

# 1. Übungsblatt zu Algorithmen II im WS 2019/2020

[http://algo2.iti.kit.edu/AlgorithmenII\\_WS19.php](http://algo2.iti.kit.edu/AlgorithmenII_WS19.php)  
{sanders, heuer, lamm}@kit.edu

## Aufgabe 1 (Analyse: Kleinaufgaben)

- Geben Sie die wesentlichen Unterschiede –laut Vorlesung– zwischen einer (normalen) *Priority Queue* und einer adressierbaren *Priority Queue* an.
- Vergleichen Sie die Laufzeit einer *merge*-Operation für *Pairing Heaps* und Array basierte *Binary Heaps* (also wie aus Algorithmen I bekannt).

## Aufgabe 2 (Analyse: Laufzeitverhalten)

- Beweisen Sie allgemein für adressierbare *Priority Queues* die untere Laufzeitschranke von  $\Omega(\log n)$  für *deleteMin* unter der Voraussetzung, dass *insert* konstante Laufzeit benötigt.
- Warum muss diese untere Laufzeitschranke nicht gelten, wenn *insert* mehr Zeit benötigen darf?

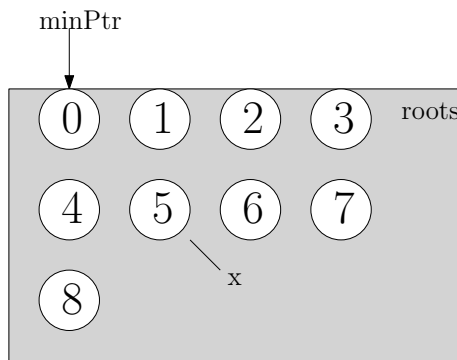
## Aufgabe 3 (Analyse: best-case Verhalten)

- Geben Sie einen Zustand eines *Fibonacci Heaps* an, für den die nächsten  $n$  *deleteMin*-Operationen jeweils konstante Laufzeit benötigen (nicht amortisiert). Begründen Sie Ihre Antwort. Gehen Sie davon aus, dass zwischen den *deleteMin*-Operationen keine anderen Operationen ausgeführt werden.
- Geben Sie einen Algorithmus an, welcher den von Ihnen angegebene Zustand für beliebige  $n$  erzeugt. Beweisen Sie die Korrektheit des Algorithmus.

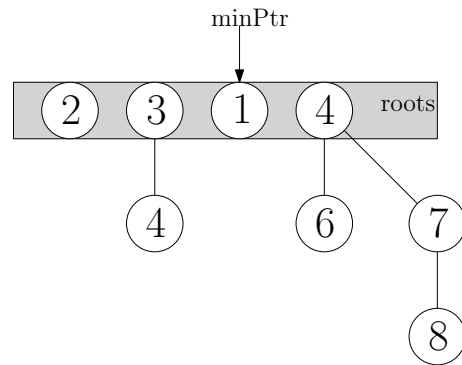
**Aufgabe 4** (Rechnen: Fibonacci Heaps)

Gegeben sei ein *Fibonacci Heap* mit unten eingezeichnetem Zustand (a).

- Geben Sie eine möglichst kurze Folge von Operationen an, die diesen Zustand erzeugt.
- Führen Sie anschließend die Operationen `deleteMin()` auf dem Heap aus. Zeichnen Sie den Zustand des Heaps nach jedem Einfügen eines Baums in ein leeres Bucket und nach jeder Union-Operation.
- Geben Sie eine möglichst kurze Folge von Operationen an, die den unten eingezeichneten Zustand (b) erzeugt. Tipp: der eingezeichnete Zustand lässt sich aus dem Heap in Abbildung (a) nach der `deleteMin`-Operation durch weitere Operationen erzeugen.



(a)



(b)

**Aufgabe 5** (Entwurf: Datenstrukturen)

- Erweitern Sie die Datenstruktur *Pairing Heap* um die Operation `increaseKey(h: Handle, k: Key)`. Ihre Operation sollte amortisiert  $O(\log n)$  Laufzeit benötigen. Geben Sie Pseudocode an. Wie würden Sie bei einem *Binary Heap* vorgehen?
- Entwerfen Sie eine Datenstruktur welche die Operationen `insert` in  $O(\log n)$ , Median bestimmen in  $O(1)$  und Median entfernen in  $O(\log n)$  unterstützt. Eine Beschreibung in Worten ist ausreichend.

**Ausgabe:** 22.10.2019

**Abgabe:** keine Abgabe, keine Korrektur