Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dr. Hanjo Täubig
Tobias Lieber

Sommersemester 2011
Übungsblatt 7
4. Juli 2011

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: In der jeweiligen Tutorübung

Hausaufgabe 1

Implementieren Sie in der Klasse `UIaHeap` einen adressierbaren binären Heap, der neben den Basisoperationen auch die Operationen `inserth`, `remove` und `decreaseKey` implementiert.
Verwenden Sie für Ihre Implementierung die auf der Übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die Klasse `UIaHeap`.
Achten Sie bei der Abgabe Ihrer Aufgabe darauf, dass Ihre Klasse `UIaHeap` heißt und auf den Rechnern der Rayhalle (`rayhalle.informatik.tu-muenchen.de`) mit der bereitgestellten Datei `main_ah` kompiliert werden kann. Anderenfalls kann eine Korrektur nicht garantiert werden. Achten Sie darauf, dass Ihr Quelltext ausreichend kommentiert ist.
Schicken Sie die Lösung per Email mit dem Betreff `[GAD] Gruppe <Gruppennummer>` an Ihren Tutor.
Hinweis: Hier macht die Benutzung eines Dynamischen Arrays Sinn. Falls Sie die früher benutzte Klasse `UIsArray` *benutzen wollen, so fügen Sie den Quellcode Ihrer Abgabe hinzu. Sie können aber auch das in Java implemntierte Dynamische Array benutzen, welches deutlich flexibler ist.*

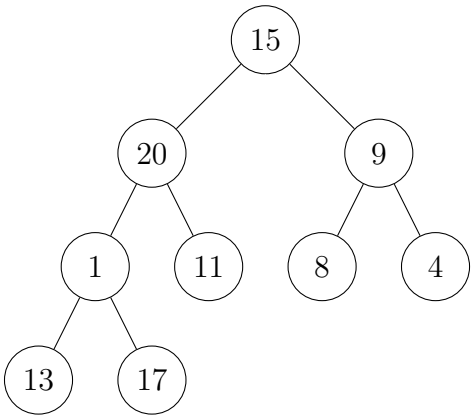
Aufgabe 1

Führen Sie auf einem anfangs leeren Binären Heap folgende Operationen aus und stellen Sie die Zwischenergebnisse graphisch dar:

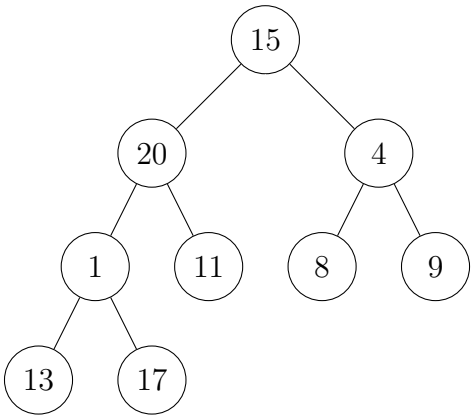
- build({15, 20, 9, 1, 11, 8, 4, 13, 17}),
- insert(7),
- delMin(),
- decreaseKey(20, 3),
- delete(8).

Lösungsvorschlag

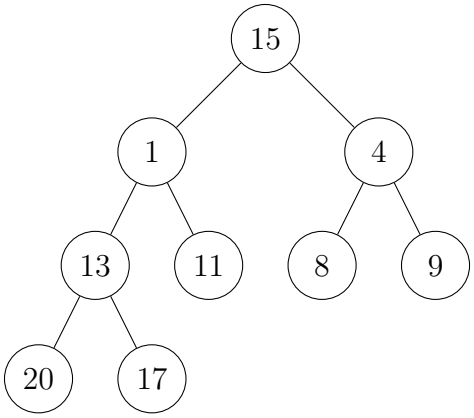
Anfang von Build:



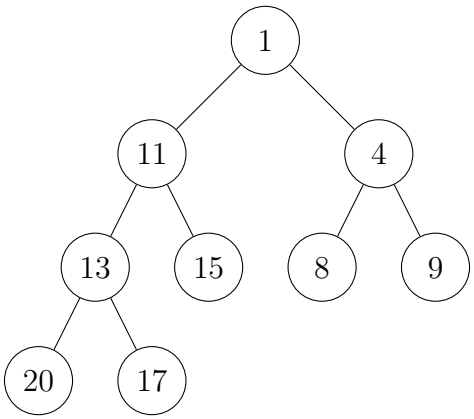
Bei siftDown von 1 passiert nichts. Daher gleich die 9:



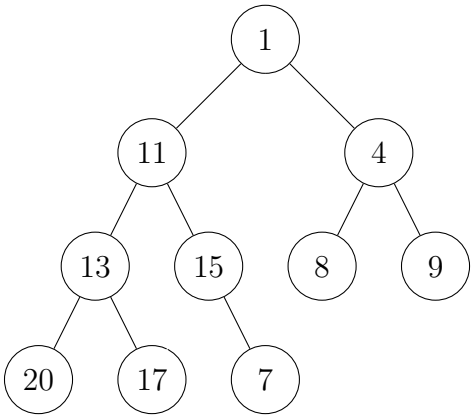
SiftDown auf der 20:



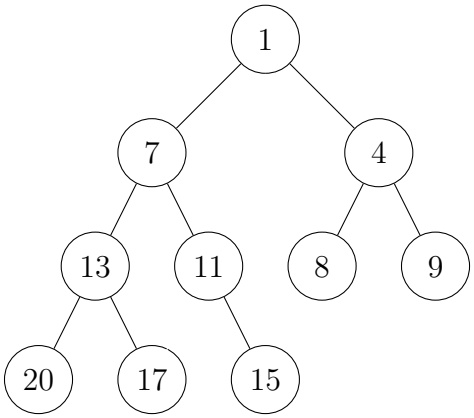
SiftDown auf der 15 und damit Endresultat für Build:



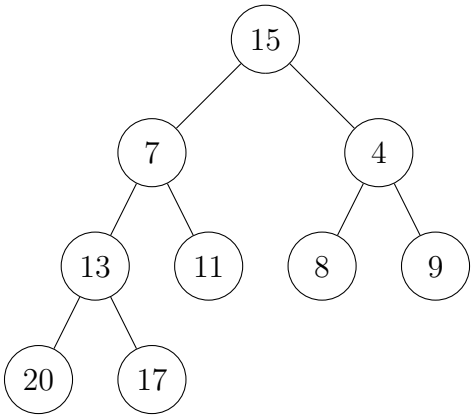
Einfügen der 7. Zuerst hinten Anhängen.



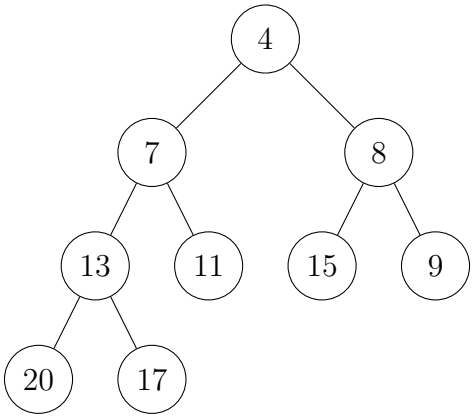
SiftUp auf die 7:



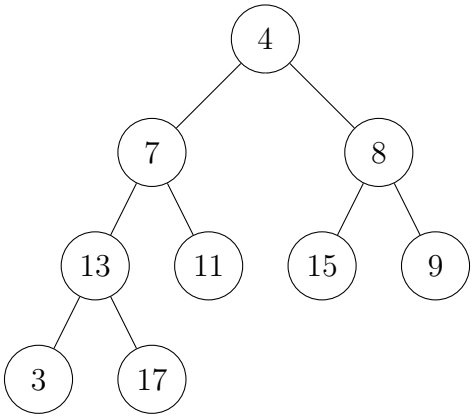
Vorbereitung für deleteMin:



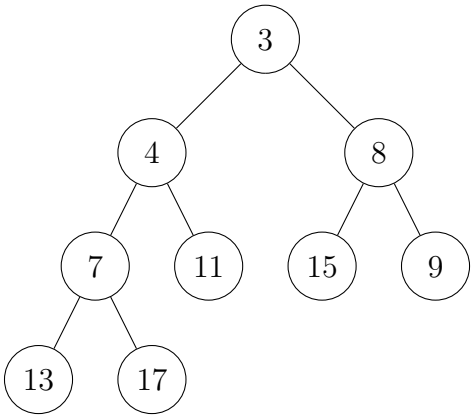
SiftDown auf die 15:



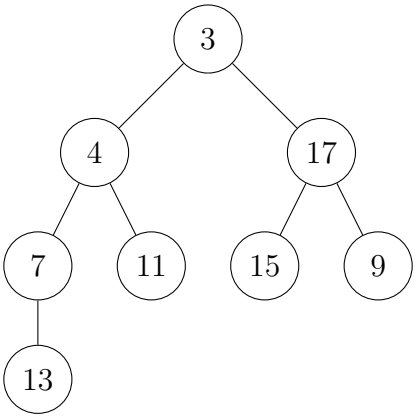
Wertänderung der 20 auf 3:



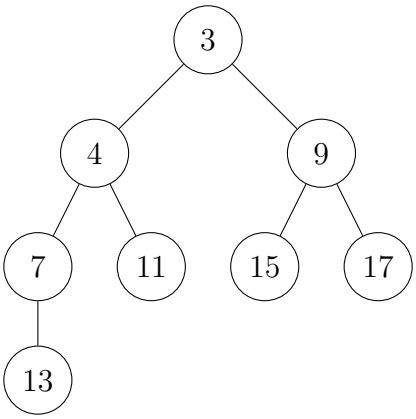
SiftUp der 3:



Delete von 8 vorbereiten:



SiftDown der 17.



Aufgabe 2

Analysieren Sie die Anzahl der Element-Vergleiche, die der in der Vorlesung angegebene `siftDown`-Algorithmus im Worst-Case benötigt.
Geben Sie ausgehend von dem Algorithmus in der Vorlesung einen Algorithmus an, der mit $\log n + \mathcal{O}(\log \log n)$ Vergleichen auskommt.

Lösungsvorschlag

Der in der Vorlesung angegebene Algorithmus führt in jedem Durchlauf der while-Schleife gerade 2 Elementvergleiche aus – Vergleich der Kinder und Test ob das kleinere Kind kleiner als das Element ist. Somit werden gerade $2 \log n$ Vergleiche benötigt.
Zur Reduzierung der Vergleiche benutzen wir zusätzlichen Speicherplatz um uns den Pfad der kleinsten Kinderelemente zu zpeichern. Dieser Pfad hat höchstens Länge $\lceil \log n \rceil$. Zum Finden der korrekten Position führen wir nun eine binäre Suche auf dem Pfad aus. Diese benötigt $\mathcal{O}(\log \log n)$ Vergleiche.
Somit erhalten wir als Gesamtanzahl an Vergleichen: $\log n + \mathcal{O}(\log \log n)$.