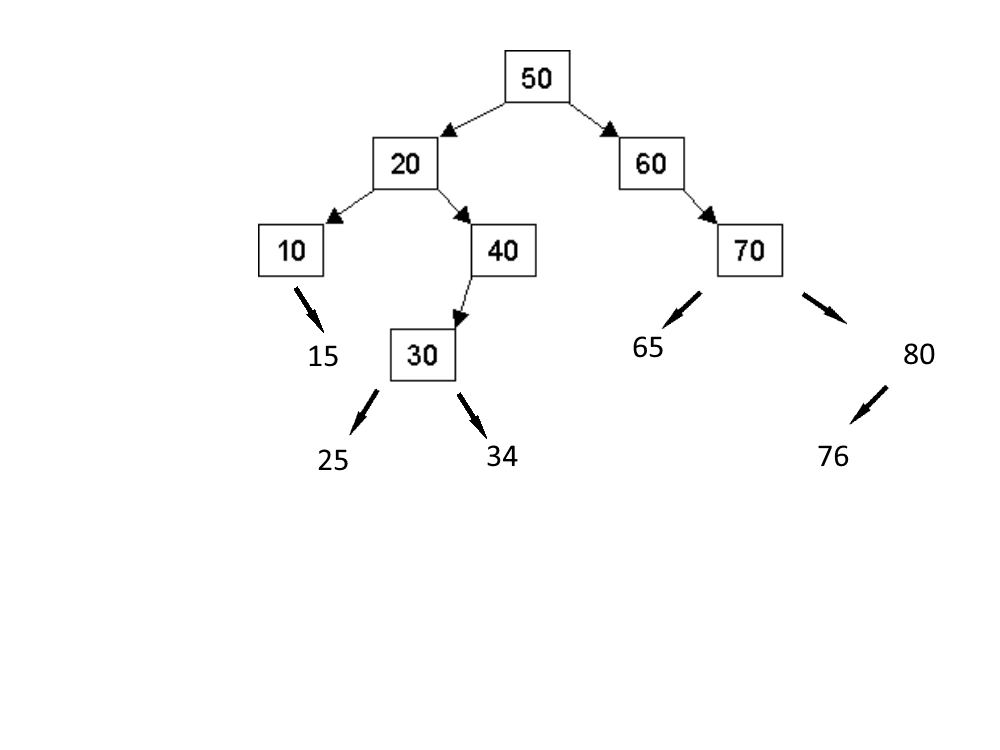
Question 1a.



Question 1b.

In order:

10, 15, 20, 25, 30, 34, 40, 60, 65, 70, 76, 80

Pre-Order:

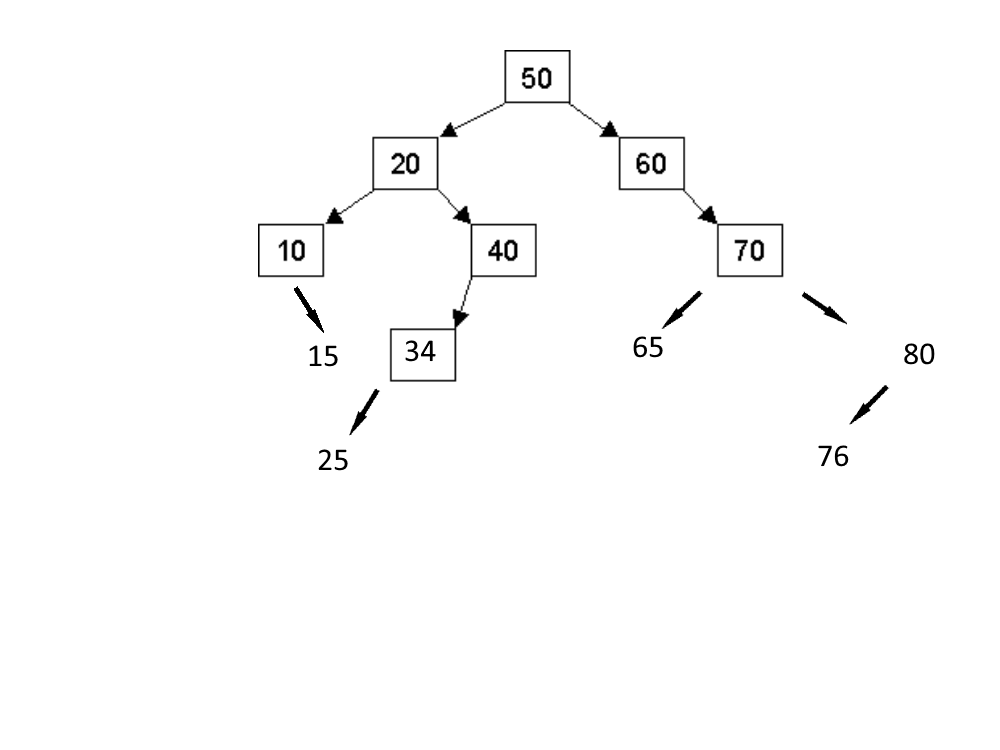
50, 20, 10, 15, 40, 30, 25, 34, 60, 70, 65, 80, 76

Post Order:

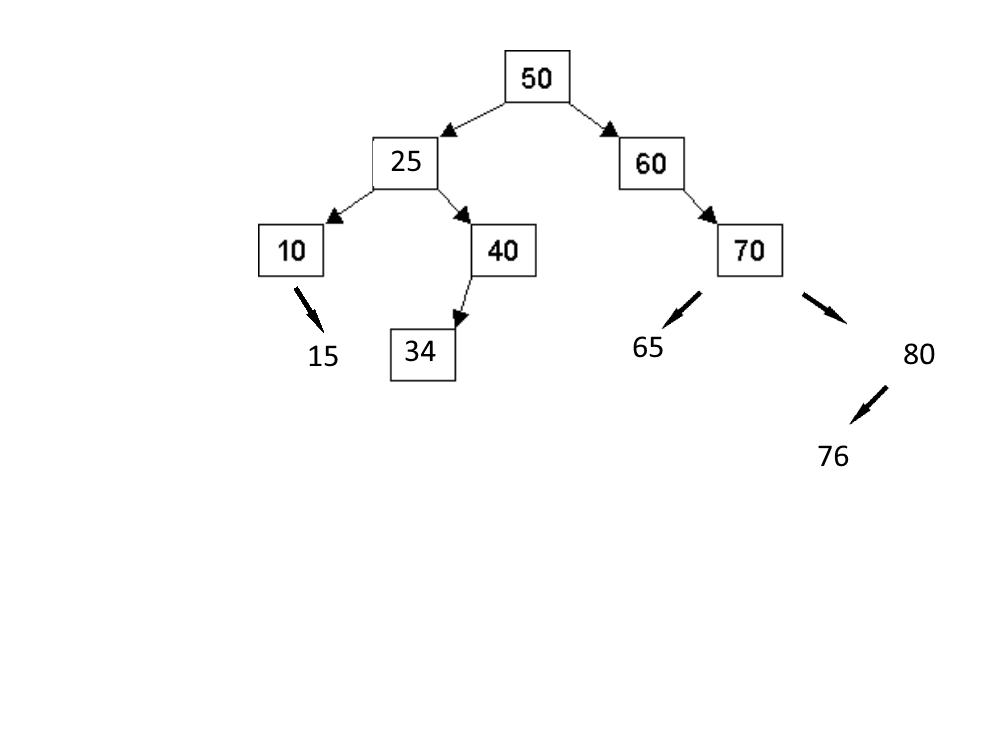
15, 10, 25, 34, 30, 40, 20, 65, 76, 80, 70, 60, 50

Question 1c.

Delete 30



Delete 20



Question 2a.

struct Node

{

int m\_value;

Node \*m\_left, \*m\_right, \*m\_parent;

};

Question 2b.

If the tree is empty, then return the new Node with nullptrs for pointers and with the key value, then set root pointer to it.

If the key is less than the current node’s key, then

Node\* with recursion to left of the node

Set the left pointer of the current node to point towards the above Node\*

Set the parent of root of the left subtree

Else if the key is greater than the current node’s key, then

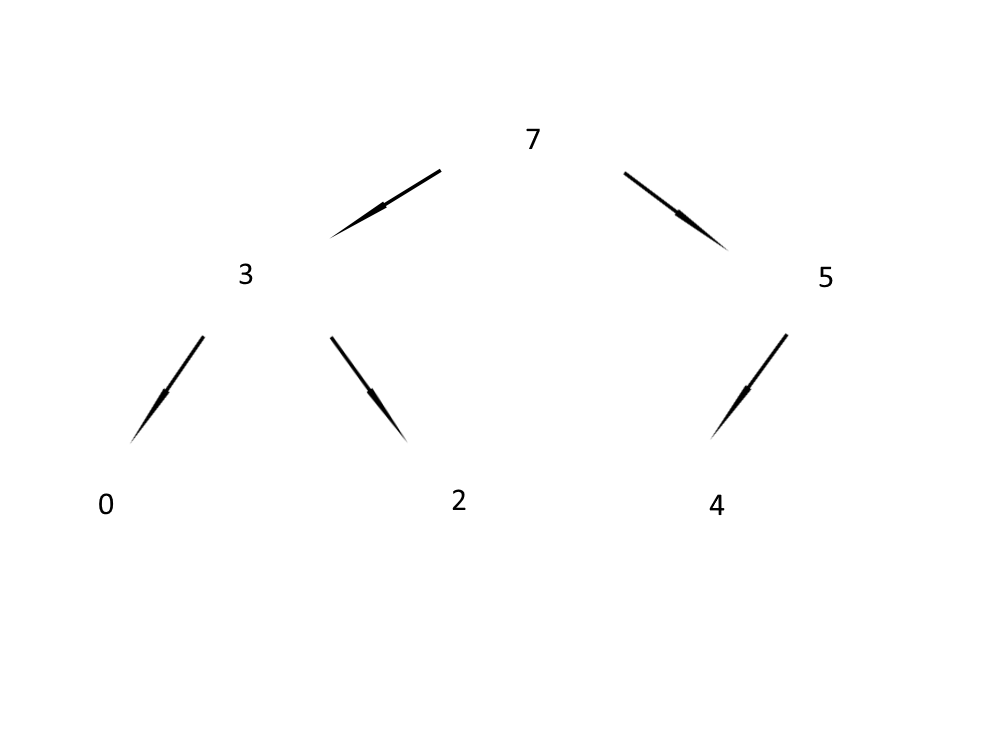
Node\* with recursion to right of the node

Set the right pointer of the current node to point towards the above Node\*

Set the parent of root of the right subtree

Return node

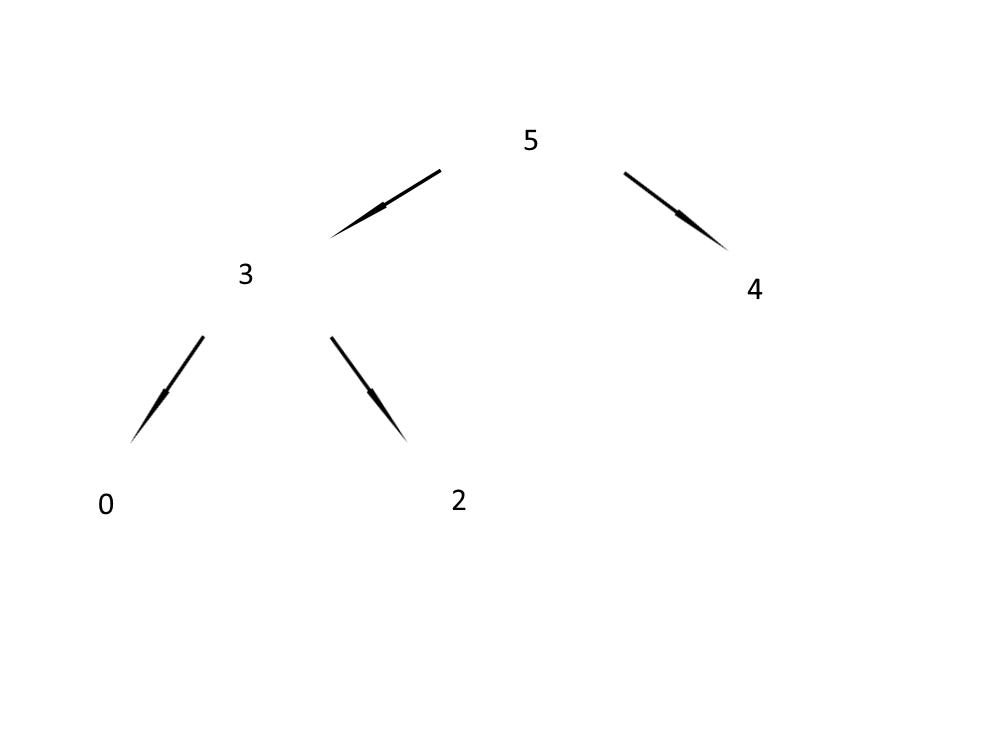
Question 3a.



Question 3b.

arr[] = { 7, 3, 5, 0, 2, 4 };

Question 3c.



Question 4a.

O(C+S)

Question 4b.

O(logC+S)

Question 4c.

O(logC+logS)

Question 4d.

O(logS)

Question 4e.

O(1)

Question 4f.

O(logC+S)

Question 4g.

O(SlogS)

Question 4h.

O(ClogS)

Question 5b.

We can’t solve this function without the two-parameter overload if we did it recursively since we would not be able to tell where it came from, or its path. Since it only knows itself and its subclasses, we would not be able to track its parents and thus not be able to print out the entire statement with only recursion. The string path lets us bypass not knowing the parents by having the path to be printed stored as a string beforehand, thus allowing us to print the statements out with recursion.