CSI2110-D – Written Assignment #3

Alae Boufarrachene (300188539)

Question 1 :

a) Considering that we're dealing with a full binary tree, the number of nodes is guaranteed to be odd because all nodes have 2 children and one parent (except for the root obviously), which corresponds to the last four digits of my student number (8559), no need to add 1.
We know that for a full binary tree, the total number of nodes should be equal to to twice the number of internal nodes minus one.

Let K be the number of leaves :

$n = 2K - 1$ (we know $n = sn = 8559$)

$2K = n + 1$

$2K = 8559 + 1$

$2K = 8540$

$K = \frac{8540}{2} = \boxed{4270}$

Logically, the number of internal nodes should be equal to the number of leaves minus one (again, we're dealing with a full tree)

Let d be the number of internal nodes :

$d = K - 1$

$d = 4270 - 1 = \boxed{4269}$

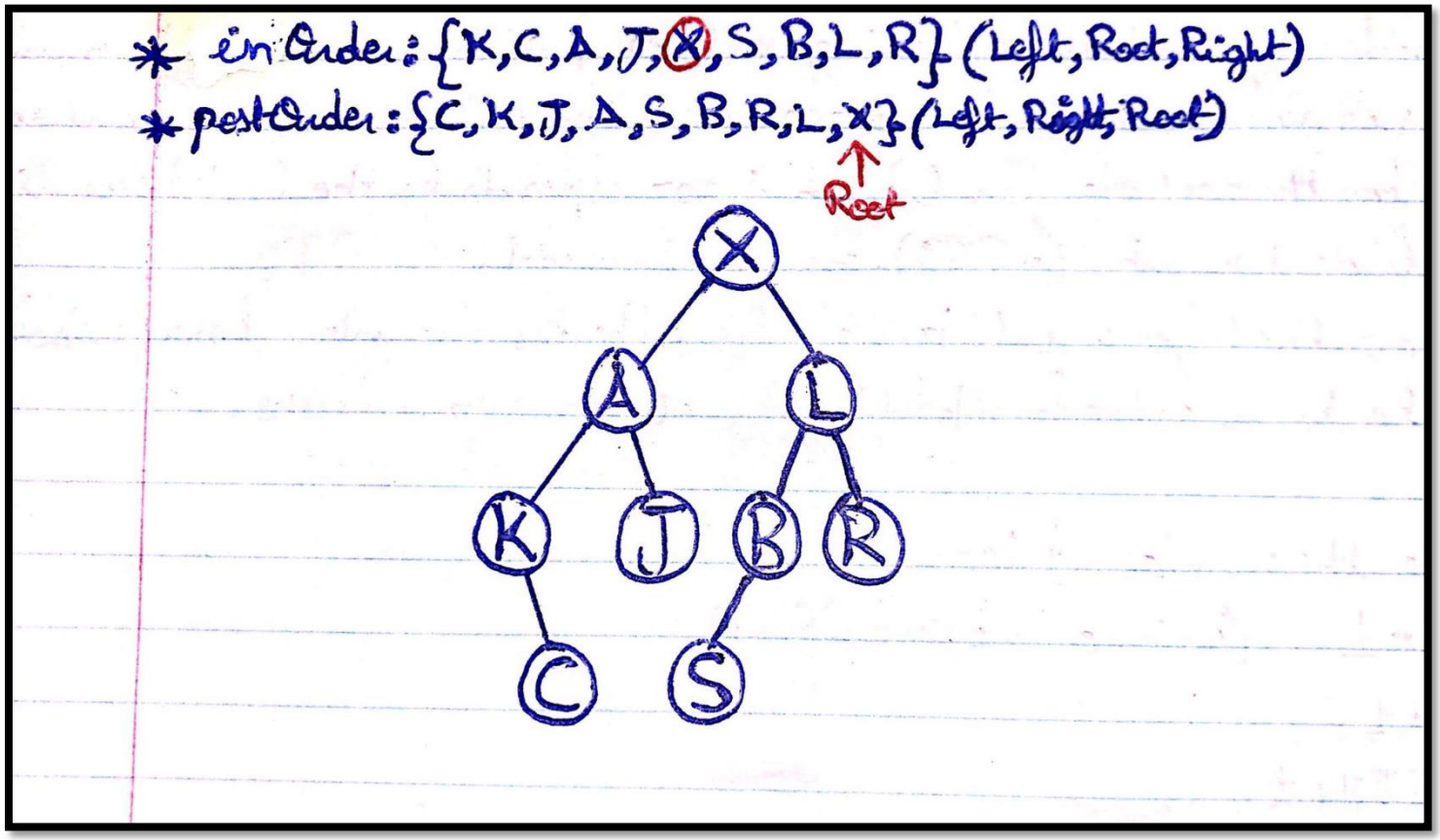
b) We know that : $K \geq 2$; $n = 3 \times 2^K$

→ $h(K) \geq \log_2(n+1) - 1$; $n = 3 \times 2K$

→ $h(K) \geq \log_2(3(2^K) + 1) - 1$

Alae Boufarrachene (300188539)

Question 2 :



Question 3 :

```
deepestAncestor(node1,node2):

    tempParent1 = node1.getParent()
    tempParent2 = node2.getParent()

    while (tempParent1.getParent()!=null & tempParent2.getParent()!=null):

        if (tempParent1.getParent()!=null & tempParent2.getParent()!=null):
            node1Ancestors.add(tempParent1)
            node2Ancestors.add(tempParent2)
            tempParent1 = tempParent1.getParent()
            tempParent2 = tempParent2.getParent()

        if (tempParent1.getParent()==null & tempParent2.getParent()!=null):
            node2Ancestors.add(tempParent2)
            tempParent2 = tempParent2.getParent()

        if (tempParent1.getParent()!=null & tempParent2.getParent()==null):
            node1Ancestors.add(tempParent1)
            tempParent1 = tempParent1.getParent()

    commonAncestors = combineCommonElementsOfTwoLists(node1Ancestors,node2Ancestors)
    lowestCommonAncestor = commonAncestors.get(0)

    return lowestCommonAncestor
```

My running time is O(d) because in the worst case scenario we would be comparing two nodes of depth equal to the height of the tree itself and the while loop is set-up in a way that would run until both node pointers have reached the root, which shouldn't exceed the depth of the nodes (or the depth of the deepest node in a case in which we run the method on two nodes of different generations/layers).