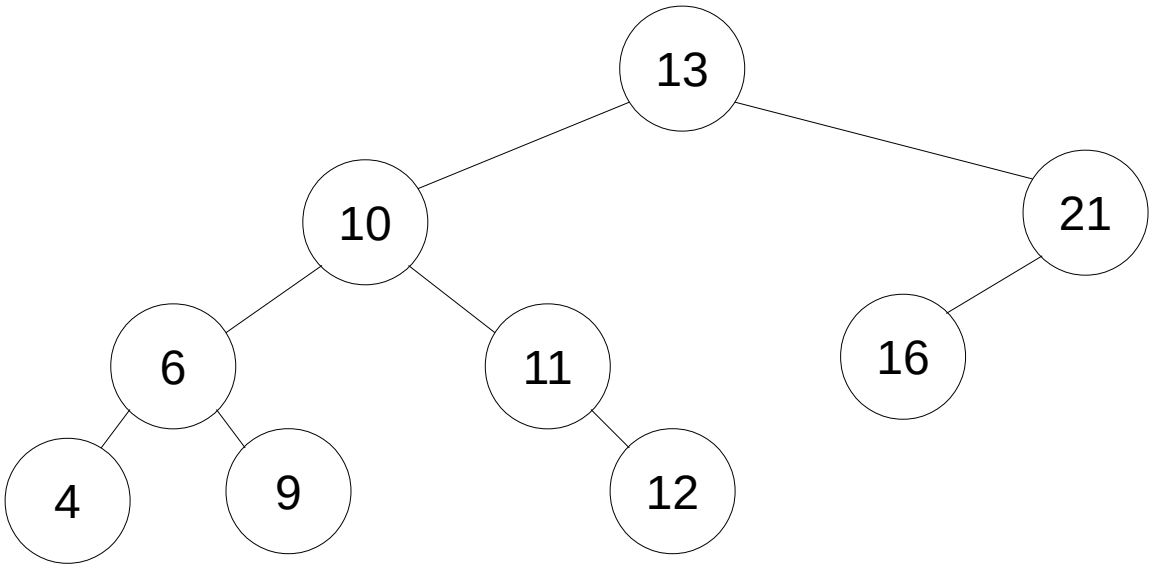
/* *
* Title : Heaps and AVL Trees
* Author : Munib Emre Sevilgen
* ID : 21602416
* Section : 1
* Assignment : 3
* Description : Part a, b, and c of the Question 1
*/

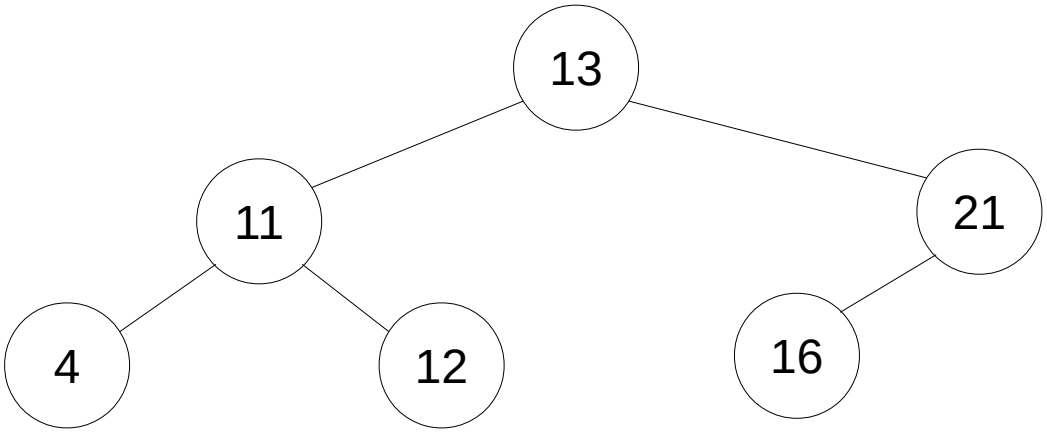
Question 1:

(a)

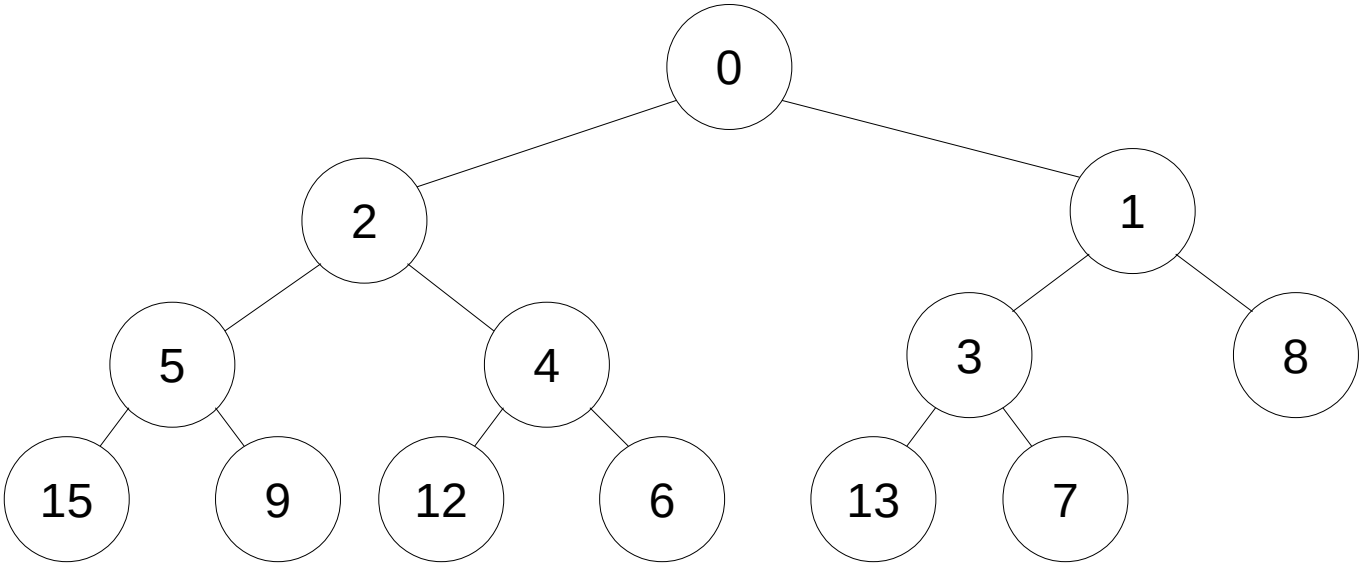
After insertions:



After deletions:



(b)



(c)

```
x = delete the rightmost node of T1 - O(logn)
k = height of the right subtree of T1
l = height of the left subtree of T2
if (k>l) // Merge T2 with proper subtree of T1
        parentNode = T1
        currentNode = T1's right
        // Find proper subtree
        while (height of the currentNode > height of T2) - O(logn)
                parent = currentNode
                currentNode = currentNode→right
        // Merge T2 and currentNode using x that is delete value
        newNode = x // Create a new node with the value of x
        newNode→left = currentNode
        newNode→right = T2
        parent→right = newNode
        // Fix the possible imbalance at currentNode and parent using corresponding
        // rotations
        fix(currentNode)
        fix(parent)

else if (k<l) // Merge T1 with proper subtree of T2
        parentNode = T2
        currentNode = T2's left
        // Find proper subtree
        while (height of the currentNode > height of T1) - O(logn)
                parent = currentNode
                currentNode = currentNode→left
        // Merge T1 and currentNode using x that is delete value
        newNode = x // Create a new node with the value of x
        newNode→right = currentNode
```

```
        newNode→left = T2
        parent→left = newNode
        // Fix the possible imbalance at currentNode and parent using corresponding
        // rotations
        fix(currentNode)
        fix(parent)

else
        if (height of T1 > height of T2) // Merge T2 with proper subtree of T1
                parentNode = T1
                currentNode = T1's right
                // Find proper subtree
                while (height of the currentNode > height of T2) - O(logn)
                        parent = currentNode
                        currentNode = currentNode→right
                // Merge T2 and currentNode using x that is delete value
                newNode = x // Create a new node with the value of x
                newNode→left = currentNode
                newNode→right = T2
                parent→right = newNode
                // Fix the possible imbalance at currentNode and parent using
                // corresponding rotations
                fix(currentNode)
                fix(parent)

        else // Merge T1 with proper subtree of T2
                parentNode = T2
                currentNode = T2's left
                // Find proper subtree
                while (height of the currentNode > height of T1) - O(logn)
                        parent = currentNode
                        currentNode = currentNode→left
                // Merge T1 and currentNode using x that is delete value
                newNode = x // Create a new node with the value of x
                newNode→right = currentNode
                newNode→left = T2
                parent→left = newNode
                // Fix the possible imbalance at currentNode and parent using
                // corresponding rotations
                fix(currentNode)
                fix(parent)
```

Total time is O(logn) since all the operations except the specified ones are O(1) and the total time is the sum of the O(logn) operations. Therefore the the complexity of the algorithm is O(logn).