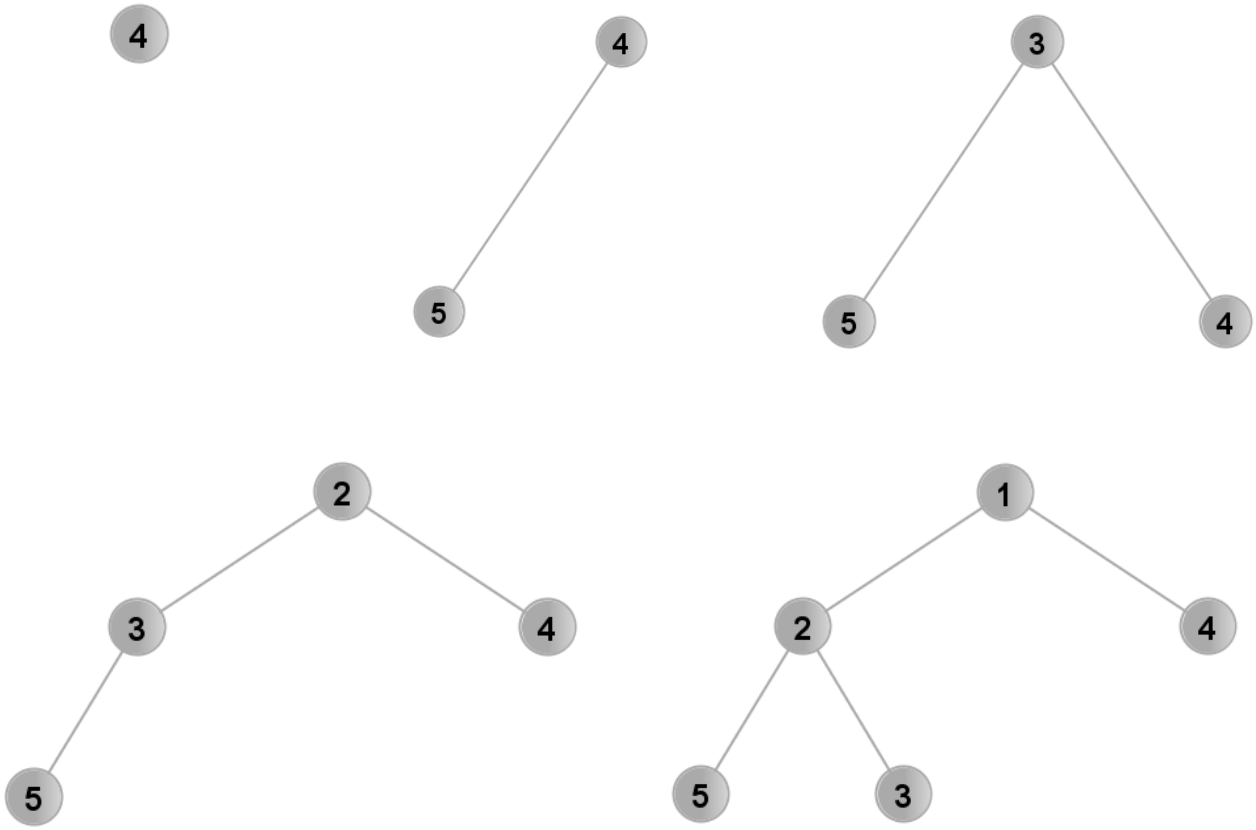
Übungsserie 4: Lösungen

Aufgabe 1: Heap-Operationen

a) Heap-Aufbau mit *insert()*

Ein Heap wird mit folgender Input-Sequenz aufgebaut: 4, 5, 3, 2, 1
Es soll der Heap nach jeder *insert()*-Operation aufgezeichnet werden.

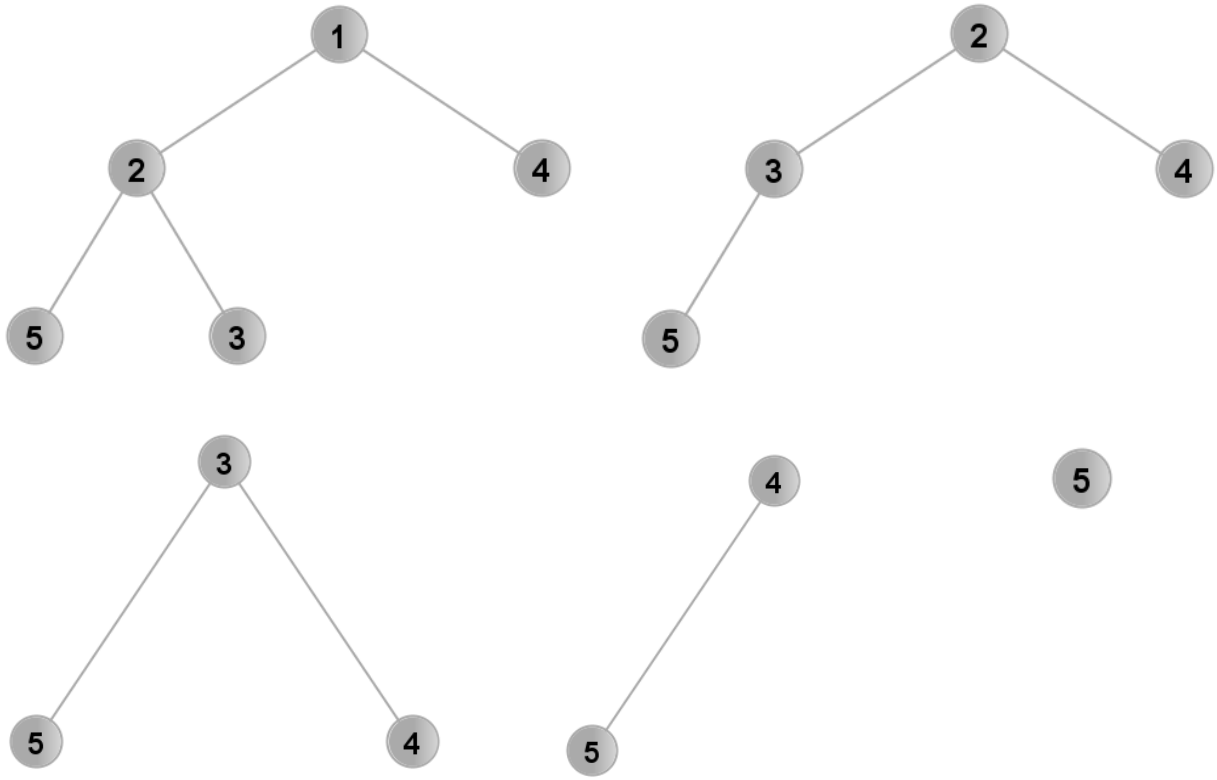
Lösung:



b) *removeMin()*

Nun wird sequentiell solange *removeMin()* aufgerufen bis der Heap leer ist.
Es soll wiederum der Heap nach jeder *removeMin()*-Operation aufgezeichnet werden.

Lösung:



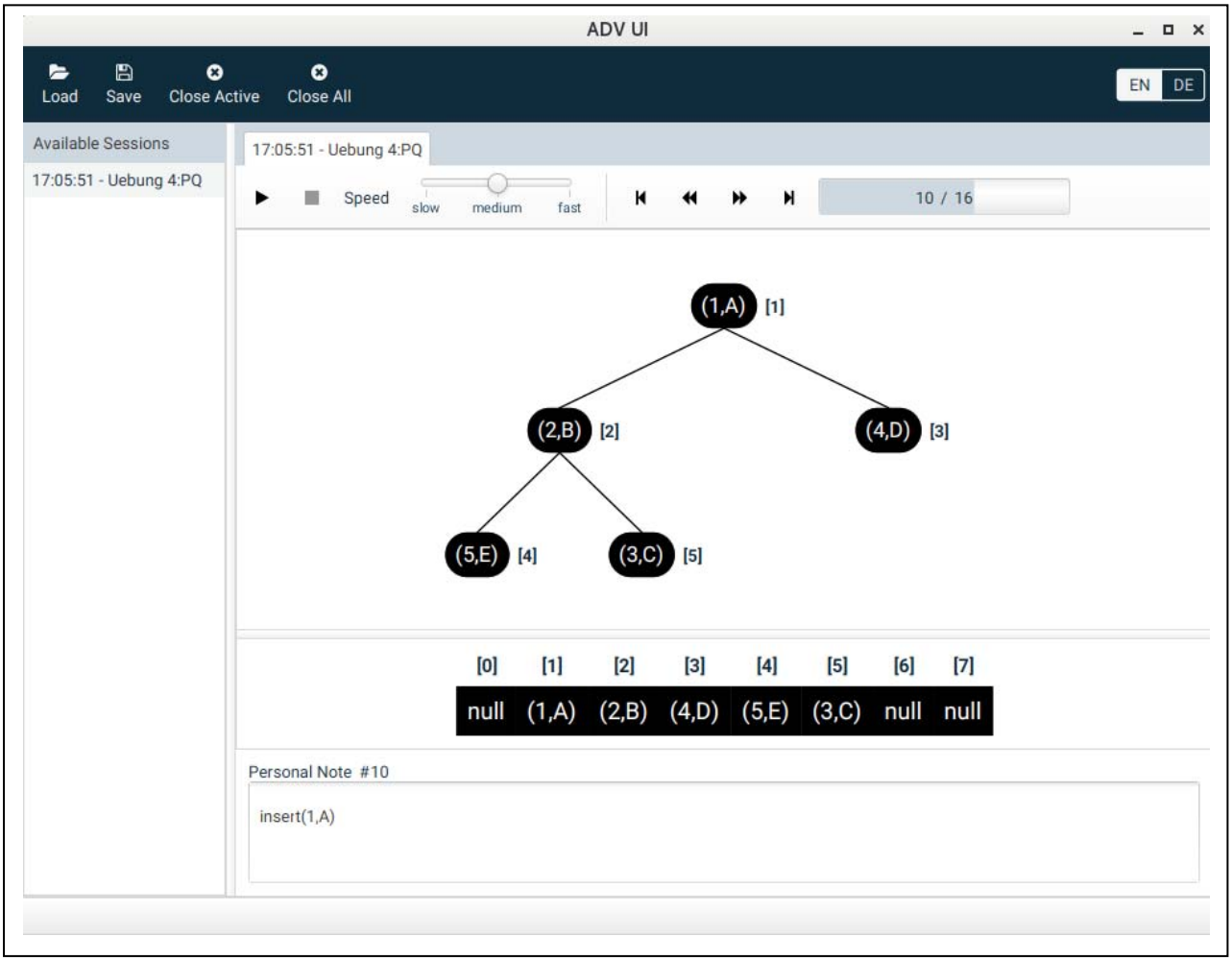
Aufgabe 2: Implementierung einer Heap-basierten Priority-Queue

Es soll eine *Heap-basierte Priority-Queue* implementiert werden.
Der Heap selbst soll dabei *Array-basiert* sein.
Die entsprechende Ausgangslage liegt auf dem ILIAS-Server.

Hinweise:

- Die Priority-Queue hat eine fixe Länge (vergrössert sich nicht automatisch).
- Beim *downheap()* mit zwei gleichen Child-Keys soll der Swap mit dem linken Child erfolgen.
- Die *Test-Applikation* mit *Session-Log* befindet sich in *PriorityQueueTest.java* resp. *priority_queue_test.py.*

Optional kann zur Visualisierung der "Algorithm & Data Structure Visualizer (ADV)" benutzt werden:



Dazu von *ILIAS:Administration>Setup Entwicklungsumgebung>Algorithm & Data Structure Visualizer (ADV)* das für die Plattform entsprechende *jar*-File downloaden.
Dann den *ADV-Visualisierungs-Server* in einem Terminal starten, z.B. für Windows:
`java -jar adv-ui_windows_x64_v2.3.0.jar`
Eine allfälliger Hinweis der Firewall kann bestätigt werden (es wird für die Kommunikation mit unserem Test-Programm *PriorityQueueTest* ein Socket geöffnet).

Der *ADV* kann u.a. auch früher gespeicherte *Session's* wieder abspielen.
Die Datei *uebung04/Uebung_04_PQ.adv* ist eine solche Session und kann mit *Load* geladen und dann abgespielt werden.
Konkret ist es genau das Szenario aus *PriorityQueueTest* (und auch von Aufgabe 1).
Somit ist es schlussendlich das Ziel von dieser Aufgabe, dass unser Programm die selbe Session erzeugt.

Dazu wird in der Klasse *PriorityQueueTest* in *main()* dann *ADV* aktiviert, indem anstelle von *PriorityQueue* neu *PriorityQueueADV* instanziert wird:

PriorityQueueTest.java:

```
...
// new PriorityQueue<>(7);
new PriorityQueueADV<>(7, "Uebung 4:PQ", 2, 2);
...
```

priority_queue_test.py:

```
...
# pq = PriorityQueue(7)
pq = PriorityQueueADV(7, "Uebung 4:PQ", 2, 2)
...
```

Von nun an werden die Informationen über unseren Baum in *PriorityQueue.java* resp. *priority_queue.py* jeweils automatisch zum *ADV* gesendet und dort dargestellt.

Bei *Java* sind dafür noch zusätzliche *jar*-Dateien vom *ADV* einzubinden (Hinweis: Im Eclipse-Projekt vom 'Setup Entwicklungsumgebung' von ILIAS ist dies bereits gemacht).
Dazu ist in der Entwicklungsumgebung der *Build-Path* um die zwei Dateien aus dem *lib*-Verzeichnis in *ILIAS:Administration>Setup Entwicklungsumgebung>Algorithm & Data Structure Visualizer (ADV)* zu erweitern: *adv-lib-2.2.jar* und *adv-util-2.0.jar*.
Z.B. in Eclipse: *Project > Properties > Java Build Path > Libraries > Classpath: Add External JARs...*
Hinweis: Sicherstellen, dass der *Classpath* erweitert wird, nicht der *Modulepath*.

Im Weiteren ist sicherzustellen, dass die Java-Version >= 15 ist.

Lösung: Siehe „PriorityQueue.java" resp. „priority_queue.py"