# Data Structures - Assignment no. 4

**Remarks:**

- **Write both your name and your ID number very clearly on the top of the exercise. Write your exercises in pen, or in clearly visible pencil. Please write *very* clearly.**
- **Recall that 80% of the theoretical exercises must be submitted. The exercises can and must be worked on and submitted alone.**
- **Give correctness and complexity proofs for every algorithm you write.**
- **For every question where you are required to write pseudo-code, also explain your solution in words.**

1. (a) On an initially empty Fibonacci heap, carry out the following sequence of operations: insert(27), insert(17), insert(19), insert(20), insert(24), insert(12), insert(11), insert(10), insert(14), insert(18), deletemin, decreasekey(19, 7), delete (17), decrease-key(24,5), deletemin.

   After each operation, draw the resulting structure of the Fibonacci heap. (Whenever elements enter the root list, they are inserted to the right of the current minimum. Successive linking of the root list also starts with the element to the right of the removed minimum and continues cyclicly.)

   (b) Let $H$ be a Fibonacci heap consisting of a single tree. The tree consists of three nodes with keys $x = 7$, $y = 25$, and $z = 31$, such that $y = parent[z]$, and $x = parent[y]$. The node that contains $y$ is marked. Find a sequence of legal operations (insert, deletemin, decreasekey, delete) by which $H$ could have emerged from an empty Fibonacci heap.

2. Professor Cohen has devised a new data structure based on Fibonacci heaps. A *Cohen heap* has the same structure as a Fibonacci heap and implements exactly the same abstract data type. The implementations of the operations are the same as for Fibonacci heaps, except that insert and meld perform successive linking as their last step. What are the worst-case and amortized running times of all operations on Cohen heaps?

3. Suppose we modify the cascading-cut rule to cut a node $x$ from its parent only when it loses its *third* child. (In the standard implementation, a node is cut from its parent when it loses its *second* child.) What would be the maximum degree of a node in a Fibonacci heap with $n$ elements? What would be the amortized running time of each operation? Prove your answers.

4. Professor Levi proposed the following pseudo-code for the FIB-DELETE procedure.

```
LEVI-DELETE(H, x)
1. if x = min[H]
2.         then FIB-DELETE-MIN(H)
3.         else y = p[x]
4.                 if y ≠ NIL
5.                         then CUT(H, x, y)
6.                                 CASCADING-CUT(H, y)
```

7. add $x$'s child list to the root list of $H$
8. remove $x$ from the root list of $H$

The call CUT$(H, x, y)$ disconnects $x$ from its parent $y$ and adds $x$ to the root list. The call CASCADING-CUT$(H, y)$ starts a sequence of cascading cuts at $y$.

- The professor claims that line 7 can be performed in $O(1)$ actual time. What is wrong with this assumption?
- Give a good upper bound on the actual time of LEVI-DELETE when $x \neq \min[H]$. Your bound should be in terms of degree$[x]$ and the number $c$ of calls to the CASCADING-CUT procedure.
- Let $H'$ be the Fibonacci heap that results from an execution of LEVI-DELETE$(H, x)$. Assuming that node $x$ is not a root, bound the potential of $H'$ in terms of degree$[x]$, $c$, the number of trees in $H$ denoted by $t(H)$, and the number of marked node in $H$ denoted by $m(H)$.
- Conclude that the amortized time for LEVI-DELETE is asymptotically no better than for FIB-HEAP-DELETE, even when $x \neq min[H]$.

5. We wish to augment a Fibonacci heap $H$ to support two new operations without changing the amortized running time of any other Fibonacci-heap operations.

   - Give an efficient implementation of the operation FIB-CHANGE-KEY$(H, x, k)$, which changes the key of node $x$ to the value $k$. Analyze the amortized running time of your implementation for the cases in which $k$ is greater than, less than, or equal to key$[x]$.
   - Give an efficient implementation of FIB-PRUNE$(H, r)$, which deletes $min(r, n[H])$ nodes from $H$. Which nodes are deleted should be arbitrary. Analyze the amortized running time of your implementation. (Hint: You may need to modify the data structure and potential function.)

6. What is the actual (not amortized!) worst-case running time of a single DecreaseKey operation on a Fibonacci heap with $n$ elements (in $O$-notation)? Justify your answer.