1. Es sei $N(r,k)$ die Anzahl der Knoten in einem Binomialbaum, die k Kanten von der Wurzel entfernt sind, wobei der Wurzelrang des Baumes r ist.

   Zeigen Sie, dass $N(r,k) = \binom{r}{k}$.

2. Zeigen Sie, dass in einem "Hollow Heap" jeder Baum mit Wurzelrang r mindestens $F_{r+3} - 1$ viele Knoten besitzt, wobei F_r die r-te Fibonacci-Zahl ist (mit $F_0 = 0$ und $F_1 = 1$).

   Falls Sie Fibonacci-Zahlen nicht kennen sollten, informieren Sie sich.

3. Hollow-Heaps erlauben einen Min-Prioritätenschlange mit den Operationen Insert, DecreaseKey, Min in jewils konstanter amortisierter Zeit zu realisieren und MinDelete in logarithmischer Zeit. Es wäre nun interessant, diesen Operationen noch die Operation IncreaseKey hinzuzufügen, die dann auch in konstanter amortiserter Zeit funktionieren sollte.

   Warum ist das nicht so ohne Weiteres möglich?

4. Wie würden Sie in C++ oder Java (oder einer ähnlichen imperativen Sprache) eine Klasse "BinomialHeap" deklarieren? Insbesondere, wie sähe die Deklaration der Baumknoten aus, und wie interagieren diese Baumknoten mit den Objekten, die in der Prioritätenschlange verwaltet werden sollen? Wie sieht diese Interaktion bei einem DecreaseKey(x, H, k) aus (bei Objekt x, neuem Schlüssel k und Prioritätenschlange H)?

   Wie wäre es mit "HollowHeap"? Insbesondere, was bedeuten die hohlen Knoten für die Interaktion zwischen Baumknoten und Objekten?

5. Überlegen Sie sich ein Programm, das die Consolidate-Operation für Hollow-Heaps mit hinreichendem Detail implementiert.

6. Bei Hollow-Heaps wäre es auch vorstellbar, bei einer DecreaseKey Operation den betreffenden Knoten leer zu machen einfach einen neuen Knoten mit dem verminderten Schlüsselwert in die Wurzelliste einzufügen (ohne Teilbäume des geleerten Knotens zu transferieren). Warum ist das eher keine gute Idee?