Universitetet i Oslo
Institutt for Informatikk
I. Yu, D. Karabeg

INF2220: algorithms and data structures
Series 4

Topic Priority Queues and Heaps (Exercises with hints for solution)

Issued: 14. 09. 2016

Classroom

Exercise 1 Draw the structure of a binary heap

1. after these priorities have been inserted:
   `19, 34, 23, 16, 54, 89, 24, 29, 15, 61, 27`.
2. after two `deleteMin` operations.

Solution: One may say: the structure conditions ("complete" binary tree) makes the procedure necessarily *deterministic* (if the order of insertions (and deletions) is given (and all priorities are different)). So there can be only *one* correct solution. For analogous exercises for BSTs, there are logically two different implementations for deletion, which give sligtly different outcomes.

A practical consequece: when doing such an exercise, there is no obvious "shortcut", i.e. a way to fill in the numbers into the tree without doing it step by step. □

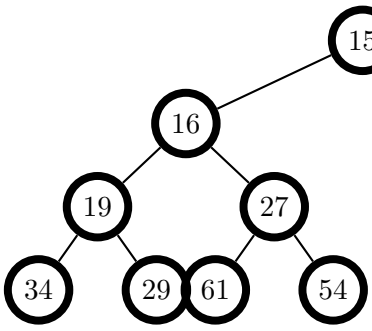


Figure 1: Solution Exercise 1.1

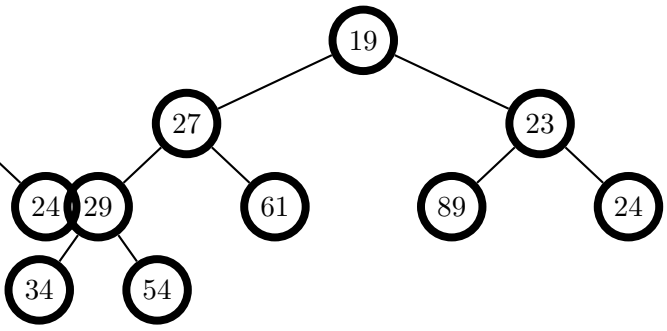


Figure 2: Solution Exercise 1.2

Exercise 2 (Binary heap) Draw the structure of a binary heap

1. after these priorities have been inserted:
   `10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, 2`.

2. after three `deleteMin` operations.

Solution: [of Exercise 2] There's nothing new in this exercise, it's the "same" as the previous one.

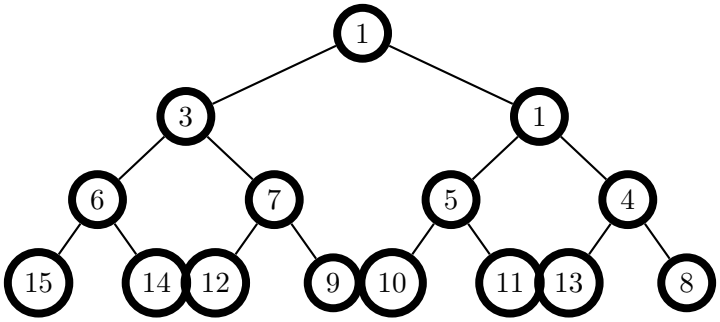


Figure 3: Solution Exercise 2.1

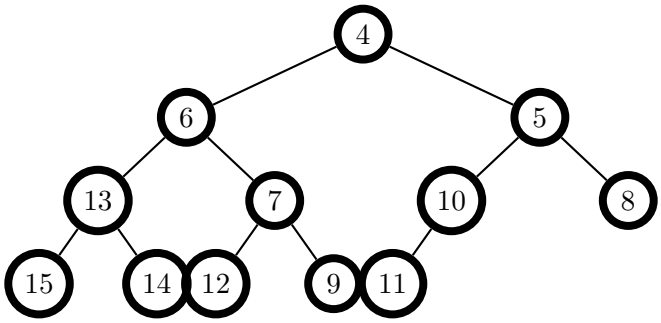


Figure 4: Solution Exercise 2.2

□

Exercise 3 (Leftist Heap I) Is the tree a *leftist heap*? Explain your answer.

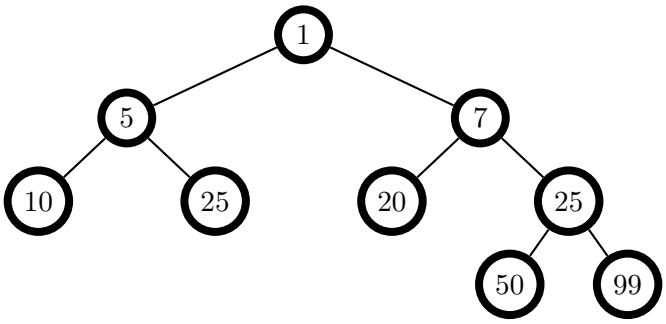


Figure 5: Exercise 3

Exercise 4 (Leftist Heap II)

1. Merge the two leftist heaps. Remember to maintain the structure of a leftist heap.
2. Perform a `deleteMin` operation on the resulting leftist heap from Exercise 4.1

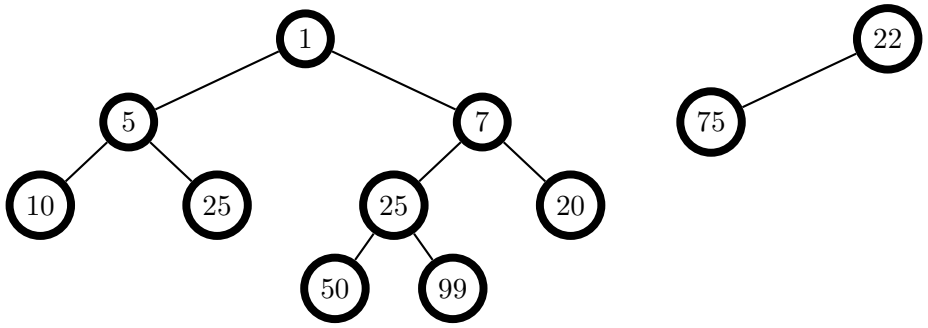


Figure 6: Exercise 4.1

Solution: Solution for Exercise 3 and 4. For the first tree: it looks already suspicious, but of course that's not an answer. There are two conditions on such trees. One is the condition in the numbers (the "priorities"). This one is alright. As for the structural condition. We need to check the official definition, i.e., the condition on *npl*'s for all nodes. One point where the condition is violated is node 7 and it's children: the null pointer lenght of 7's subtree is smaller than that of the right one, that is, $npl(20) = 0 < 1 = npl(25)$. Note in particular, that *locally* at the root node 1, the structural condition for leftist heaps is ok! Both direct children have a nlp of 1! A correct answer in an exam would require that one points to (at least) one node where where the condition is concretely broken. Of course, also node 1 is not a leftist-heap since it's right-subtree is not, but the full explanation must locate the point where it directly breaks.

The fact that node 7 is the *only* place where the ordering requirement is broken means: one can repair it (in constant time) by simply swapping the two subtrees (even if after the swapping, the right-hand subtree still looks larger, now it's ok). That's the tree which is used in the second part of the exercise to do a merge.

```
merge (1,22)                                  =>
    merge (7,22)                                =>
      merge (25,22)                               =>
       merge (25,⊥)                             // the right had of 22 = ⊥
       22[75,25]                                  <=
   7[20[50,99], 22[75,25]]                      <=
 1[7[20[50,99], 22[75,25]], 5[10,15]] <=       // that's not leftist ⟹ swap
 1[22[75,25]], 5[10,15],7[20[50,99] ]
```

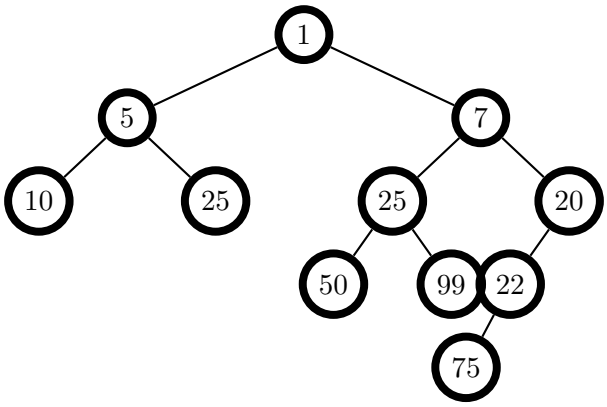


Figure 7: Result of the merge (Ex. 4.1)

For the third exercise:

Here's one easy way (which is also the one from the book). Now that we have merging, deleting a node splits the heap into two. Since the recursive definition of leftists heaps (all decent tree definitions are recursive), the two sub-trees are again leftist heaps, so we can merge them with the procedure we have learnt. The result is given

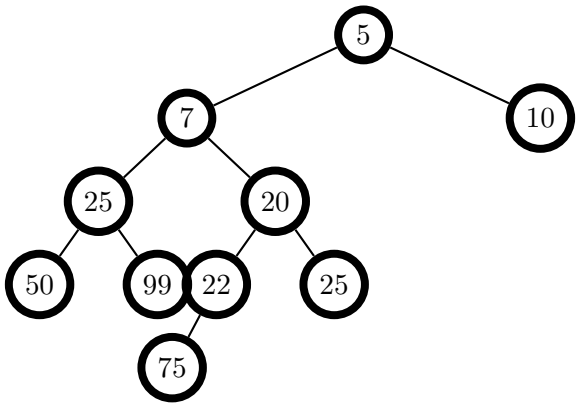


Figure 8: Result of the merge (Ex. 4.2)

Another idea is; to use the delete minimum as is done for the "perfectly balanced" tree. That's also possible, actually, in a way that is simpler, because the requirements (the balance condition) is not as strict. The way that deletion for the min-heaps is done: the "perfect balance" must not be destroyed. That's done by replacing the root by "the last leaf" on the bottom (from left to right). Since the tree is stored in an array, and since we need to remember which "is the last leaf"[1] we have direct access to that and can immediately copy that to the root and then percolate it down. Note that percolating down there does not change the structure. We can achieve something like that also here, except that it's not so easy to find a leaf, we can of course search for it. The main question is: which leaf should we take.

□

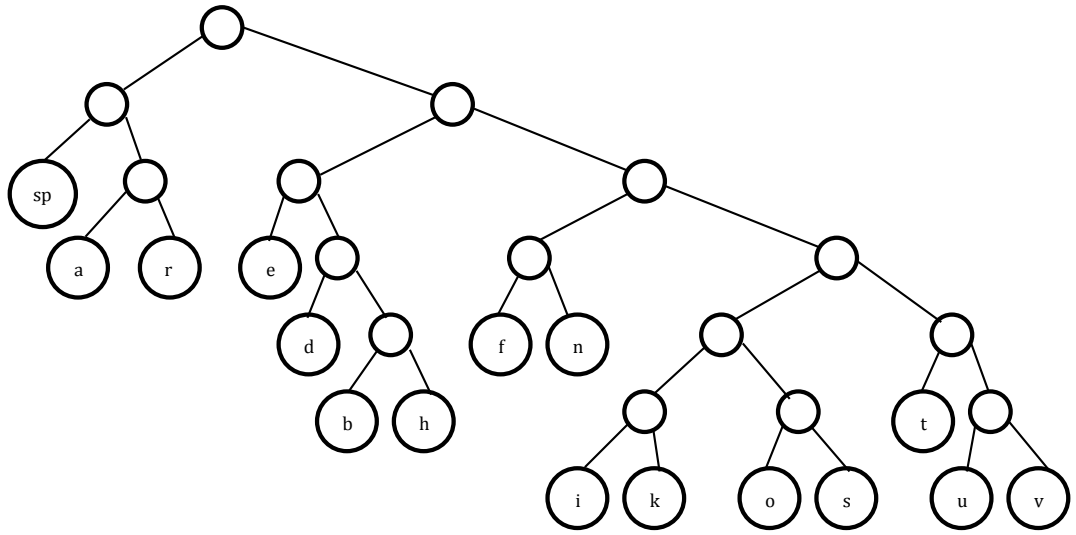
Exercise 5 Show that the element with the lowest priority (highest number)

1. can be found in one of the leaf-nodes
2. that it can be found in an arbitrary leaf-node (it's sufficient to show that the requirements of a binary heap holds with different arrangements)
3. show that a binary heap with an even number of elements (N) will have exactly $N/2$ leaf-nodes.

   This assignment is perhaps a bit difficult, try to think about the data-structure we used to represent a binary heap

Exercise 6 A file contains only spaces and digits in the following frequency: space (9), a (5), b (1), d (3), e (7), f (3), h (1), i (1), k (1), n (4), o (1), r (5), s (1), t (2), u (1), v (1). Construct the Huffman code.

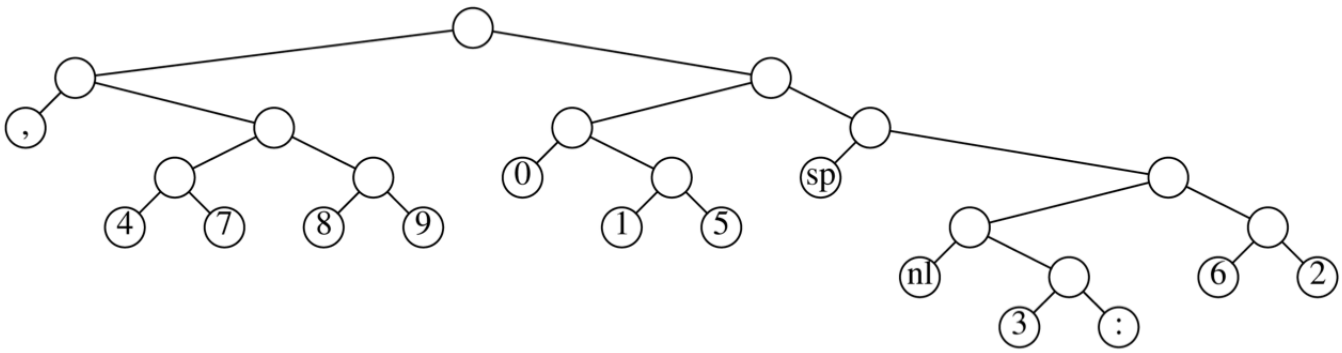
Solution:



One possible tree:

Exercise 7 (Huffman) A file contains only colons, spaces, newlines, commas, and digits in the following frequency: colon (100), space (605), newline (100), comma (705), 0 (431), 1 (242), 2 (176), 3 (59), 4 (185), 5 (250), 6 (174), 7 (199), 8 (205), 9 (217). Construct the Huffman code.

Solution: One possible tree:



Lab

Exercise 8 Implement the interface below with a BinaryHeap.

```
interface HeapInterface<T>{

  public void insert(int pri, T o);
  public T deleteMin();
}
```

Exercise 9 Make a program which sorts a sequence of numbers using a heap.

[1]The alternative to store a special priority value like $\perp$ or *null* or similar to indicate that the node is not there is not too smart, even if that would work.

Exercise 10 A binary heap is a special case of a `DHeap`, where the heap is allowed to have D children. i.e., a binary heap is a `DHeap` with $D == 2$. Implement a

```
TernaryHeap  (DHeap with D == 3)
```

HINT:

now we get three children to examine to find out whether or not they have a higher priority then ourselves, functions to calculate where the elements can be found in the array is given below.

```
private int dad(int pos){
  return (int) Math.ceil( ((double) pos - 1.0) / 3.0);
}

private int first(int pos){ return  (3 * (pos - 1)) + 2; }
private int second(int pos){ return (3 * (pos - 1)) + 3; }
private int third(int pos){ return  (3 * (pos - 1)) + 4; }
```

Exercise 11 What is the maximum height of a `TernaryHeap` with N elements?