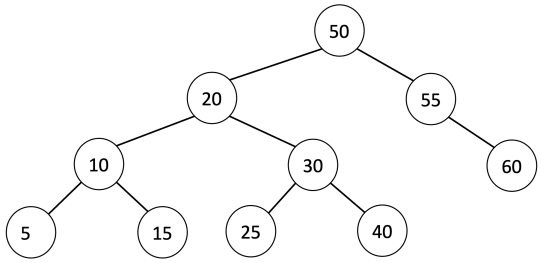
Washington State University
School of Electrical Engineering and Computer Science
Spring 2022

CptS 223 Advanced Data Structures in C++
Homework 5 - Solution
Due: February 16, 2022 (11:59pm pacific time)

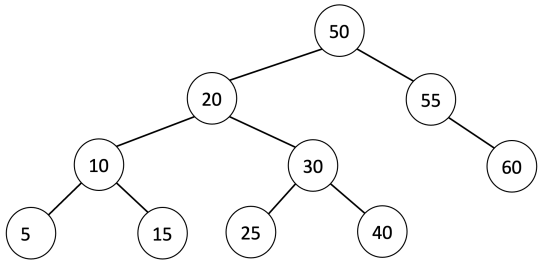
General Instructions: Input your answers in the Homework 5 Quiz on Canvas.

1. (3) Show the pre-order traversal for the following binary search tree.



Solution: 50, 20, 10, 5, 15, 30, 25, 40, 55, 60

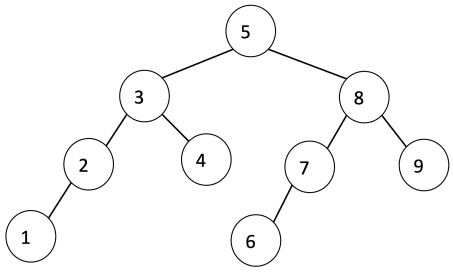
2. (3) Show the post-order traversal for the following binary search tree.



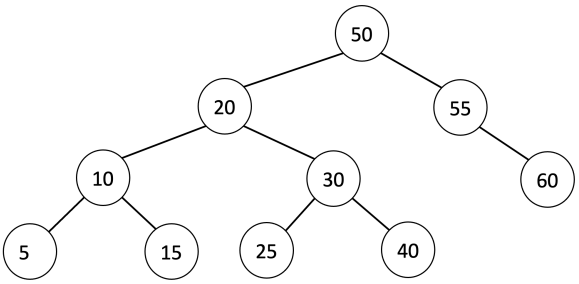
Solution: 5, 15, 10, 25, 40, 30, 20, 60, 55, 50

3. (5) Show the binary search tree whose post-order traversal is 1, 2, 4, 3, 6, 7, 9, 8, 5.

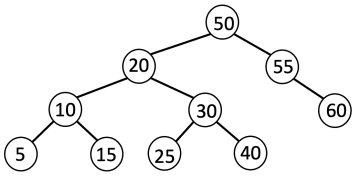
Solution:



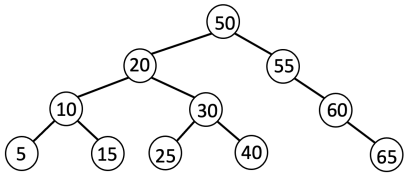
4. (8) Show the two AVL trees resulting from inserting keys 65 and then 45 in the following AVL tree. Indicate which of the four cases were used in each insertion.



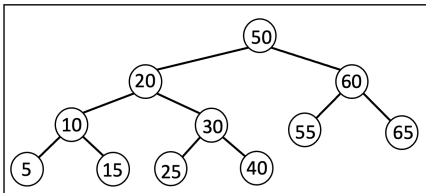
Solution:



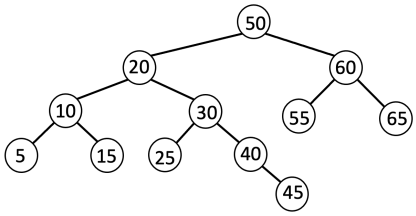
- Original tree



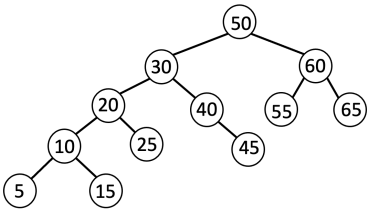
- Insert 65 in correct place
- Move up tree
- Violation at 55: h(L)=-1, h(R)=+1



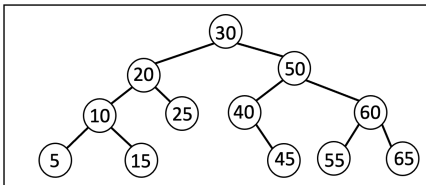
- Case 4
- Rotate left around 55,60
- Tree balanced



- Insert 45 in correct place
- Move up tree
- Violation at 50 h(L)=3, h(R)=1

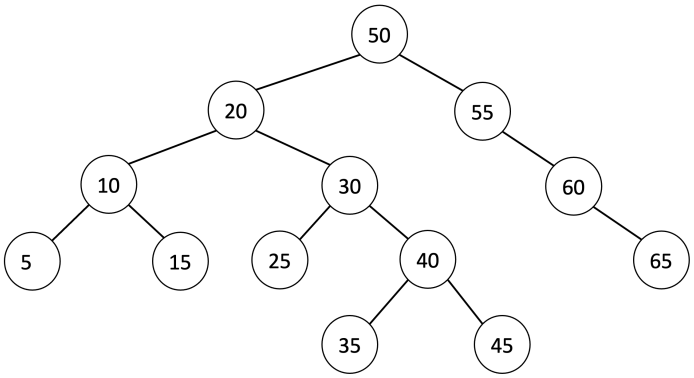


- Case 2
- Step 1: Rotate left around 20,30

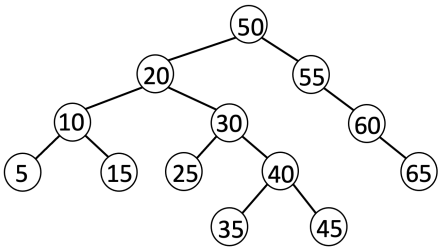


- Case 2
- Step 2: Rotate right around 30,50
- Tree balanced

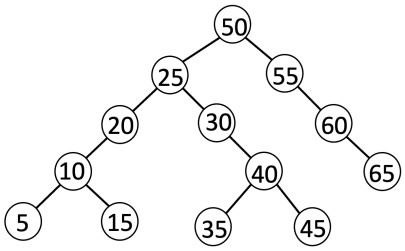
5. (10) Show the intermediate and final Splay trees resulting from removing key 25 from the following Splay tree. The intermediate trees are those resulting from zig-zag or zig-zig rotations. Indicate which scenario (zig-zag or zig-zig) occurred before each intermediate tree.



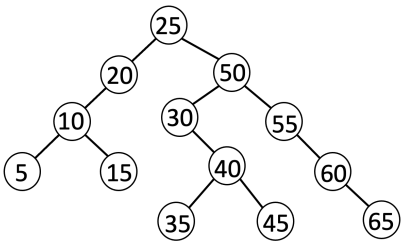
Solution:



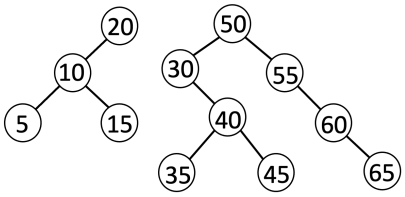
Original tree
Delete 25



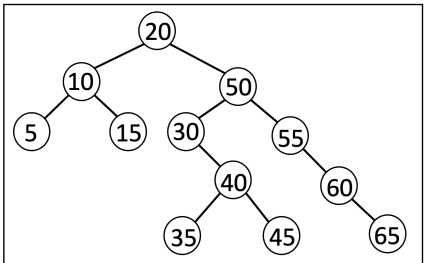
Bring 25 to root (zig-zag)
- Rotate right around 25,30
- Rotate left around 20,25



Bring 25 to root
- Rotate right around 25,50



Remove node 25
Access predecessor 20 (already at root)



Make right subtree right child of 20

6. (5) Consider the method DepthEqual(T) that outputs all node values in a binary search tree T such that the value stored in the node is equal to the depth of the node in the tree. The values can be any integer, including negative numbers. Show pseudocode for an efficient implementation of DepthEqual(T). *Hint*: For a node *n* at depth *d*, whose value is *v*, if $d \geq v$, then there is no need to search *n*'s left subtree.

   Solution:

```
DepthEqual(T)
     DepthEqual1(T, 0)

DepthEqual1(T, d)
     If T == null
     Then return
     If T->value == d
     Then print value
     If T->value > d
     Then DepthEqual1(T->left, d+1)
     DepthEqual1(T->right, d+1)
```

7. (3) Describe the best-case scenario for the running time of DepthEqual and give an asymptotically-tight Θ expression for the best-case running time in terms of N, the number of nodes in the tree.

   Solution: Best-case scenario is when all the nodes are in the left subtree of the root node, and the value at the root node is less than or equal to 0. In this case, the call to DepthEqual1 on the left subtree will not be executed, and the call to DepthEqual1 on the right subtree will immediately return, resulting in constant Θ(1) running time.

8. (3) Describe the worst-case scenario for the running time of DepthEqual and give an asymptotically-tight Θ expression for the worst-case running time in terms of N, the number of nodes in the tree.

   Solution: Worst-case scenario is when the value at a node is always greater than the depth of the node, in which case all nodes must be visited, resulting in Θ(N) running time.