**Q3:**

It is impossible to perform a BST in O(Root log n) as the recursive function works by performing the loop in log n time and the lower bound of the BST is to O(Log n) performing the insertion with the O(root log n) would violate the lower bound of the BST and generate an different algorithm.

**Q4 a):**

We will place all the ducks in the order of a binary Search tree and then preorder traverse the tree checking each ducks height as we go comparing it to the height of the tree. There is no need for the ducks to be sorted we will just pick a random duck as root and place the ducks taller than it on the right side and the ones smaller on the left side.

Pseudo Code:

Def insert\_duck(root,data):

If root==null:

Root=node(data)

Return root

elif data<root.data:

root.left=self.insert(root.left,data)

elif data>root.data:

root.right=self.insert(root.right,data)

This method is to insert the ducks in the tree.

def preOrder(self,root):

if root==None:

return

if root.height==scale\_height:

return “same”

else:

Return “no duck with same height”

print(root.data)

self.preOrder(root.left)

self.preOrder(root.right)

This way we will move along the tree in Preorder and compare height.

**b):**

any algorithm in this model of computation must use Ω(log(n)) comparisons because with each step we gain information about the possible location of the element as when checking for a certain duck we will compare it with the root and immediately know whether it will be on the right side of the tree or left and also the lower bound of the tree is Log n and the upper boundary is also log n so it will always give result in log n time.

**Q5:**

We know that every comparison based algorithm uses O(nlogn) time. In this case if we consider the following:

1. **Insert**: We can insert each element into the gooseTree using gooseInsert(k). This operation is claimed to run in O(1) time.
2. **In-order traversal**: To retrieve the elements in sorted order, we can perform an in-order traversal of the gooseTree. Since gooseSearch(k) is claimed to run in O(log(n)) time, the in-order traversal would take O(n log(n)) time.
3. **Retrieve and delete**: For each element found during the traversal, we can use gooseDelete(k) to remove and return the element. This operation is claimed to run in O(log(n)) time. This too will take O(nlog n)

Hence the total time complexity of the goosetree will be O(nlogn) not O(log n) hence the goose is wrong.