

Übungsserie 3

Datenstrukturen & Algorithmen

Universität Bern
Frühling 2018

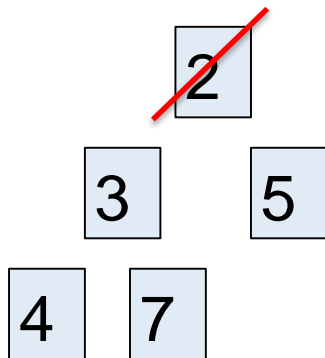
Übungsserie 3

- > Heapsort, Quicksort
 - > 6 Aufgaben auf Papier
 - > 3 praktische Aufgaben zu Quicksort

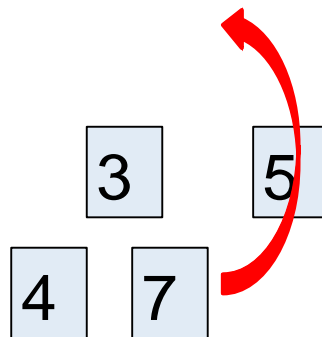
 - > Poolstunde: Montag 17:00 – 18:00
-

Aufgabe 1

- > EXTRACT-MIN durchspielen
- > EXTRACT-MIN entfernt das Element mit dem kleinsten Schlüssel und gibt dieses zurück
— Buch Kapitel 6.5
- > **Oft vergessen** Wurzelement durch hinterstes Element ersetzen: $A[1] = A[A.\text{heap-grösse}]$



[2,3,5,4,7]



[7,3,5,4]

Danach MIN-HEAPIFY....

[...]

Aufgabe 2

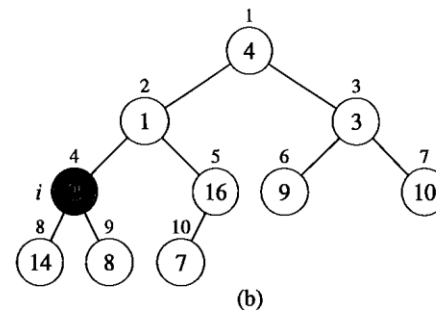
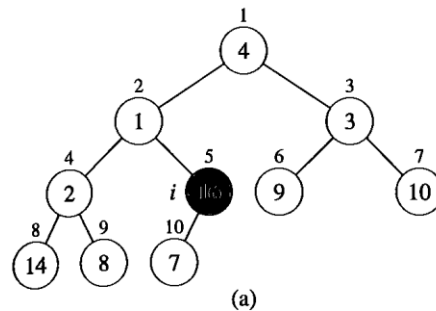
- > **Zeige** In jedem Teilbaum eines Max-Heaps speichert die Wurzel den grössten Wert des Teilbaumes
- > Max-Heap Eigenschaft: $A[\text{parent}(i)] \geq A[i]$
- > Induktionsbeweis.
 - **Verankerung** Überprüfe die Aussage für einen Baum mit nur einem Blatt
 - **Induktionsschritt** Gegeben ein Knoten T mit linkem und rechten Unterbaum X, Y für welche die Aussage stimmt....

Aufgabe 4

- > Zeige den Ablauf der `Build-Max-Heap` Funktion auf einem vorgegebenen Array
 - Buch Abbildung 6.3
 - Schaut euch die Abbildung an!
 - Zeichne den Baum zu **jedem** Zwischenschritt

A

4	1	3	2	16	9	10	14	8	7
---	---	---	---	----	---	----	----	---	---



USW...

Aufgabe 5

- > Laufzeit von `HEAPSORT`
 - ... wenn Eingabefeld `A` in aufsteigende Reihenfolge sortiert ist
 - ... wenn Eingabefeld `A` in absteigender Reihenfolge sortiert ist

 - > Hilfsfragen
 - Welche Algorithmen kommen in `HEAPSORT` vor?
 - Was ist die Laufzeit dieser Algorithmen?
 - Wie oft werden diese ausgeführt?

 - > Kapitel 6.4 im Buch
-

Aufgabe 6

- > QUICKSORT durchspielen
 - Alle Aufrufe der QUICKSORT-Funktion mit ihren **Parametern** chronologisch auflisten
 - Siehe auch Abbildung 7.1 im Buch

QUICKSORT(A, p, r)

```
1  if  $p < r$ 
2       $q = \text{PARTITION}(A, p, r)$ 
3      QUICKSORT( $A, p, q - 1$ )
4      QUICKSORT( $A, q + 1, r$ )
```

PARTITION(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          vertausche  $A[i]$  mit  $A[j]$ 
7  vertausche  $A[i + 1]$  mit  $A[r]$ 
8  return  $i + 1$ 
```

Aufgabe 6

> Teilen

- Eingabefeld $A[p..r]$ so in zwei Teilfelder $A[p..q-1]$, $A[q+1..r]$ zerlegen, dass jedes Element von $A[p..q-1] \leq A[q]$ ist, welches wiederum \leq jedem Element von $A[q+1..r]$ ist.
Berechne diesen Index q

> Beherrschen

- Beide Teilfelder durch rekursiven Aufruf von QUICKSORT sortieren

> Vereinigen


- Da die Teilfelder bereits sortiert sind, ist das gesamte Feld $A[p..r]$ nun sortiert

Quicksort Beispiel

> Bsp: Input [35,96,57,84] p r

Quicksort(A,1,5)

Partition



35	96	57	40	84
35	96	57	40	84
35	96	57	40	84
35	57	96	40	84
35	57	40	96	84
35	57	40	84	96

Quicksort Beispiel

> Bsp: Input [35,96,57,84]

Quicksort(A,1,5)

Partition

Quicksort(A,1,3)

Quicksort(A,5,5)

35	96	57	40	84
35	96	57	40	84
35	96	57	40	84
35	57	96	40	84
35	57	40	96	84
35	57	40	84	96

Quicksort Beispiel

> Bsp: Input [35,96,57,84]

Quicksort(A,1,5)

Partition

Quicksort(A,1,3)

Partition

Quicksort(A,5,5)

35	96	57	40	84
35	96	57	40	84
35	96	57	40	84
35	57	96	40	84
35	57	40	96	84
35	57	40	84	96
35	57	40		
35	57	40		
35	40	57		

Quicksort Beispiel

> Bsp: Input [35,96,57,84]

Quicksort(A,1,5)

Partition

Quicksort(A,1,3)

Partition

Quicksort(A,1,1)

Quicksort(A,3,3)

Quicksort(A,5,5)

35	96	57	40	84
35	96	57	40	84
35	96	57	40	84
35	57	96	40	84
35	57	40	96	84
35	57	40	84	96
35	57	40		
35	57	40		
35	40	57		

Praktische Aufgabe

- > **Gegeben** Code, welcher Studenten nach **Matrikelnummer** sortiert
 - Nicht-randomisierter Quicksort Algorithmus

- > **Teilaufgaben**
 - Neue Klasse schreiben, sodass nach **Name/Vorname** sortiert werden kann
 - Comparator-Konzept in Java
 - **Nützlich** `String.compareTo(...)`
 - `Quicksort(Quicksort(A))` → *stack overflow error*
 - Wieso??
 - Randomized Quicksort to the rescue
 - Algorithmus modifizieren, *stack overflow error* verhindern

Praktische Aufgabe

- > Code Übersicht
 - **StudentIn.java** Klasse für Objekte, die Daten einer StudentIn enthalten. Dieser Datentyp soll sortiert werden.
 - **Quicksort.java** Quicksort Implementation
 - **MatrikelNrComparator.java** Beispiel eines Comparators
 - **MiniTestApp.java** Testprogramm zum Testen eurer Lösung
 - **SortTestApp.java** Testet Quicksort mit grossen zufälligen Eingaben
 - **DataGenerator.java** Erstellt Zufallsdaten
 - **Timer.java** Zeitmessung
-

Quicksort

```

public static <T> void quickSort(ArrayList<T> array, int left, int right, Comparator<T> comp) {
    if (right > left) { // Abbruchbedingung der Rekursion
        T temp;        // temporäre Hilfsvariable zum swappen

        // *** 1. Pivotelement selektieren:
        T pivot = array.get(right);

        // *** 2. Aufteilung in Subsequenzen durchführen:
        int l = left-1;
        int r = right;
        do {
            do l++; while (comp.compare(array.get(l), pivot) < 0);
            do r--; while (r > l && comp.compare(array.get(r), pivot) > 0);

            // swap(array, l, r):
            temp = array.get(l);
            array.set(l, array.get(r));
            array.set(r, temp);
        } while (r > l);

        array.set(r, array.get(l)); // Korrektur: einmal zuviel getauscht

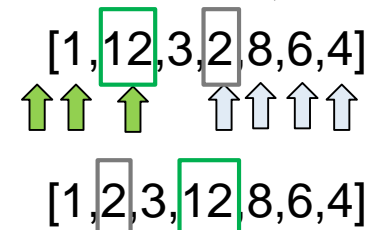
        // Pivotelement in sortierte Position bringen:
        array.set(l, pivot);
        array.set(right, temp);

        // *** 3. Rekursiv die beiden Subarrays sortieren:
        quickSort(array, left, l-1, comp);
        quickSort(array, l+1, right, comp);
    }
}
    
```

Auswahl des Pivot Elements

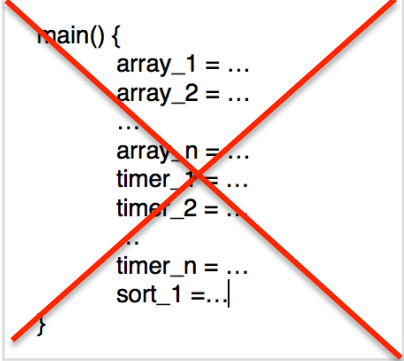
Partitionieren

Leicht anders als in Vorlesung.
Sucht von vorne und hinten Paare von
'falschen' Elementen, vertauscht diese



Beherrschen

Allgemein

- > Entweder Übung von Hand lösen...
 - ... und **leserlich** schreiben!
- > Oder Lösung am Computer verfassen und ausdrucken
- > Programmieraufgaben
 - > Objektorientiert arbeiten, keine Scripts schreiben
 - >  → `public int[] prepareArray() {...}`
 - > Code kommentieren

Fragen?

u^b

b
UNIVERSITÄT
BERN

