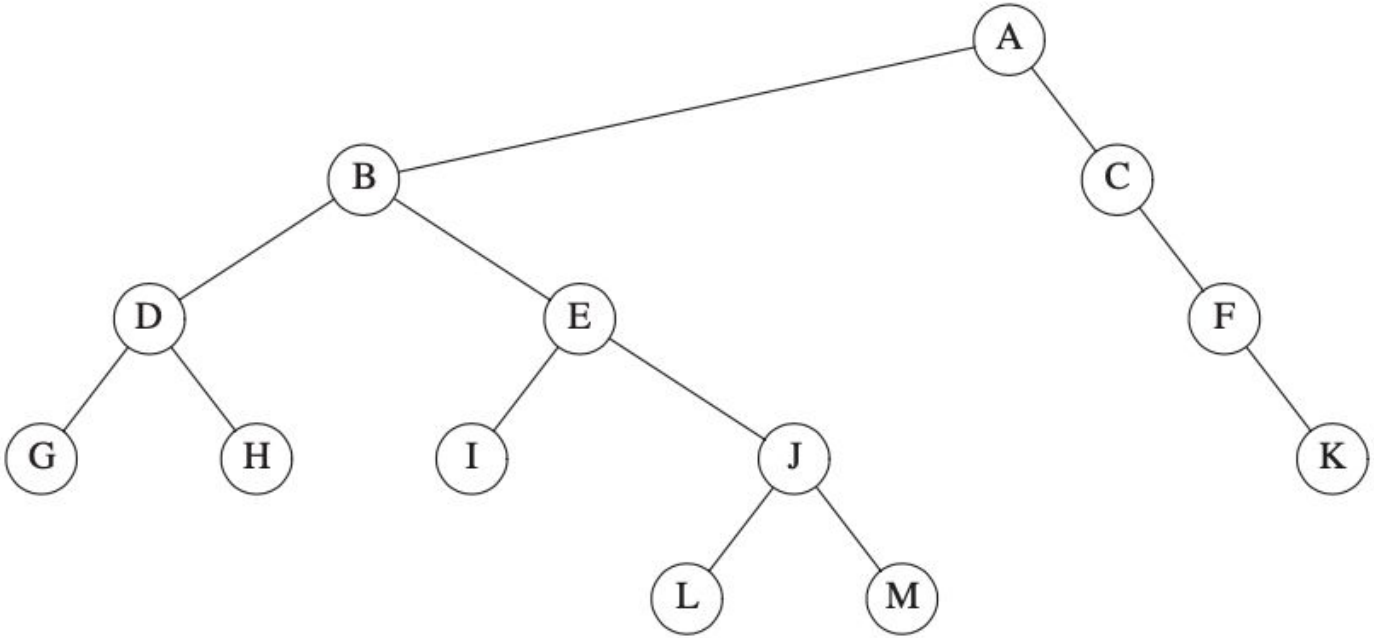
Avl Trees

Q1

Find the depth of the tree below



[1]

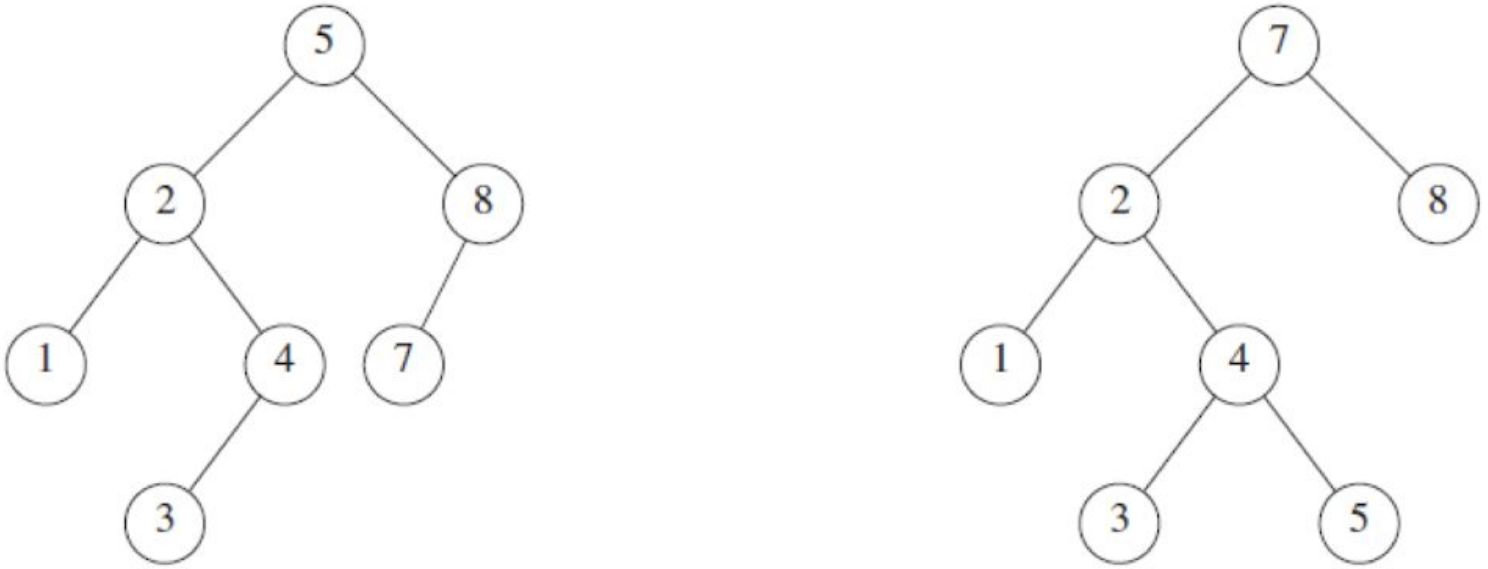
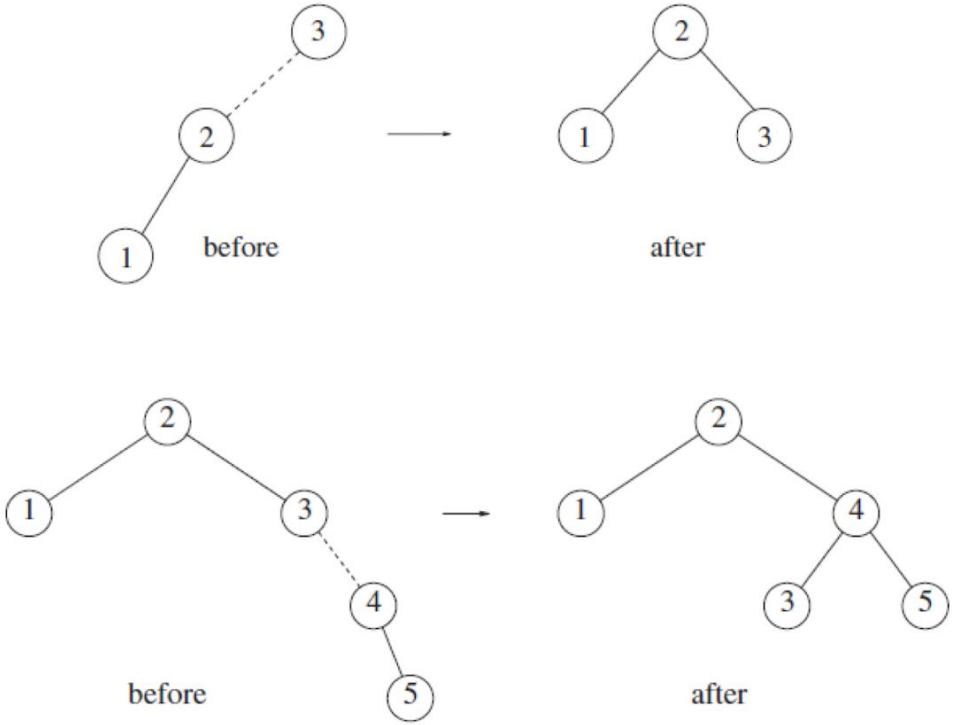


Figure 4.29 Two binary search trees. Only the left tree is AVL.

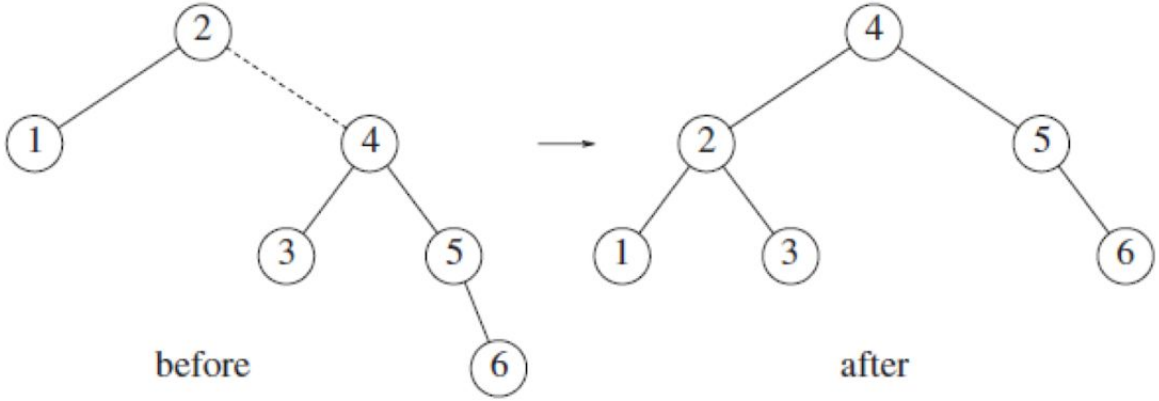
Q2

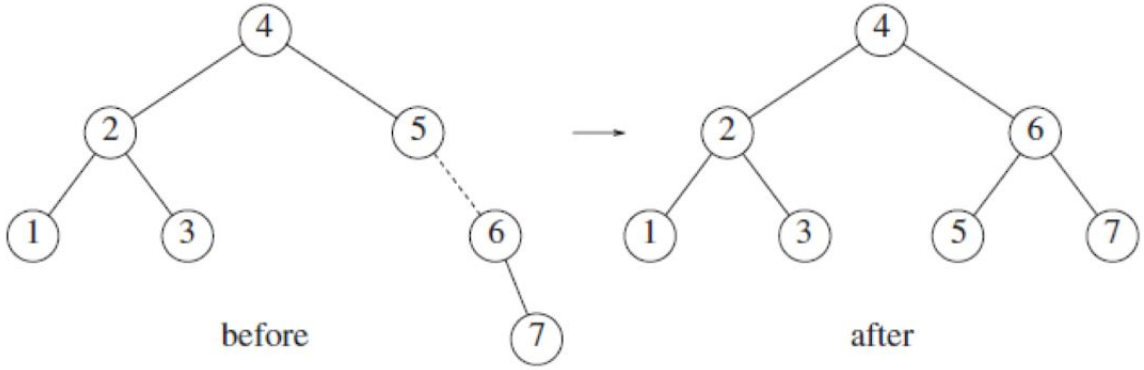
Suppose we insert 3,2,1 and 4 to 7 to an empty AVL tree.



[1]

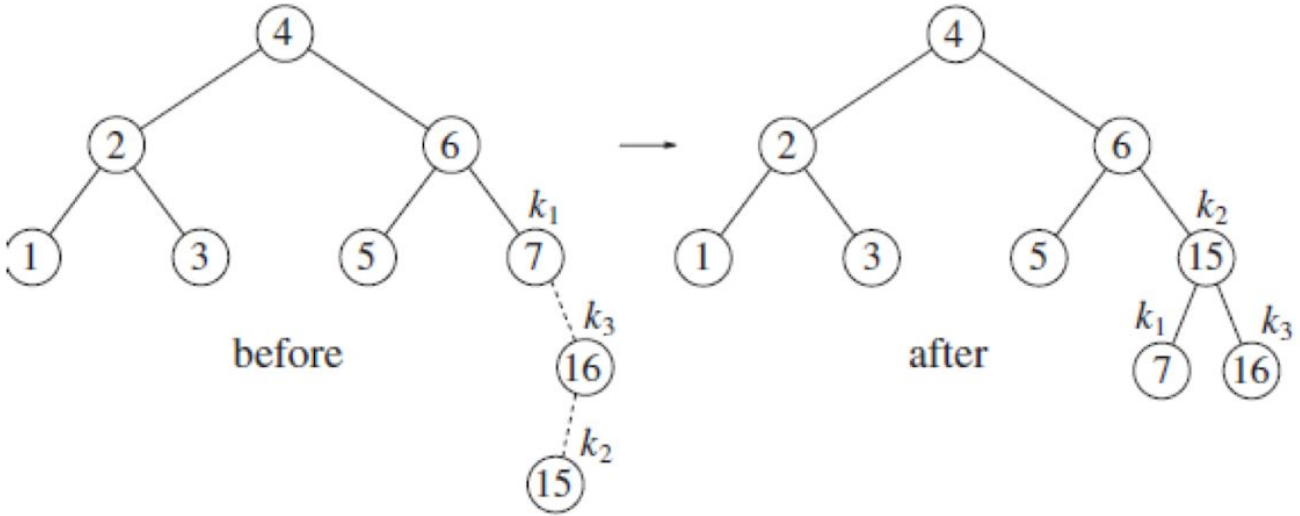
Continue: insert 6 and 7 to the AVL tree.





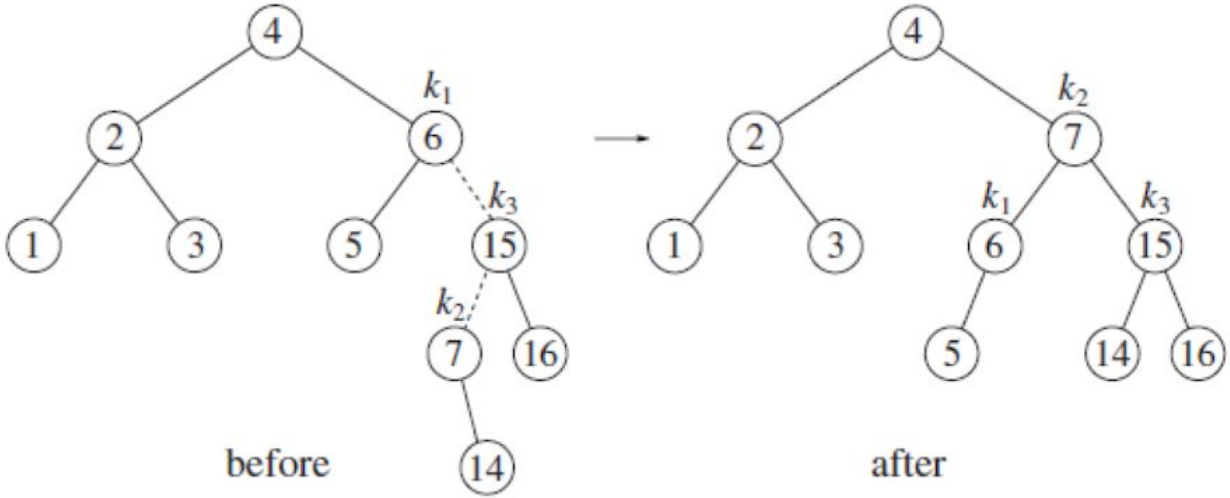
[1]

continue our previous example by inserting 14 through 16 in reverse order. Inserting 16 is easy, since it does not destroy the balance property, but inserting 15 causes a height imbalance at node 7. This is case 3, which is solved by a right–left double rotation.



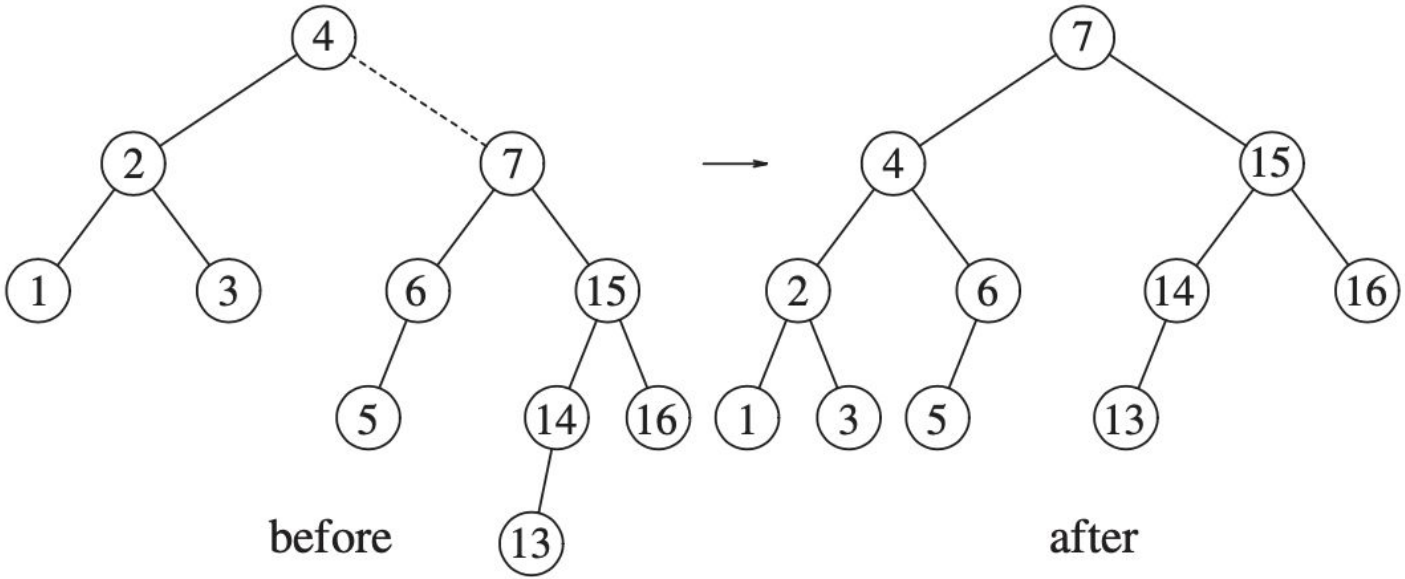
[1]

we insert 14, which also requires a double rotation. Here the double rotation that will restore the tree is again a right–left double rotation that will involve 6, 15, and 7.

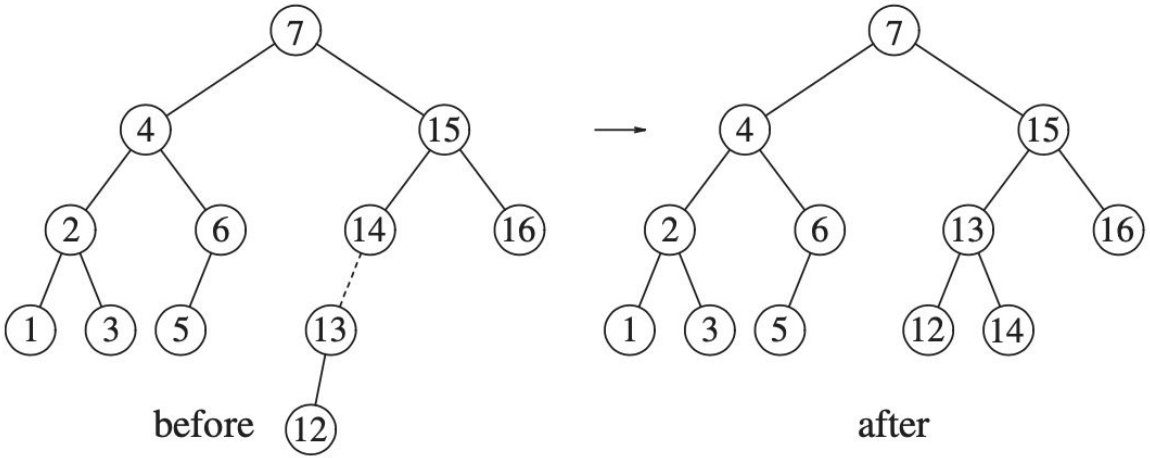


[1]

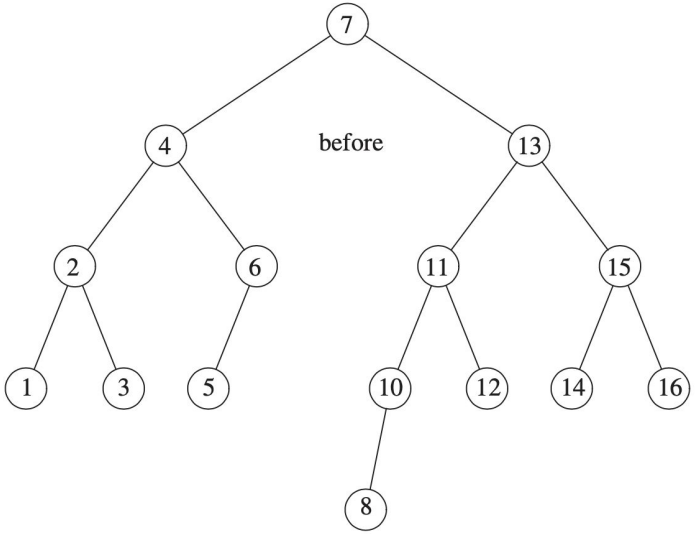
Insert 13



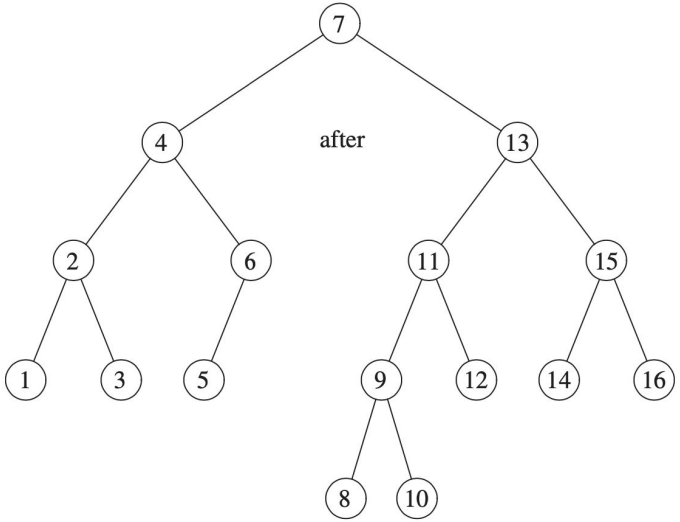
Insertion of 12 will also require a single rotation:



To insert 11, a single rotation needs to be performed, and the same is true for the subsequent insertion of 10. We insert 8 without a rotation creating an almost perfectly balanced tree:



before



after

Q3

Show that in a binary tree of N nodes, there are $N + 1$ null links representing children.

Q4

Show the result of inserting 2, 1, 4, 5, 9, 3, 6, 7 into an initially empty AVL tree.

[1]

Q5

Write efficient methods that take only a reference to the root of a binary tree, T, and compute:

a. The number of nodes in T.

b. The number of leaves in T.

c. The number of full nodes in T.

What is the running time of your routines?

CountNodes CountLeaves CountFull

```
static int countNodes( Node t )
{
  if (t == null)
     return 0;
  return 1  +  countNodes(t.left)  +  countNodes(t.right);
}

static int countLeaves( Node t )
{
  if (t == null)
     return 0;
  else if ( t.left == null && t.right == null)
     return 1;
  return countLeaves(t.left)  +  countLeaves(t.right);
}

static int countFull( Node t )
{
  if (t == null)
     return 0;
  int tIsFull  =  ( t.left != null && t.right != null) ? 1 : 0;
  return tIsFull  +  countFull(t.left)  +  countFull(t.right);
}
```

[1]

References

[1]Weiss - Data Structures and Algorithm Analysis in Java 3rd Edition