CS200 Functional Data Structures
Assignment 4

Beep Beep: Anas-am00634 and Affan-sa00310

April 7, 2016

Problem 1. Meldable Heap

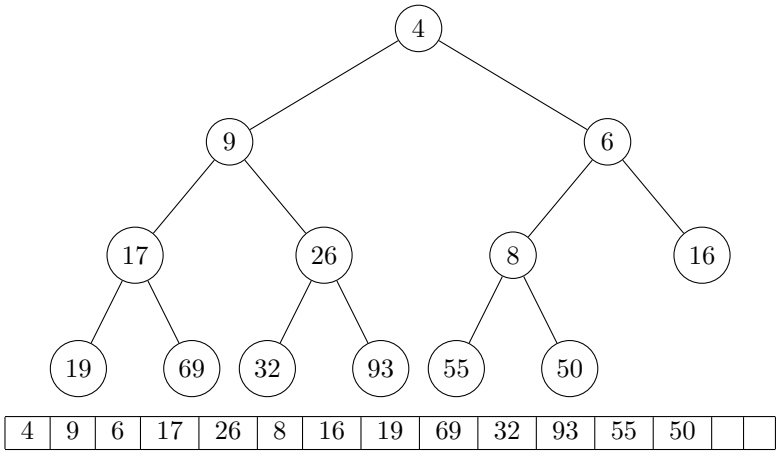
Please see the ass4.hs file in the same folder.

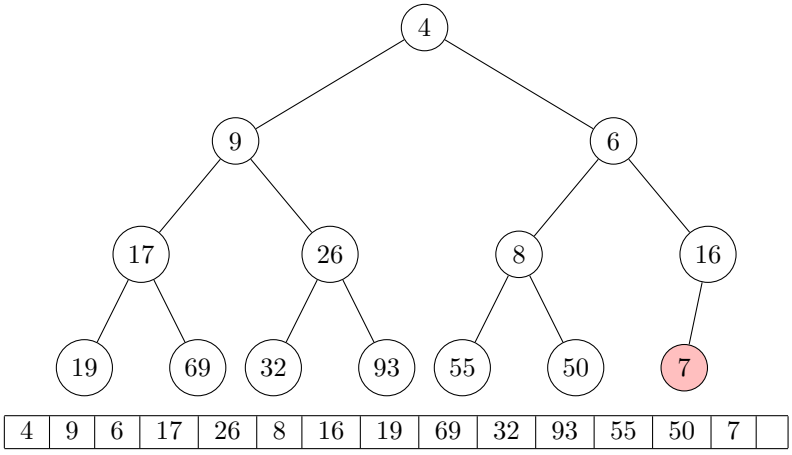
Problem 2. Heap Sort

Please see the ass4.hs file in the same folder.

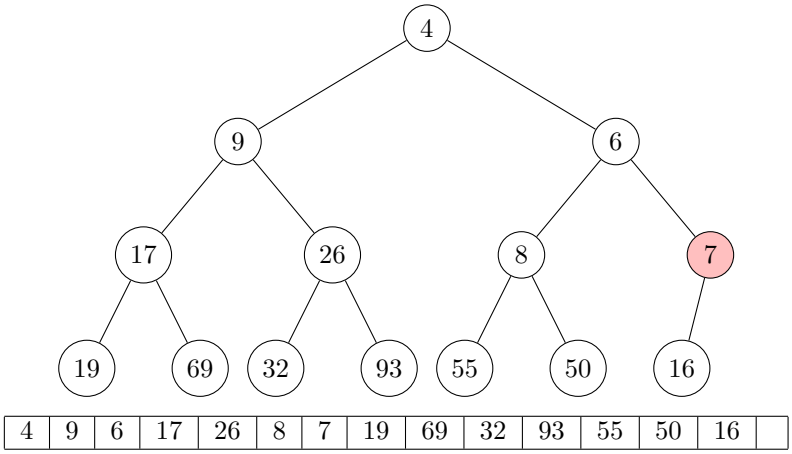
Problem 3. Heap Operations

1. **10.1** Adding 7 to the BinaryHeap.

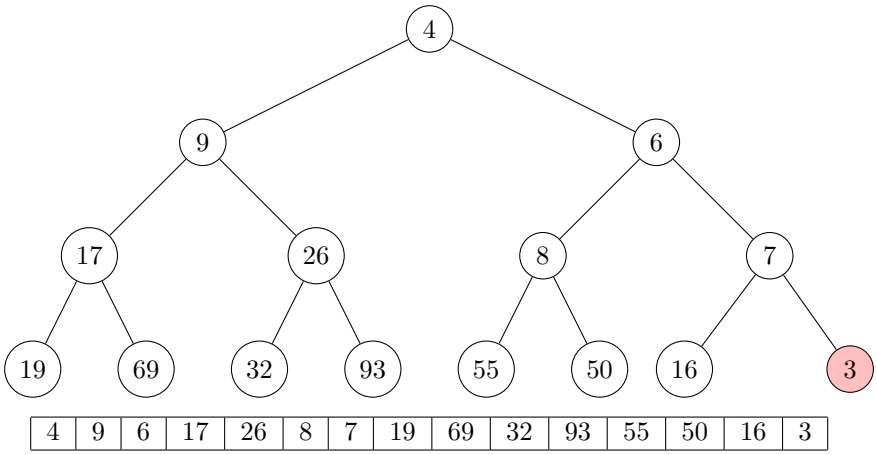



4
9
6
17
26
8
16
19
69
32
93
55
50
7
4 9 6 17 26 8 16 19 69 32 93 55 50 7

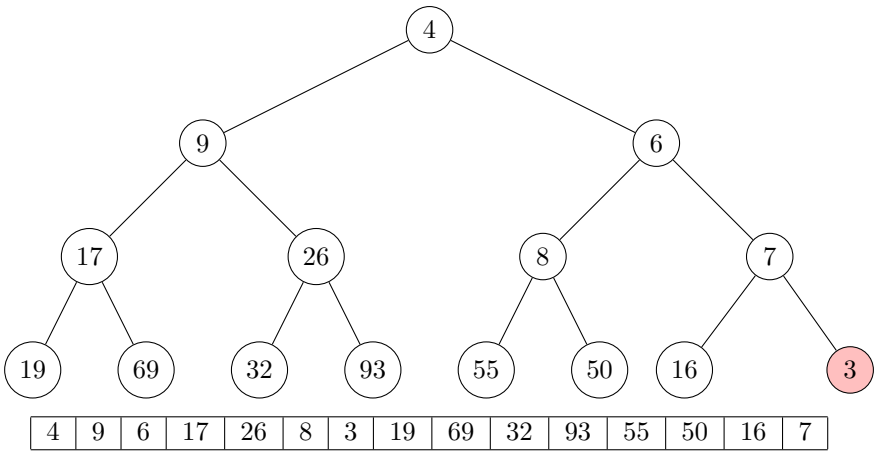
Bubble up.

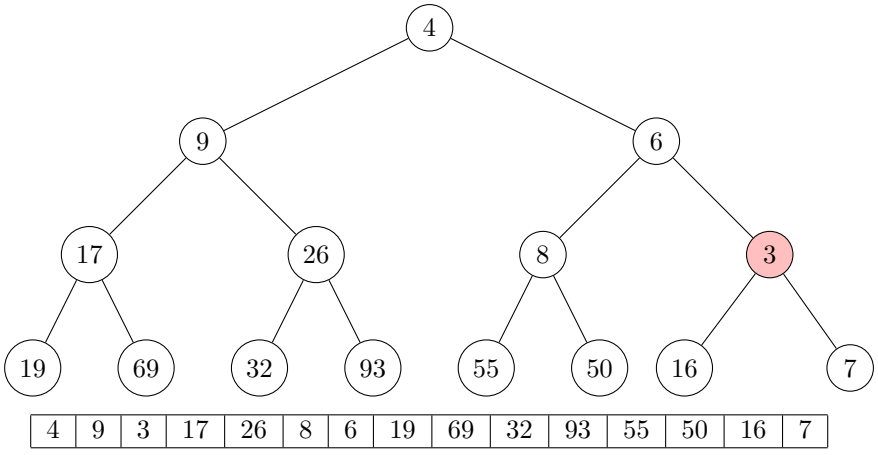


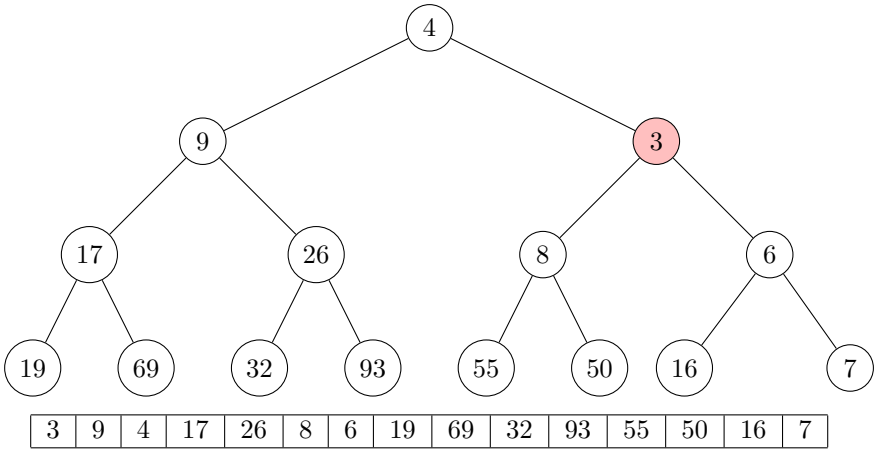
Adding 3 to the BinaryHeap

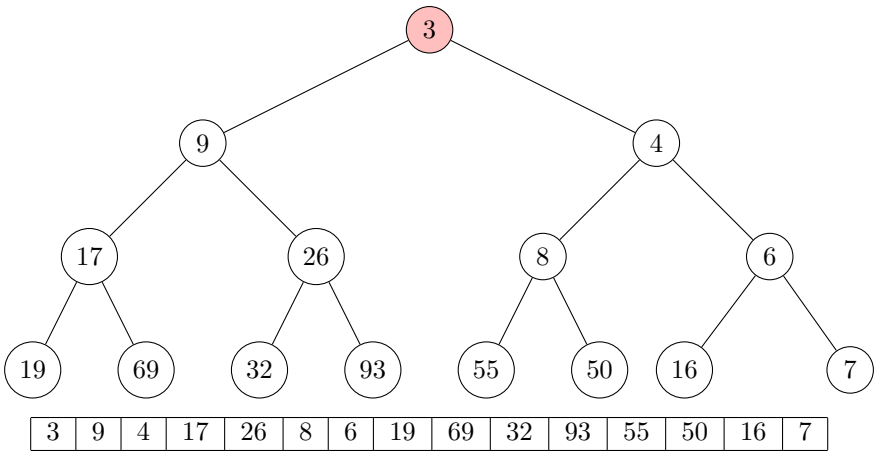


Bubble up


4
9
6
17
26
8
7
19
69
32
93
55
50
16
3
4 9 6 17 26 8 3 19 69 32 93 55 50 16 7

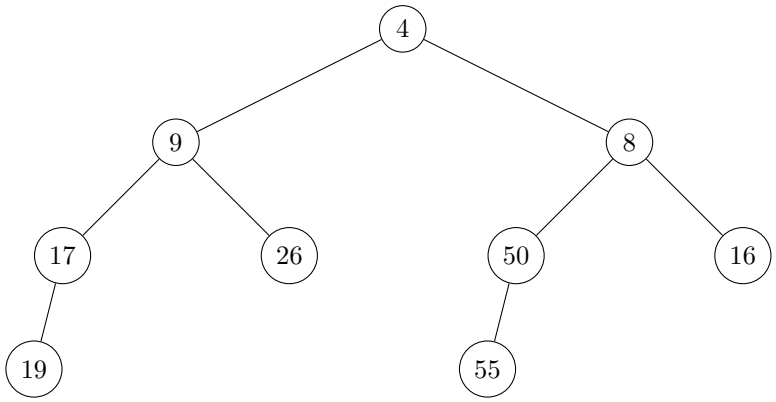

4
9
6
17
26
8
3
19
69
32
93
55
50
16
7
4 9 3 17 26 8 6 19 69 32 93 55 50 16 7


4
9
3
17
26
8
6
19
69
32
93
55
50
16
7
3 9 4 17 26 8 6 19 69 32 93 55 50 16 7

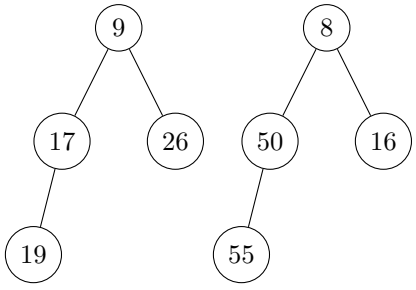


2. **10.7**

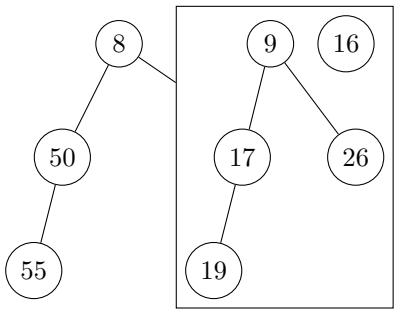
(a) We begin with the tree provided:



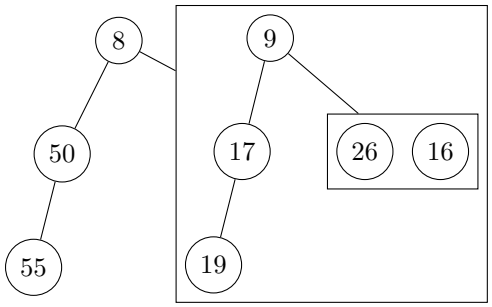
(b) Remove the first item, we get the two following trees:



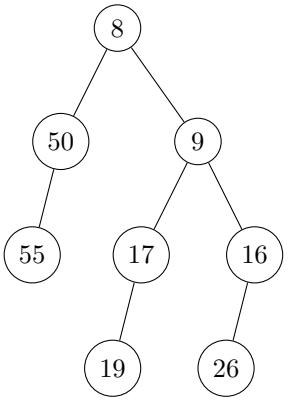
(c) We merge the two trees merge(h1.right,h1.left). The algorithm takes care of the cases where node value x1¿x2, therefore, we are not showing the swaps in our diagrams.



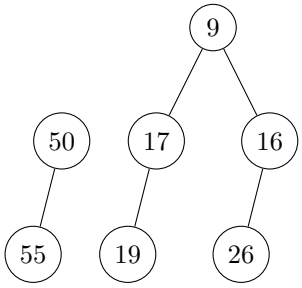
(d) Next recursive call, we do a coin flip for the node 9, because it is less than 16. We imagine that we get right on the random coin flip.



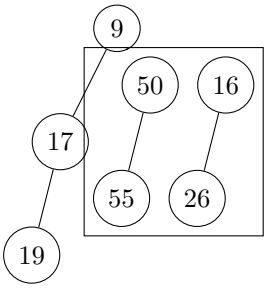
(e) In the next recursive call, 16 is less than 26, therefore the resulting tree becomes;



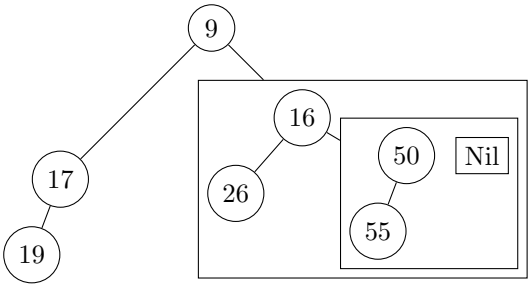
(f) Remove(8). First merge call would call merge on 8.left and 8.right



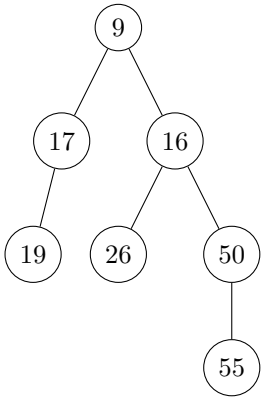
(g) Since 9 is less than 50, we would swap and then call the recursive function on a random child of 9 and the other tree. We assume we get random 1, and therefore, merge on the right child of 9.



(h) Next recursive call. We assume random to be 1 and therefore call is merge (16.right,h2) where h2 is the tree with node 50.



(i) Since right of 16 was empty, we insert the case trivially.



3. **10.9** Show how to find the second smallest value in a BinaryHeap or MeldableHeap in constant time.

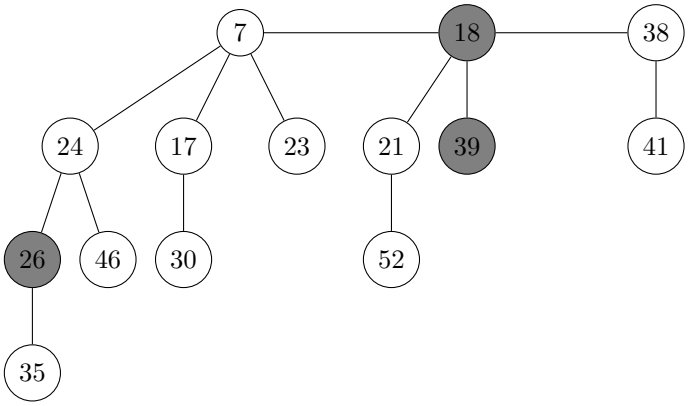
```
def secondSmallest(binaryHeap):

if binaryHeap = Empty:
generate ERROR
else:
if binaryHeap is singleton:
generate ERROR
else:  // binary tree is not empty and greater than equal to 2
min(root.left,root.right)
```

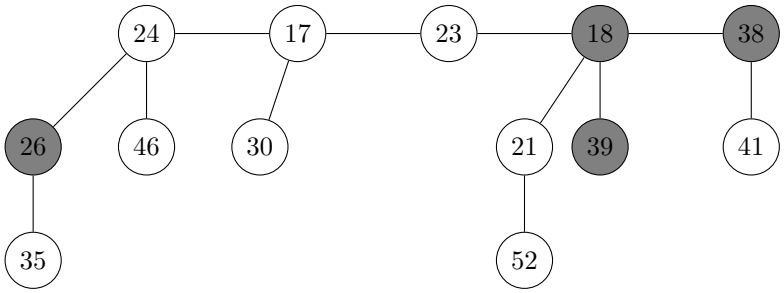
Problem 4. Fibonacci Heap

Solve Q19.2-1 on Page 518 in the linked reference on Fibonacci Heaps. When going over the notes, you may skip over the theoretical discussion and assume: $\mathcal{D}(n)$ to be $log(n)$.

1. H.min is pointing at node 7.

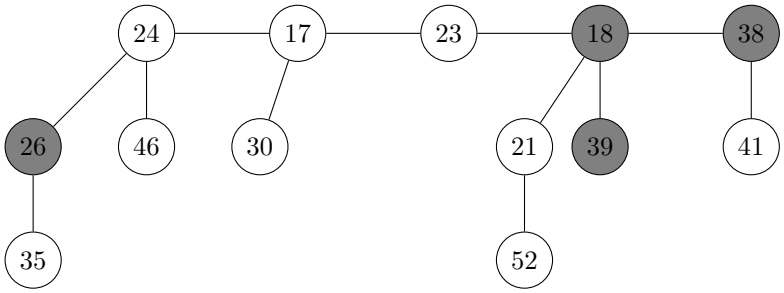


2. H.min is pointing to Node 18.



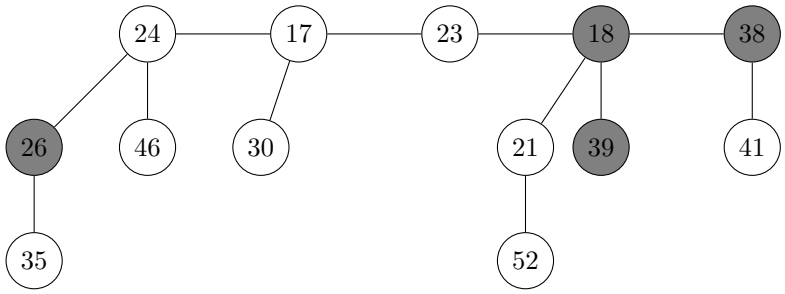
3. We create a degree array of dimension 4, because number of elements = log 13 3rd element of array is pointing to Node 18.

nil	nil	18	nil



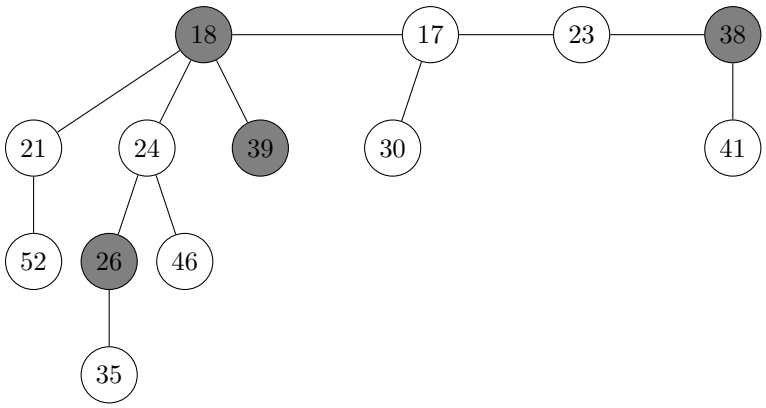
4. We move right from degree array, because there are no other nodes pointing to 18. Therefore, we move to the right of 18, and that is degree 1. Array[1] will not point to node 38.

nil	38	18	nil



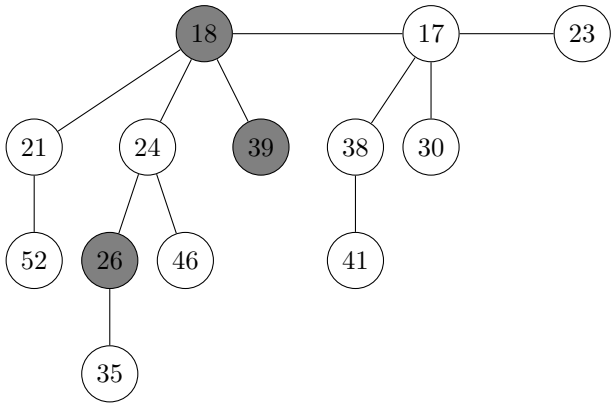
5. We move right again. This time, a node with the same degree as degree[2] is found which was 18. We check if 26¡18 and since it is not, we consolidate 18 with 26. 18 becomes parent and 26 becomes its child. 18 stays in the degree list. H.min is still pointing at 17. Since 18 now has 3 children, degree[3] is now 18, and degree[2] is changed to nil.

nil	38	nil	18



6. We move right again. This time, a node with the same degree as degree[1] is found which was 38. We check if 38¡17 and since it is not, we consolidate 17 with 38. 17 becomes parent and 38 becomes its child. H.min is still pointing at 17. Since 17 now has two children,Array[2] becomes 17. Since 38 was marked, we unmark it after consolidate.

nil	nil	17	18



7. We move right again. There is a node of degree 0 (23), and we add it to the degree list. Since we have traversed the whole degree list, and each element's degree is unique, we return the resulting tree which is the same tree. In the next step, we look for the minimum element again, and H.min still points to 17.

23	nil	17	18