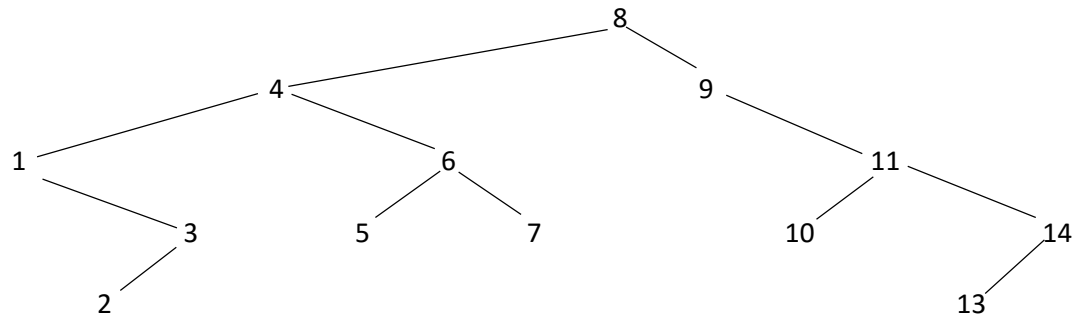
Workshop 3 – Week 4 – Worksheet 4

Question 4.1 Given the following input:

8 4 9 11 6 7 1 5 3 14 10 13 2

Write the insertion function used to get the binary search tree shown below:



There were a few ways you could do this one, this isn't necessarily the best one, however it should be enough for you to work out whether your solution was right or not.

```
insert (address of subtree, item to insert){
        if (value at address of subtree is NULL){
                insert at address of subtree and return
        }
        if(item < subtree value){
                insert item in left subtree
        } else {
                insert item in right subtree
        }
}
```

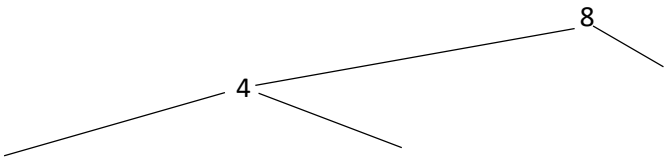
Draw a new tree after inserting each number and show the pointer reassignments and comparisons necessary to implement each step of these insertions.

1. Insert 8

Root->value = 8

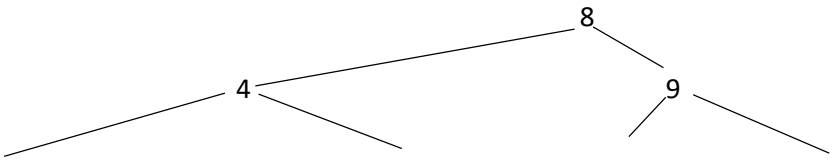
Root->left = NULL
Root->right = NULL
(We'll now refer to "Root" as 8)

2. Insert 4



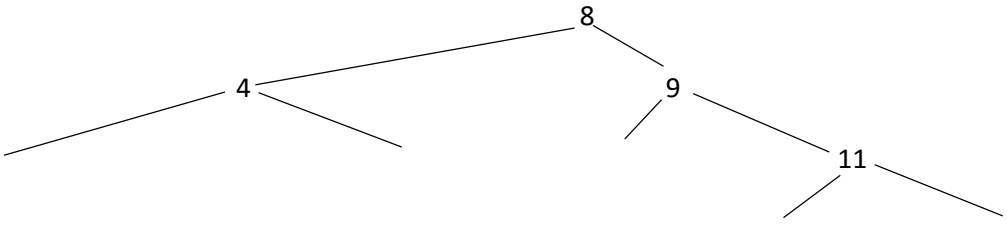
8->left = 4
4->left = NULL
4->right = NULL

3. Insert 9



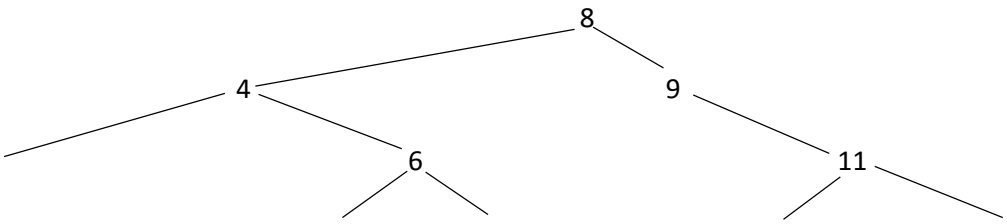
8->right = 9
9->left = NULL
9->right = NULL

4. Insert 11



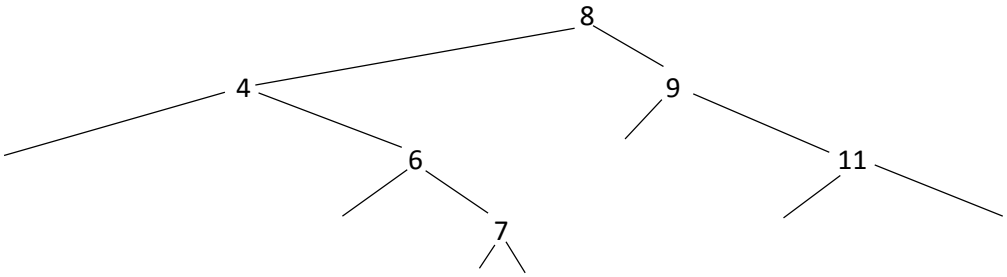
9->right = 11
11->left = NULL
11->right = NULL

5. Insert 6



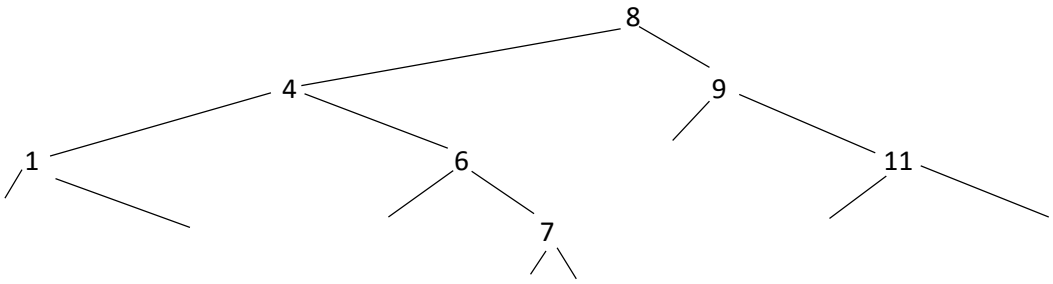
4->right = 6
6->left = NULL
6->right = NULL

6. Insert 7



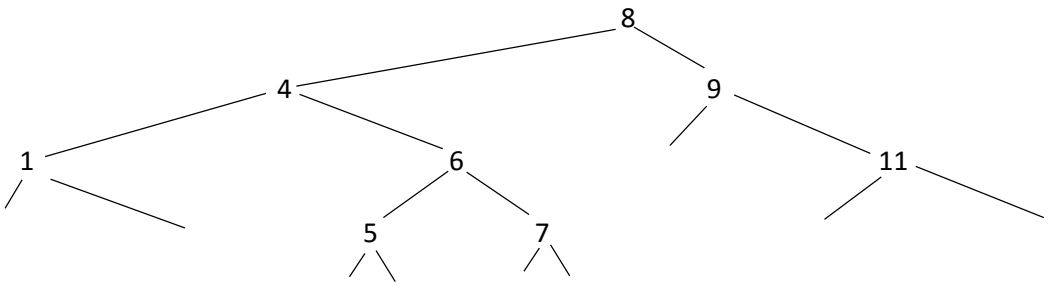
6->right = 7
7->left = NULL
7->right = NULL

7. Insert 1



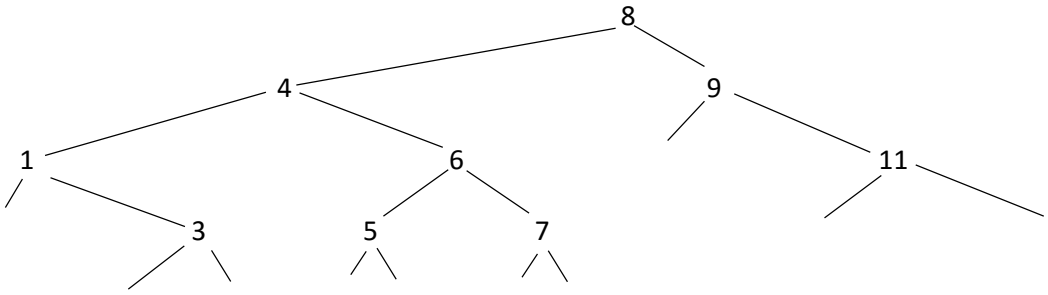
4->left = 1
1->left = NULL
1->right = NULL

8. Insert 5



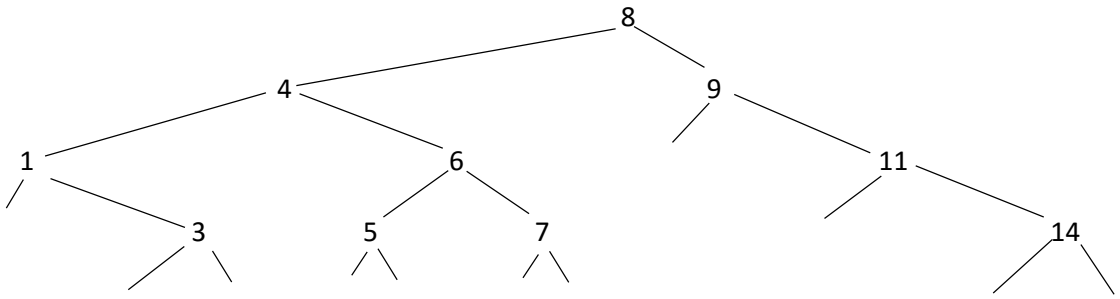
6->left = 5
5->left = NULL
5->right = NULL

9. Insert 3



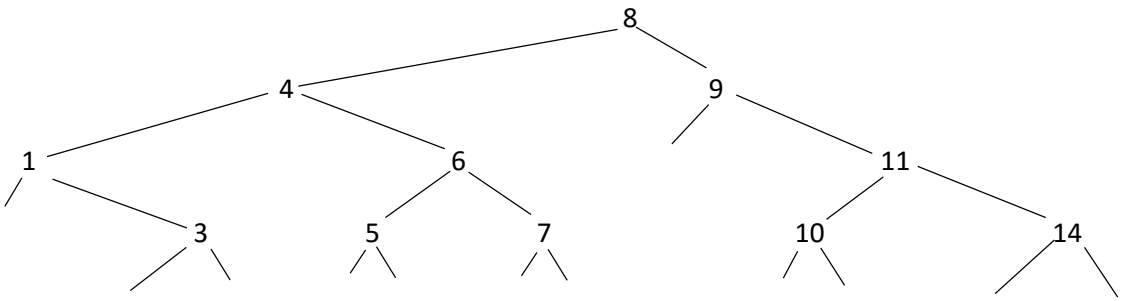
1->right = 3
3->left = NULL
3->right = NULL

10. Insert 14



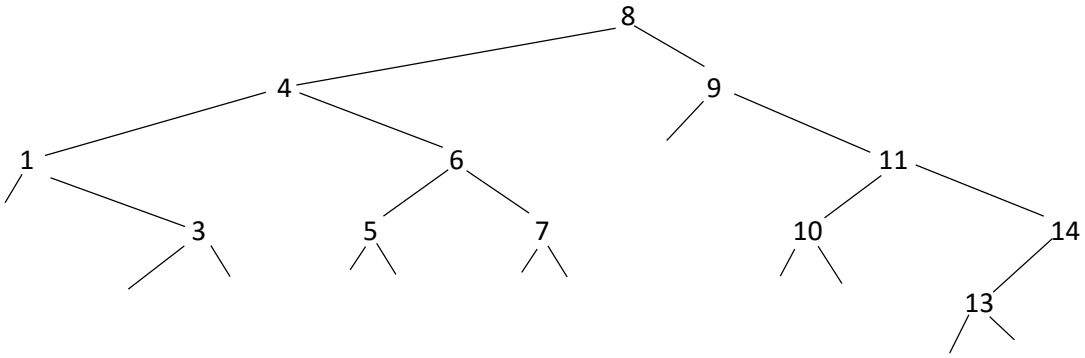
11->right = 14
14->left = NULL
14->right = NULL

11. Insert 10



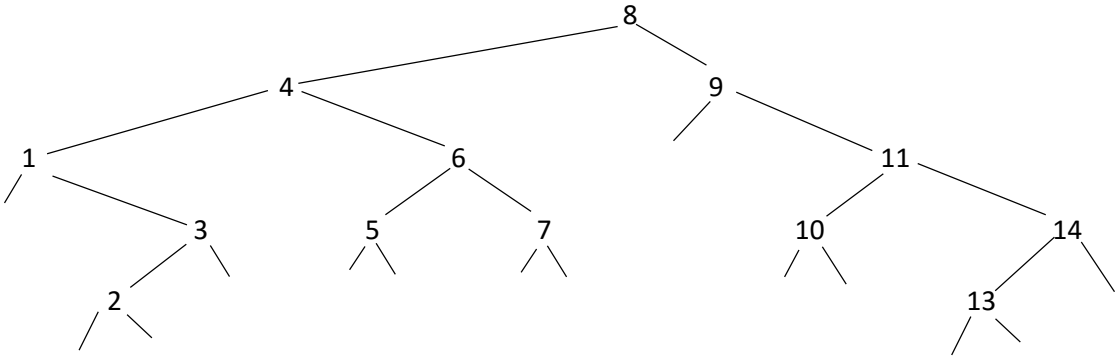
11->left = 10
10->left = NULL
10->right = NULL

12. Insert 13



14->left = 13
13->left = NULL
13->right = NULL

13. Insert 2



3->left = 2
2->left = NULL
2->right = NULL

Question 4.2 In some circumstances, it might be useful to have a doubly-linked tree, i.e. each node has a parent pointer (root pointer is initialized to null), as well as two child pointers. Assume each node has a counter and a depth variable. After each insertion some counters get updated using the following assignment:
node.counter = node.left.depth − node.right.depth;

Show how can you use the doubly-linked tree to keep the counters updated after a node is inserted.

This question is primarily hinting at AVL trees. When a node is inserted into the tree, it is inserted where it is expected to be inserted with some defined starting depth (such as 1 or 0), the algorithm then travels up the tree, updating each node in the tree with a depth of one more than the highest depth of its two children, if a depth ever need not be updated, the movement up the tree halts. At each step, the counter can be updated, in an AVL tree, any point where the counter reaches a value of 2 will cause a rotation to occur, however in this case the question hasn't asked about that. Importantly, empty children (however they are represented) are given a depth value of one less than the defined starting depth.