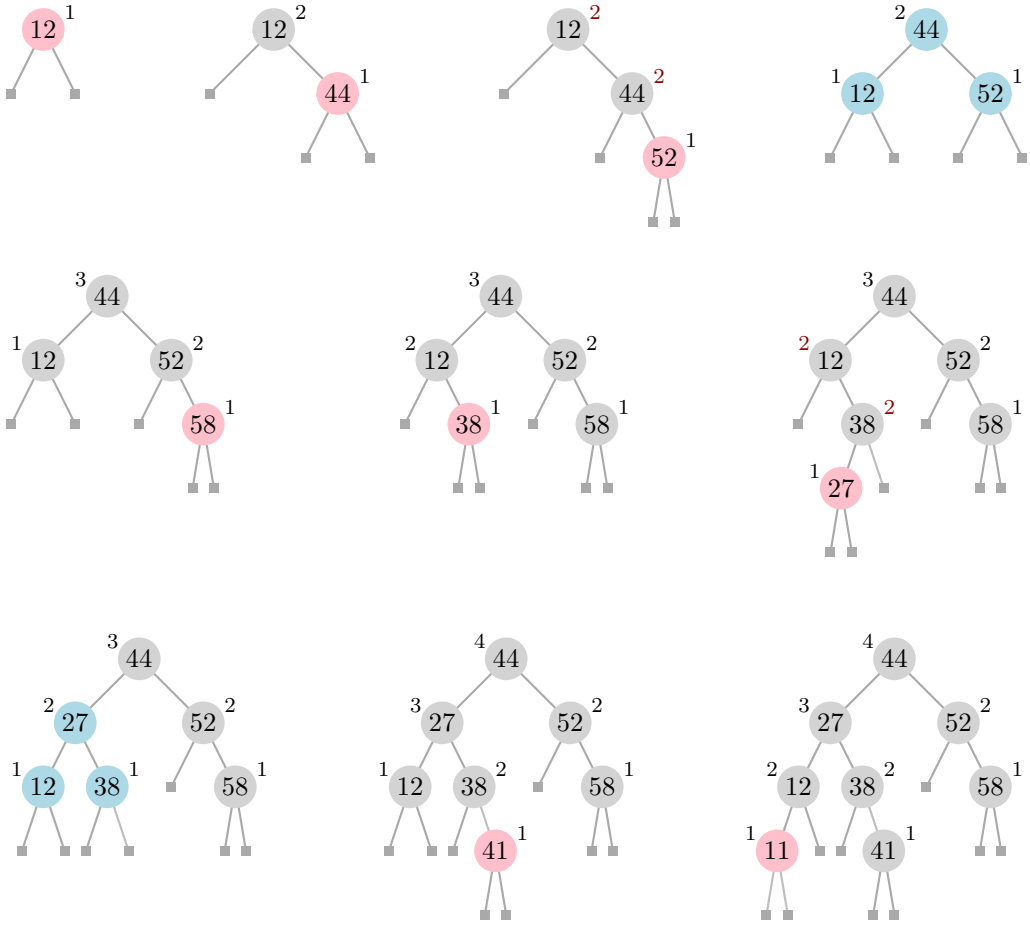
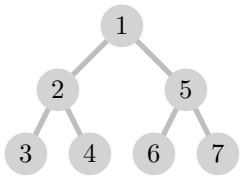
R-4.2 Consider the insertion of items with the following keys in the given order into an initially empty wavl tree: 12, 44, 52, 58, 38, 27, 41, 11. Draw the final tree that results.



R-5.11 Is there a (min-)heap T storing seven distinct elements such that a preorder traversal of T yields the elements of T in sorted order?

Solution. Yes, the following one:



•

How about an inorder traversal?

Solution. In an inorder traversal, a parent is always listed after its left child. But in a min-heap, a parent is always smaller than its left child. So the parent, which is smaller, is listed after its left child, which is bigger. So an inorder traversal of a min-heap is not sorted (in increasing order). •

How about a post order traversal?

Solution. No, for the very same reason as for inorder traversal. For post-order traversal, you can also replace the word "left" by "right" in the previous argument, and the argument will still be true. •

A-5.1 Describe a method for maintaining the median of an initially empty set, S, subject to an operation, insert(x), which inserts the value x, and an operation, median(), which returns the median in S. Each of these methods should run in at most $O(\log n)$ time, where n is the number of values in S.

Solution. There are two solutions. This one uses a min-heap and a max-heap.

```
class median-structure:
    # initially
    lo = an empty max-heap
    hi = an empty min-heap
    # invariant: size(lo) - size(hi) is always 0 or 1

    def median():
      # median-structure is empty
      if size(lo) is 0:
        return null
      # median-structure has odd number of elements
      if size(lo) > size(hi):
         return lo.max()
      # median-structure has even number of elements
      return (lo.max() + hi.min())/2

    def insert(x):
      if median() is null or x <= median():
        lo.insert(x)
        if size(lo) - size(hi) == 2:
          y = lo.remove-max()
          hi.insert(y)
      else:
        hi.insert(x)
        if size(lo) - size(hi) == -1:
          y = hi.remove-min()
          lo.insert(y)
```

If we only have min-heaps available, we can define max-heaps in terms of them.

```
class max-heap:
    # initially
    h = an empty min-heap

    def insert(x):
      h.insert(-x)

    def max():
      return - h.min()

    def remove-max():
      return - h.remove-min()
```

The second solution uses a balanced binary search tree in which each node contains an additional piece of data called size, which stores the size of the subtree at that node. (Technically this new data structure is called an order-statistic tree, where order is what we call size).

The procedure for finding the median is similar to quickselect for finding the k-th sorted item. Given k, start at the root, visit each of its children to find the size of their subtrees. If k is equal to the size of the left subtree, return that node. If k is strictly less than the size of the left subtree, go to the left subtree and repeat. Otherwise, decrease k by the size of the left subtree, go to the right subtree and repeat. To find the median, if n is odd, do this procedure using $\lfloor n/2 \rfloor$. If n is even, do this procedure twice with $k = n/2$ and $k = n/2 + 1$ and take their average.

To insert x, do a typical insertion of the binary search tree. Keep a pointer to to the inserted node before all the necessary rotations are done. Then use the pointer to go to the node. Update the size of that node by summing the size of its two children, plus 1 (for the node itself). Then, go to the node's parent and update its size. Repeat this update the procedure all the way up to the root of the tree. •

R-6.5 Draw the 11-item hash table resulting from hashing the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5, using the hash function $h(i) = (2i + 5) \mod 11$ and assuming collisions are handled by linear probing.

Solution. The hash values are

key	12	44	13	88	23	94	11	39	20	16	5
hash	7	5	9	5	7	6	5	6	1	4	4

The final hash table will have the following content

index	0	1	2	3	4	5	6	7	8	9	10
value	11	39	20	5	16	44	88	12	23	13	94

•