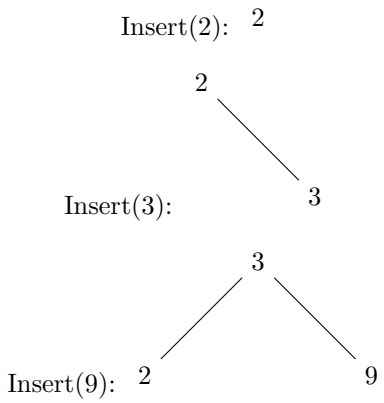
Homework Sheet 5

Author	Matriculation Number	Tutor
Abdullah Oğuz Topçuoğlu	7063561	Maryna Dernovaia
Ahmed Waleed Ahmed Badawy Shora	7069708	Jan-Hendrik Gindorf
Yousef Mostafa Farouk Farag	7073030	Thorben Johr

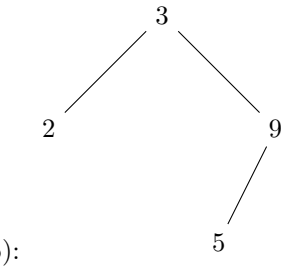
Exercise 1

We begin with an empty AVL tree and perform the following operations step by step:

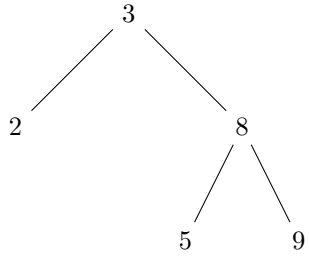
1. Insert(2)
2. Insert(3)
3. Insert(9)
4. Insert(5)
5. Insert(8)
6. Insert(6)
7. Remove(8)
8. Remove(5)
9. Insert(4)
10. Insert(5)
11. Remove(6)



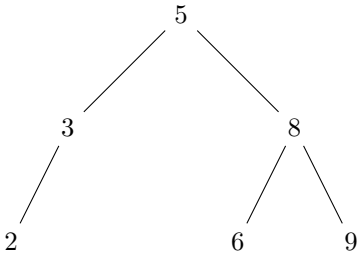
Insert(5):



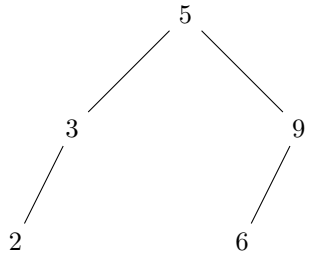
Insert(8):



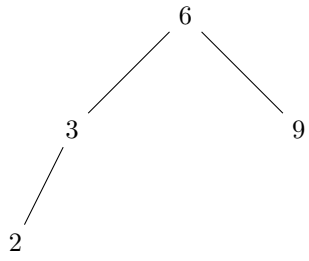
Insert(6):

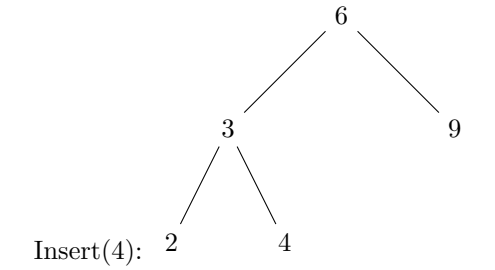


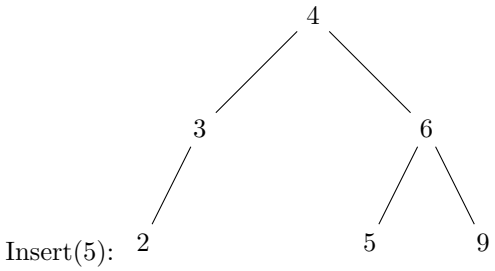
Remove(8):

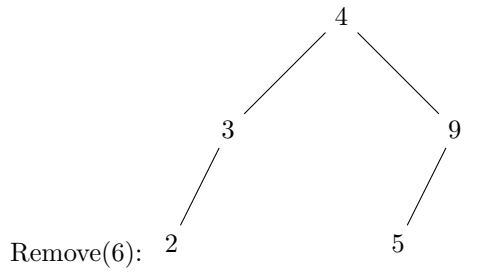


Remove(5):





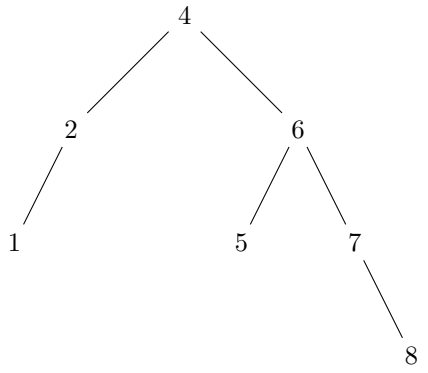




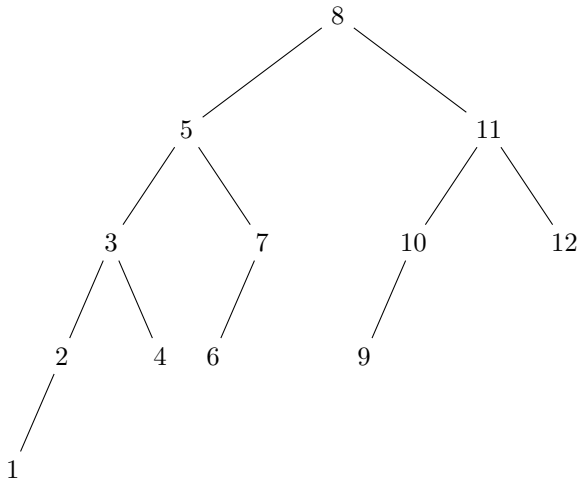
Exercise 2

(a)

An AVL tree of maximum height with 7 nodes:



An AVL tree of maximum height with 12 nodes:



(b)

Proof by Induction:

- **Base Case:** For height $h = 0$, the minimum number of nodes in an AVL tree is $F_{0+2} - 1 = F_2 - 1 = 1 - 1 = 0$, which is correct since an AVL tree of height 0 has no nodes.
- **Inductive Step:** Assume that for some height $h \geq 0$, the minimum number of nodes in an AVL tree of height h is $F_{h+2} - 1$. We need to show that this holds for height $h + 1$.
- Consider an AVL tree of height $h + 1$. To minimize the number of nodes, we can have one subtree of height h and another subtree of height $h - 1$ (since the height difference must be at most 1 for AVL trees).
- By the inductive hypothesis the minimum number of nodes in the subtree of height h is $F_{h+2} - 1$ and in the subtree of height $h-1$ is $F_{(h-1)+2} - 1 = F_{h+1} - 1$.
- Therefore the total minimum number of nodes in the AVL tree of height $h + 1$ is:

  $$N(h+1) = (F_{h+2} - 1) + (F_{h+1} - 1) + 1 = F_{h+2} + F_{h+1} - 1 = F_{(h+1)+2} - 1$$

  (using the Fibonacci property $F_n = F_{n-1} + F_{n-2}$).
- Thus by induction the statement holds for all heights $h \geq 0$.

Exercise 3

(a)

To implement the Min(h) operation, we can follow the left child pointers starting from the node represented by handle h until we reach a node that does not have a left child. This node will be the node with the smallest key in the subtree rooted at h since AVL trees are just BSTs with a constraint on the height of the nodes. The algorithm is as follows:

```
Handle Min(Handle h)
    current := h
    while current.left != NULL do
        current := current.left
    return current
```

The running time of this algorithm is O(log n) because in an AVL tree, the height is O(log n), and we may need to traverse from the root of the subtree down to the leaf.

(b)

To maintain the size of each subtree during Insert and Remove operations, we can update the size attribute of each node as we perform these operations.

- **Insert:** When inserting a new node, we start from the root and traverse down to find the correct position for the new node. As we traverse back up the tree, we increment the size attribute of each node by 1 to account for the newly added node. And when we do Rotate operations updating the size attribute of the nodes affected is just constant time because there is only three nodes affected by that in the Rotate operations.
- **Remove:** When removing a node, we first find the node to be removed. After removing the node, we traverse back up the tree and decrement the size attribute of each node by 1 to account for the removed node. And similar to Insert operation, when we do Rotate operations updating the size attribute of the nodes affected is just constant time because there is only three nodes affected by that in the Rotate operations.

Since both Insert and Remove operations involve traversing the height of the tree, which is O(log n) in an AVL tree, updating the size attributes can be done in O(log n) time as well.

(c)

To implement the Rank(k) operation, we can use the size attribute of each node to count the number of nodes with keys less than or equal to k. The algorithm is as follows:

```
int Rank(Key k)
    current := root
    result := 0
    while current != NULL do
        if k < current.key then
            current := current.left
        else
            result += (current.left != NULL ? current.left.size : 0) + 1
            current := current.right
    return result
```

The running time of this algorithm is O(log n) because in an AVL tree, the height is O(log n), and we may need to traverse from the root down to a leaf.