1 Algorithms and Data Structures 2

1.1 Problem 1

1. **Give the important differences between a normal priority queue and an addressable priority queue.**

   Normal priority queues maintain a set M of Elements with Keys under the following operations:
   $M.build(\{e_1, ..., e_n\}) : M := \{e_1, ..., e_n\}$
   $M.insert(e) : M := M \cup \{e\}$
   $M.min : return\ min\ M$
   $M.deleteMin : e := min\ M; M := M\ \{e\}; return\ e$

   Addressable priority queues additionally support operations on arbitrary elements addressed by an element handle h:
   insert: As before but return a handle to the element inserted.
   $remove(h)$: Remove the element specified by handle h.
   $decreaseKey(h, k)$: Decrease the key of the element specified by handle h to k.
   $Q_1.merge(Q_2) : Q_1 := Q_1 \cup Q_2; Q_2 := \emptyset$.

2. **Compare the running time of a merge operation for pairing heaps and binary heaps.**

   Pairing Heap: merge $O(1)$
   The minPtr is updated and an addressable priority queue is simply attached to the forest.

   Binary Heap: merge

1.2 Problem 2

1. **Prove the general lower bound for addressable priority queues of Ω(log n) for deleteMin under the condition that insert runs in constant time.**

2. **Why does this bound not have to hold if insert is allowed to consume more time?**

1.3 Problem 3

For a very big festival, you are responsible for the bar. For this task, you designed a bar robot (aka Bender) that mixes excellent cocktails. The ingredients are stored in big boxes. But exactly here is the problem: you have to make sure that none of the boxes runs empty. On the other hand, you also want to enjoy the evening and not constantly check all of the boxes. To deal with this problem, there is only one solution: a refill display! Unfortunately, the only displays available are the ones that can display one line only. It is clear that this line should display the most urgently need ingredient. The goal is to design an algorithm that keeps the display up to date.

1. **As a base for you algorithm, think about a data structure to efficiently implement your algorithm. Assume that the number of ingredients for a specific cocktail is significantly smaller than the number of ingredients totally available.**
2. **Design a function MixDrink(recipe), that operates on your data structure. Give Pseu- docode. Your data structure has to be updated! Other functions of the robot don't have to be "programmed".**
3. **When a box is exchanged, your data structure has to be updated as well. Describe what consequences the exchange has on your data structure.**

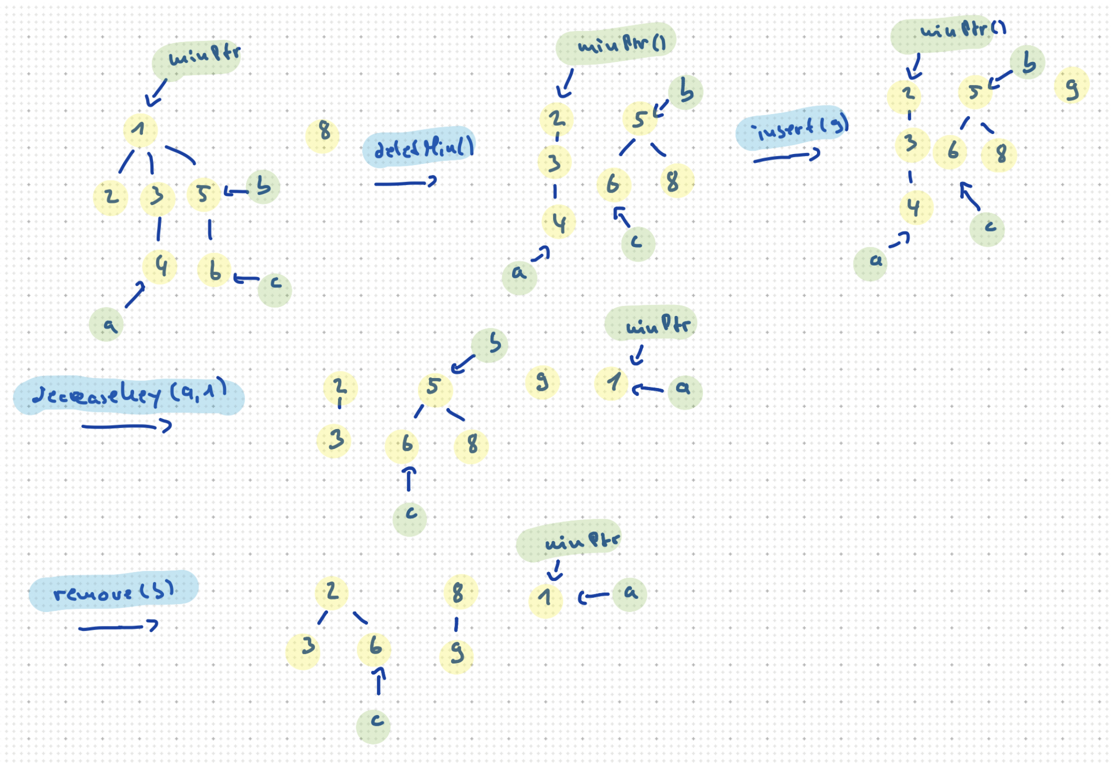
1.4 Problem 4

The definition of a pairing heap does not state which elements are neighbors. For the solution of the next tasks we assume a sorted list of roots (by insertion order) and neighborhood induced by this. Given is a Pairing Heap in the following state.

1. **Give a very short sequence of operations that creates this state.**

   insert(0)
   insert(1)
   insert(2)
   insert(3)
   insert(4)
   insert(5)
   insert(6)
   deleteMin()
   insert(0)
   deleteMin()
   insert(0)
   deleteMin()
   insert(8)

2. **Execute the following operations step by step on the given heap and draw the intermediate state of the heap after each operation: deleteMin(), insert(9), decreaseKey(a,1), remove(b).**


minPtr
deleteMin()
minPtr()
insert(9)
minPtr()
decreaseKey(a,1)
minPtr
remove(5)
minPtr