Exercise Session 08

Exercise 1.
(CLRS 12.3-1) Implement a recursive variant of the TREE-INSERT procedure.

Solution 1.
We present two alternative implementations. The first one is described in the following pseudocode

```
TREE-INSERT(T, z)
x = T.root
if x == NIL
    z.p = NIL
    T.root = z
else
    if z.key < x.key
        if x.left == NIL
            x.left = z
            z.p = x
        else
            TREE-INSERT(x.left, z)
    else
        if x.right == NIL
            x.right = z
            z.p = x
        else
            TREE-INSERT(x.right, z)
```

An alternative solution can make use of an auxiliary procedure which is called when T is not empty.

```
TREE-INSERT(T, z)
if T.root == NIL
    z.p = NIL
    T.root = z
else
    TREE-INSERT-AUX(NIL,T.root,z)
```

```
TREE-INSERT-AUX(y, x, z)
if x == NIL
    z.p = y
    if z.key < y.key
        y.left = z
    elseif z.key ≥ y.key
        y.right = z
elseif z.key < x.key
    TREE-INSERT-AUX(x, x.left, z)
else
    TREE-INSERT-AUX(x, x.right, z)
```

Exercise 2.
(CLRS 12.3-3) We can sort a sequence of n numbers by iteratively inserting each number in a binary search tree and then performing an inorder tree walk. Write the pseudocode of this algorithm. What are the worst-case and best-case running times for this sorting algorithm?

Solution 2.
The pseudocode of the suggested sorting procedure is

```
BST-Sort(A)
1  let T be an empty binary search tree
2  for i = 1 to n
3        Tree-Insert(T, A[i])
4  Inorder-Tree-Walk(T.root)
```

The worst-case running time is $\Theta(n^2)$ and occurs when A is already sorted (either in increasing or decreasing order). In this case the tree T in line 4 is a chain, i.e., a tree of height $n-1$. We have seen in class that this may occur e.g., when the array A is already sorted.
The best-case running time is $\Theta(n \log n)$ and occurs when the tree T in line 4 has height $\Theta(\log n)$. This may occur when the elements in A are uniformly distributed, i.e., they have no particular relative order one another in the array.

Exercise 3.
Consider the binary search tree T depicted in Figure 1. Delete the node with key = *10* from T by applying the procedure Tree-Delete(T, z) as described in CLRS pp. 298.

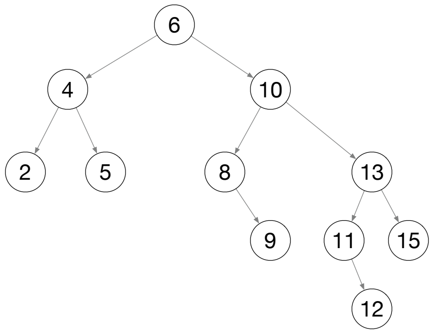


Figure 1: Binary Tree

Solution 3.

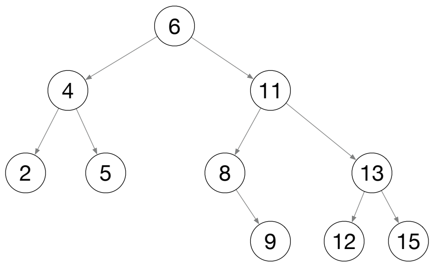
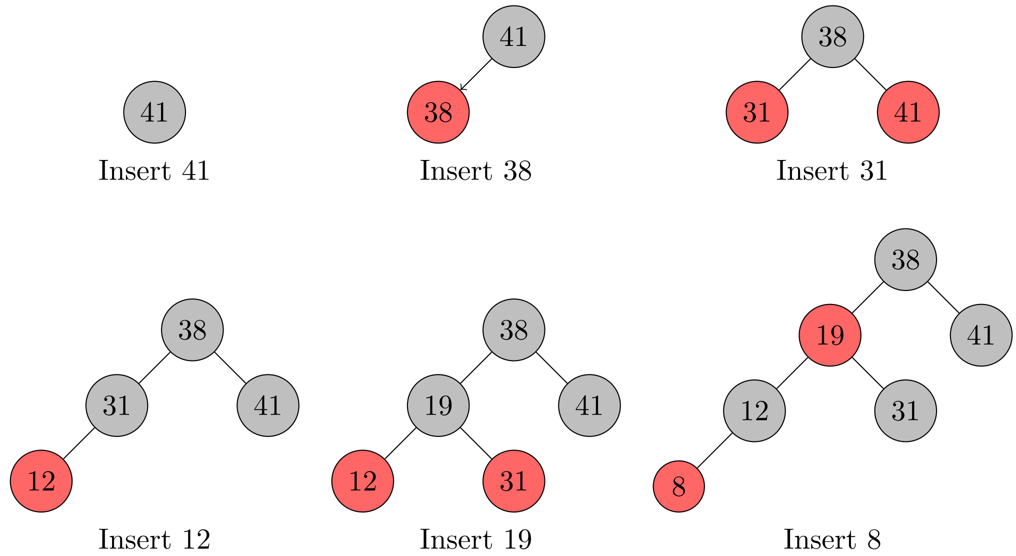


Figure 2: Binary Tree after deletion of node with $key = 10$

Exercise 4.
(CLRS 11.1-1) Show the red-black trees that result after successively inserting the keys 41; 38; 31; 12; 19; 8 into an initially empty red-black tree.

Solution 4.
Step-by-step insertions



Exercise 5.
Consider the red-black tree T depicted in Figure 3. Insert first a node with $key = 15$ in T, then delete the node with $key = 8$. Show all the intermediate transformations of the red-black tree with particular emphasis on the rotations.

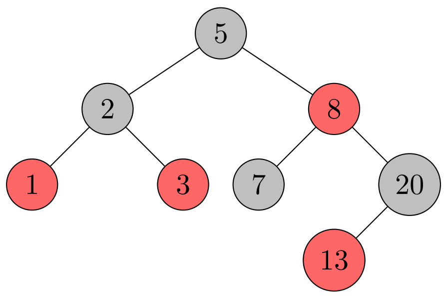
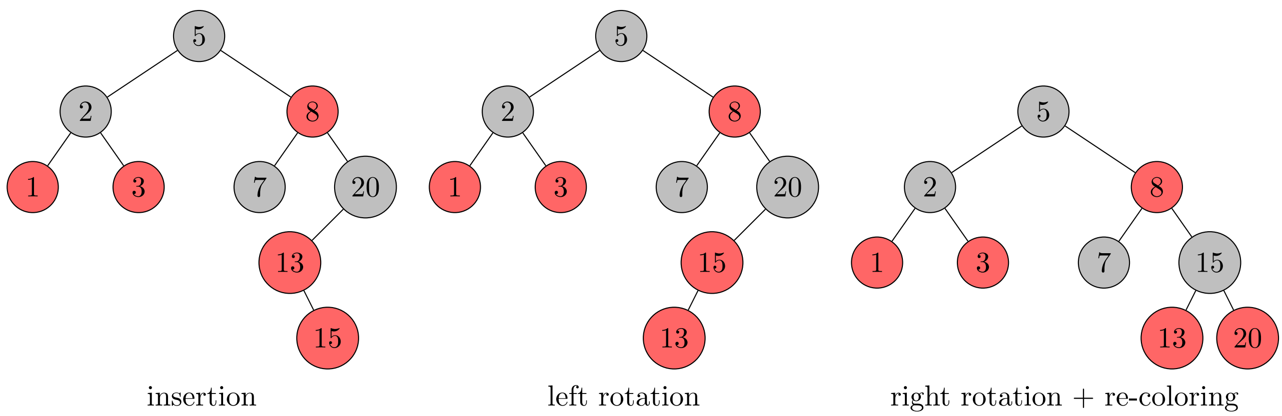
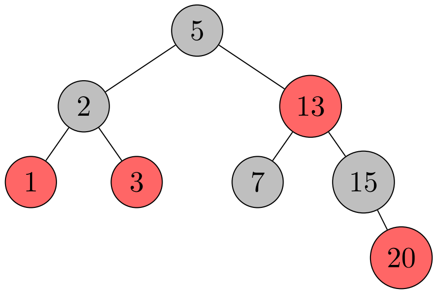


Figure 3: RB-Tree (NIL leaf nodes are omitted from the drawing)

Solution 5.
Step-by-step insertion of 15



The procedure for deleting 8 follows the same steps as for deletion for binary search trees. In this case, the transplant of the tree rooted at 8 with tree rooted at 13 (which is the successor of 8 in T) does not cause any violation of the red-black properties, because the node 13 is red. The result of the deletion is show below.



delete 8 & transplant