Aufgabe 1

Bei der Analyse grosser Datenmengen muss man bisweilen aus sehr vielen Elementen (z.B. mehrere Millionen) wenige (z.B. 100) bedeutsame heraussuchen. Nehmen wir an, ein Iterator liefert nacheinander N Elemente, von denen die M mit den grössten Schlüsseln gesucht sind.

Ansatz 1: Die N Elemente werden in einer Liste gespeichert, sortiert, und dann werden die M grössten aus dieser Folge gelesen.

Aufwand: O(N log N)

- Speichern und Sortieren dauert O(N log N)
- Auslesen dauert O(M)

Aufgabe 1

Bei der Analyse grosser Datenmengen muss man bisweilen aus sehr vielen Elementen (z.B. mehrere Millionen) wenige (z.B. 100) bedeutsame heraussuchen. Nehmen wir an, ein Iterator liefert nacheinander N Elemente, von denen die M mit den grössten Schlüsseln gesucht sind.

Ansatz 2: Während Element für Element aus dem Iterator gelesen wird, werden in einer Priority Queue die M grössten jeweils bereits gelesenen Elemente gesammelt. Für jedes neu gelesene Element wird geprüft, ob es grösser ist, als das kleinste in der Priority Queue. Nur dann muss es berücksichtigt werden.

Aufwand: O(N) average, O(N log M) worst case

- Jedes Element Speichern und Vergleichen: O(N)
- Ein Element Einfügen: O(log M)

Aufgabe 1

```
<K extends Comparable<? super K>> K[] maxM(Iterator<K> it, int M) {
  /* Priority Queue erstellen */
  MinPriorityQueue<K> q = new ...;
  /* Priority Queue füllen */
  while (q.size() < M && it.hasNext()) q.add(it.next());
  /* Halte nur die M grössten Elemente in der Priority Queue */
  while (it.hasNext()) {
     K elem = it.next();
     if (elem > q.min()) {
        q.removeMin();
        q.add(elem);
     }
  }
  /* Gib die M grössten Elemente aus der Priority Queue aus */
  K[] arr = (K[])new Comparable[M];
  for (int i = 0; i < M; i++) arr[i] = q.removeMin();
  return arr;
}
```

Aufgabe 2 (Median)

Sie möchten aus N Elementen jeweils den Median kennen. Entwickeln Sie eine Datenstruktur mit den zwei Methoden add(elem) und middle().

Idee: Min-Heap (Big) mit den ⌈n/2⌉ grösseren, Max-Heap (Small) mit den ⌊n/2⌋ kleineren Elementen

middle():

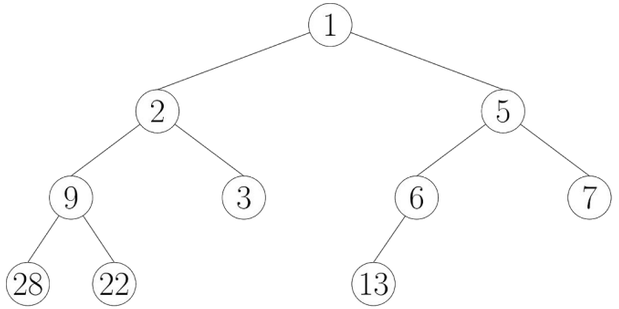
```
return Big.min()
```

add(elem):

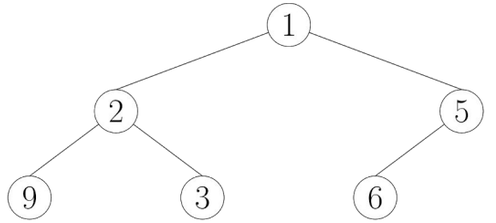
```
if( elem < middle() ) {
   Small.add(elem);
   if ( Small.size() > Big.size() ) { Big.add(Small.removeMax()); }
else {
   Big.add(elem);
   if ( Big.size()-1 > Small.size() ) { Small.add(Big.removeMin()); }
}
```

Fachhochschule Nordwestschweiz

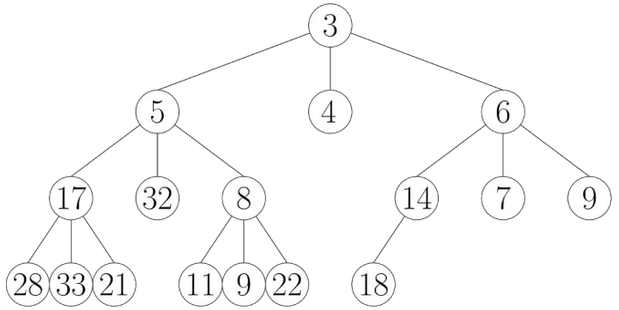
Aufgabe 3 (Heap or not)



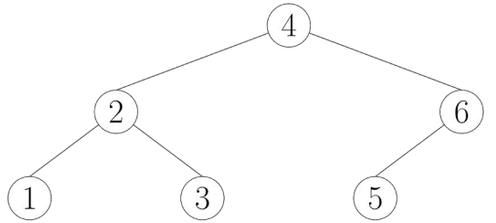
Kein Heap, 3 hat keine Blätter



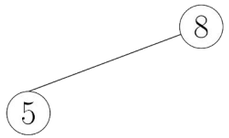
korrekter Min-Heap



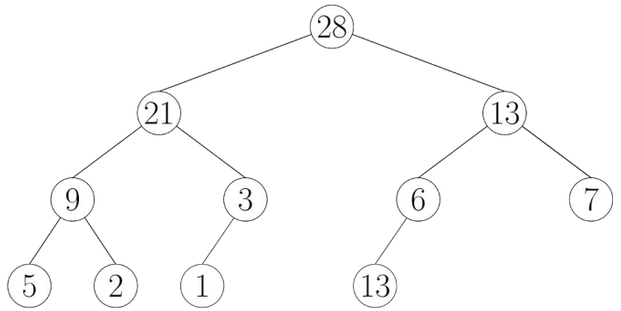
Kein Heap, weil kein Binärbaum



AVL-Baum, aber kein Heap



korrekter Max-Heap



kein Heap. Blatt 13 passt nicht.

Aufgabe 4 (Eigenschaften Heap)

1) Ist ein Teilbaum eines Heaps ein Heap?

   Ja

2) Wo liegt das kleinste Element in einem Min-Heap?

   Immer in der Wurzel, also an Index 0

3) Wo liegt das grösste Element in einem Min-Heap?

   Immer in einem Blatt, also in der zweiten Hälfte des Array

4) Wie hoch ist ein Heap aus n Elementen?

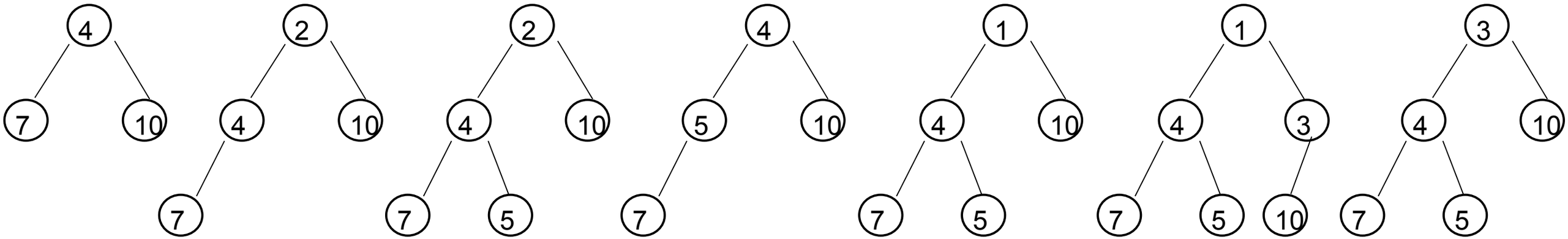
   Der Baum hat Höhe log n (aufgerundet)

5) Wie viele Blätter hat ein Heap aus n Elementen?

   n/2 aufgerundet

Aufgabe 5a (Min-Heap Zeichnen)

add(4), add(2), add(5), removeMin(), add(1), add(3), removeMin()
add(7),
add(10)



Aufgabe 5b (Max-Heap Zeichnen)

add(4), add(7), add(10), add(2), add(5), removeMax(), add(8), removeMax()