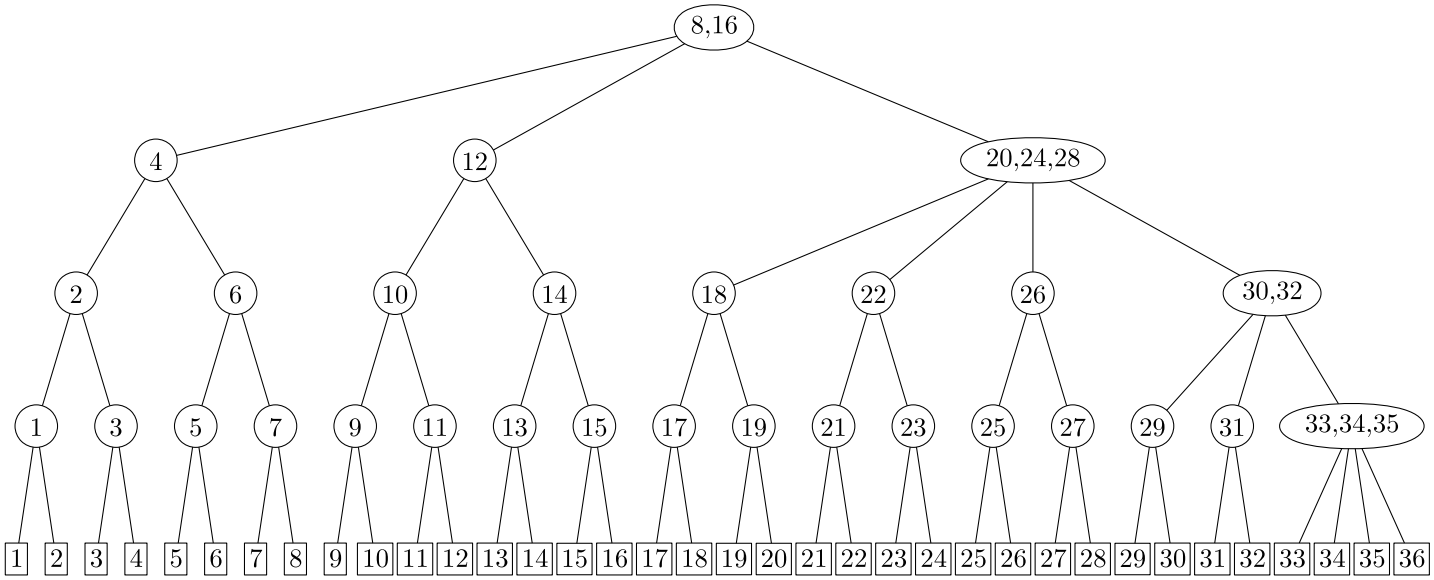
Erläutern Sie die Idee und stellen Sie den Algorithmus zusätzlich in Form von Pseudocode oder einer Implementierung in einer vernünftigen Programmiersprache vor.

Lösungsvorschlag

Wir zählen, wie häufig jede Zahl zwischen -1000 und 1000 vorkommt. Dafür initialisieren wir zuerst ein Array der Länge 2001 mit 0-en. Dann laufen wir einmal durch das Array und erhöhen in jedem Schritt den Eintrag an der entsprechenden Stelle. Dies geht in $O(n)$. Danach benutzen wir unser Zählarray um ein sortiertes Ausgabearray zu erzeugen. Dieses Verfahren is In-Place, da wir das Eingabearray überschreiben können und wir nur konstant viel extra Speicher brauchen (Array der Größe 2001 ist in $O(1)$). Eine Beispielimplementation in Java:

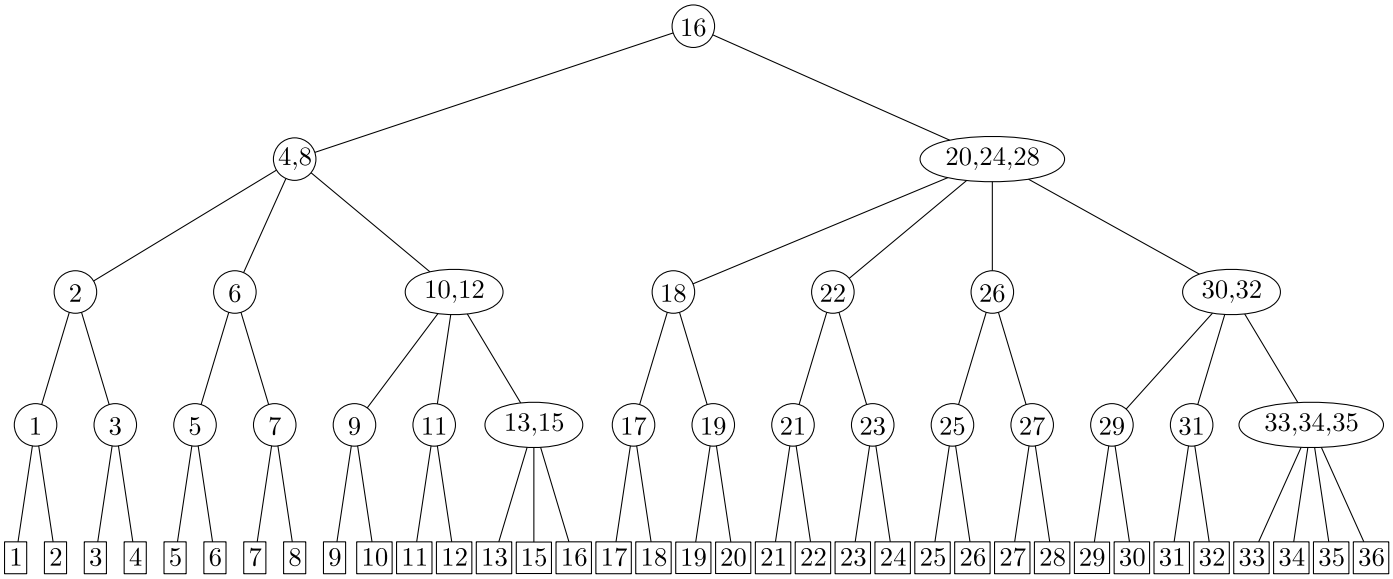
```
public void sort1000(int[] a) {
    int[] rates = new int[2001];
    for(int x : a) {
        rates[x + 1000]++;
    }
    int ai = 0;
    for(int ri = 0; ri < rates.length; ri++) {
        for(int rate = rates[ri]; rate > 0; rate--) {
            a[ai] = ri - 1000;
            ai++;
        }
    }
}
```

Aufgabe T23



Was passiert, wenn wir den Schlüssel 14 aus diesem $(2, 4)$-Baum löschen?

Lösungsvorschlag



Aufgabe H16 (10 Punkte)

Hier ist noch einmal eine einfache Variante des Quicksort-Algorithmus:

```
procedure quicksort(L, R) :
    if R ≤ L then return fi;
    p := a[L]; l := L; r := R + 1;
    do
        do l := l + 1 while a[l] < p;
        do r := r − 1 while p < a[r];
        vertausche a[l] und a[r];
    while l < r;
    temp := a[r]; a[L] := a[l]; a[l] := temp; a[r] := p;
    quicksort(L, r − 1); quicksort(r + 1, R)
```

Das Array a enthalte die Zahlen $4, 1, 3, 2, 5, 9$. Welche rekursiven Aufrufe gibt es? Geben Sie den Inhalt des Arrays jeweils am Anfang jedes Aufrufs und bevor die letzte Zeile ausgeführt wurde an.

Lösungsvorschlag

```
quicksort(0,5):
4 1 3 2 5 9
2 1 3 4 5 9
quicksort(0,2):
2 1 3 4 5 9
1 2 3 4 5 9
quicksort(0,0):
1 2 3 4 5 9
quicksort(2,2):
1 2 3 4 5 9
quicksort(4,5):
1 2 3 4 5 9
1 2 3 4 5 9
quicksort(4,3):
1 2 3 4 5 9
quicksort(5,5):
1 2 3 4 5 9
```

Aufgabe H17 (10 Punkte)

Gegeben sei ein Array der Länge n, welches garantiert nur k verschiedene Zahlen enthält – jede aber beliebig oft. Wir gehen davon aus, dass n viel größer als k ist und k sogar viel kleiner als $\log n$ sein kann.

Erfinden und beschreiben Sie ein Sortierverfahren, welches in dieser speziellen Situation sehr schnell ist.

Genauer gesagt: Die Laufzeit soll nur $O(n \log k)$ betragen.

Lösungsvorschlag

Wir können zum Beispiel einmal durch das Array gehen und alle Einträge in einen balancierten Binärbaum (z.B. AVL-Baum oder Splay-Baum) eintragen. Dies benötigt bereits $O(n \log k)$ Zeit, da wir n Operationen auf einen Baum der Größe $O(k)$ ausführen. Dabei können wir aber auch gleichzeitig zählen, wie oft jeder Eintrag im Array vorkommt.

Sagen wir, die Einträge seien z_i mit $1 \leq i \leq k$ und ihre Anzahlen n_i. Jetzt überschreiben wir einfach die ersten n_1 Positionen des Arrays mit z_1, die nächsten n_2 Positionen mit z_2, usw.

Hier ist eine Prozedur in Java notiert, welche ein Array a der Größe n auf diese Weise sortiert:

```
public void sortNlogK(int[] a) {
    Tree rates = new Tree();
    for(int x : a) {
        rates.incrementOrInsertOne(x);
    }
    int ai = 0;
    for(int key : rates.keys()) {
        for(int rate = rates.get(key); rate > 0; rate--) {
            a[ai] = key;
            ai++;
        }
    }
}
```

Alternativ kann man auf gleiche Weise auch Heaps oder Skip-Lists verwenden. Es wäre auch denkbar erst die Arrayeinträge auf $O(t)$ bits zu hashen und dann Radix-Sort oder Counting-Sort durchzuführen.

Aufgabe H18 (5+5 Punkte)
Gegeben sei das Array $[3, 9, 6, 4, 8, 10, 5, 2]$.

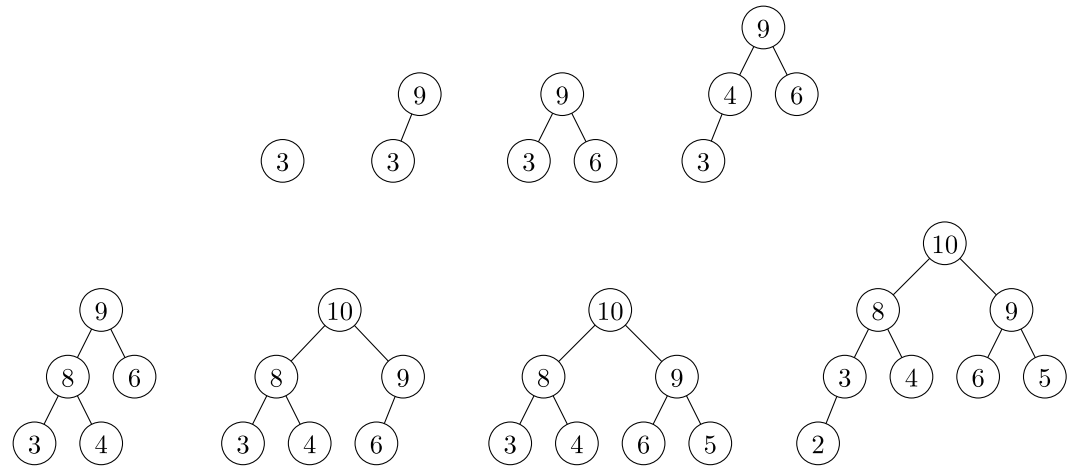
a) Führen Sie den Heapsort-Algorithmus aus der Vorlesung darauf aus.

b) Führen Sie den Mergesort-Algorithmus aus der Vorlesung darauf aus.

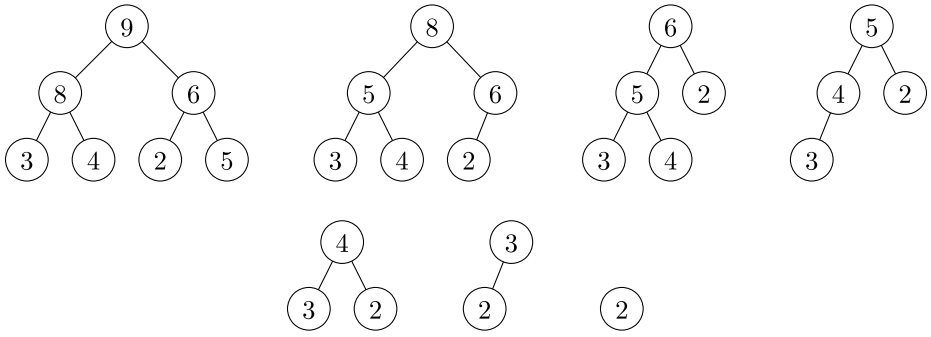
Geben Sie alle Zwischenschritte an.

Lösungsvorschlag

a) Als erstes fügen wir die Elemente der Reihe nach in den Heap ein.



Jetzt sind alle Elemente eingefügt worden. Wiederholtes Löschen der Wurzel liefert die Elemente in absteigender Reihenfolge.



b) Als erstes teilen wir das Array rekursiv in zwei Hälften, bis wir Arrays der Länge eins haben. Anschließend werden die beiden Hälften, die bereits sortiert sind, zusammengefügt um ein neues sortiertes Array zu erhalten.

Die Teilung und Mischung verläuft wie folgt, sortierte Unterarrays, die entstehen wenn die Rekursion terminiert, sind unterstrichen:

- $[3, 9, 6, 4, 8, 10, 5, 2]$
- split: $[3, 9, 6, 4]$ $[8, 10, 5, 2]$
- split: $[3, 9]$ $[6, 4]$ $[8, 10, 5, 2]$

- split: $\underline{[3]}$ $\underline{[9]}$ $[6, 4]$ $[8, 10, 5, 2]$
- merge: $\underline{[3, 9]}$ $[6, 4]$ $[8, 10, 5, 2]$
- split: $\underline{[3, 9]}$ $\underline{[6]}$ $\underline{[4]}$ $[8, 10, 5, 2]$
- merge: $\underline{[3, 9]}$ $\underline{[4, 6]}$ $[8, 10, 5, 2]$
- merge: $\underline{[3, 4, 6, 9]}$ $[8, 10, 5, 2]$
- split: $\underline{[3, 4, 6, 9]}$ $[8, 10]$ $[5, 2]$
- split: $\underline{[3, 4, 6, 9]}$ $\underline{[8]}$ $\underline{[10]}$ $[5, 2]$
- merge: $\underline{[3, 4, 6, 9]}$ $\underline{[8, 10]}$ $[5, 2]$
- split: $\underline{[3, 4, 6, 9]}$ $\underline{[8, 10]}$ $\underline{[5]}$ $\underline{[2]}$
- merge: $\underline{[3, 4, 6, 9]}$ $\underline{[8, 10]}$ $\underline{[2, 5]}$
- merge: $\underline{[3, 4, 6, 9]}$ $\underline{[2, 5, 8, 10]}$
- merge: $\underline{[2, 3, 4, 5, 6, 8, 9, 10]}$