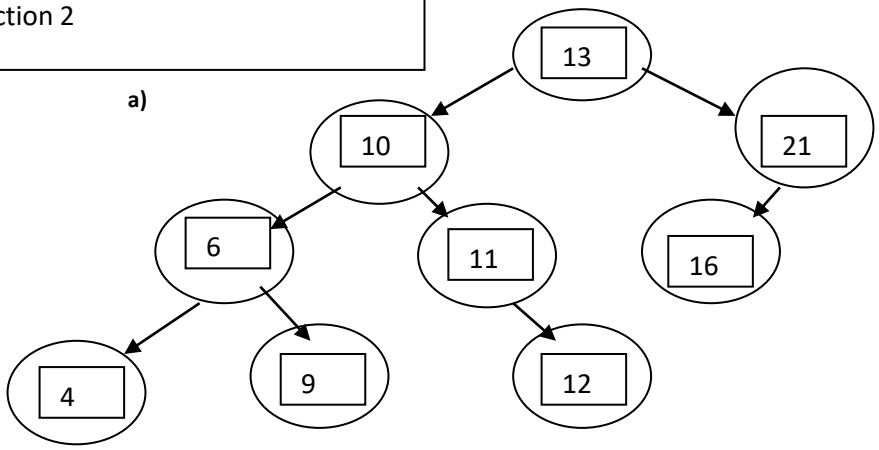
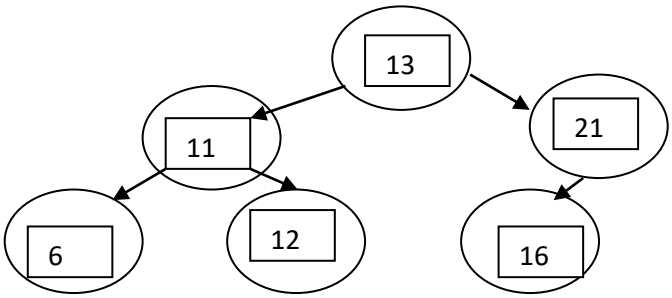
HW3 CS202

Mehmet Bora Kurucu 21703404

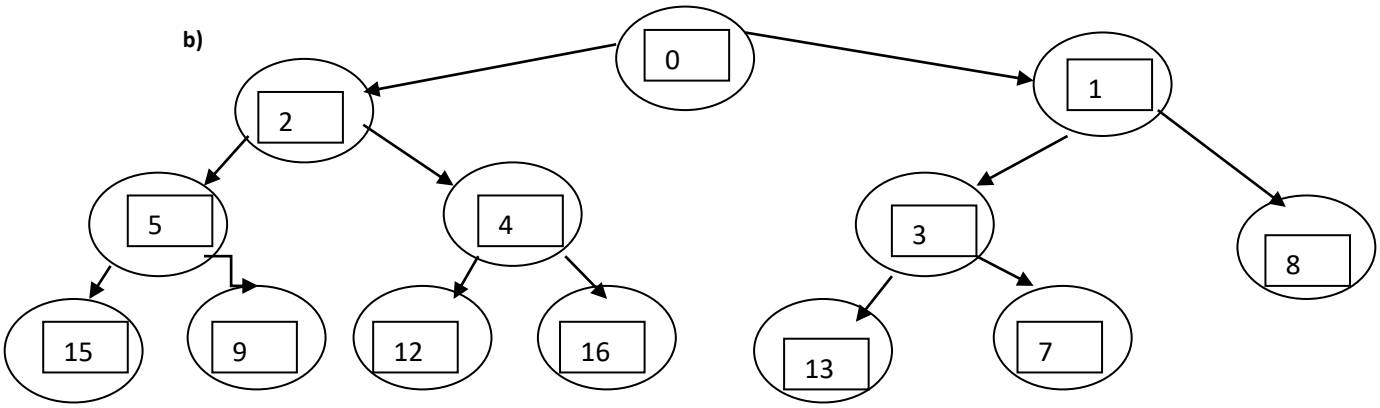
Section 2

a)





b)



c)

```
JOIN-AVL-TREES( T1, T2)
{
İnt H1 = calcHeight(T1)
İnt H2 = calcHeight(T2)
İf(H2 > H1)
{
While(calcHeight((T2->left) – H1 > 1)
{
        T2Parent = T2
        T2 = T2->left
}
FindRigt(T1,RParent,TRight)
TRight->right = T2
T2Parent->left = TRight
TRight->left = RParent
}
```

```
Else
{
While(calcHeight((T1->right) – H2 > 1)
{
        T1Parent = T1
        T1 = T1->right
}
FindLeft(T2,LParent,TLeft)
T1Parent->right = T2
TLeft->left = T1
T1Parent->right = TLeft
TLeft->right = LParent

}
}
```

The idea is to put the tree with less height into the tree with greater hight. Than in greater heighted tree, find the appropriate place depending on the height of the smaller tree,put the smaller tree there, than do the required rotation. Find the height of both of the trees with calcheight,completes in O(log(T1.size + T2.size) time. İf T2'size is bigger,go to left, until a node's size of subtree is not greater than T1 at most(traversal to only one side in a balanced tree,O(log(T2.size)). This is the proper place to insert, because inserting T1 here won't harm the balance here, because the other subtree is balanced with that height too. The important thing is to do the necessary rotations O(1) here in order to keep the balance with the other subtree of the node. The algorithm is very similar when T1's size is bigger, just use the opposite ways and rotations. Here,since T1 + T2 = n, since T1 and T2 are variants of n, the complexity is **not** O(log(T1)) or any variant of that, it is O(logn),especially when you consider the other methods like calcheight and traversal that I explained earlier.