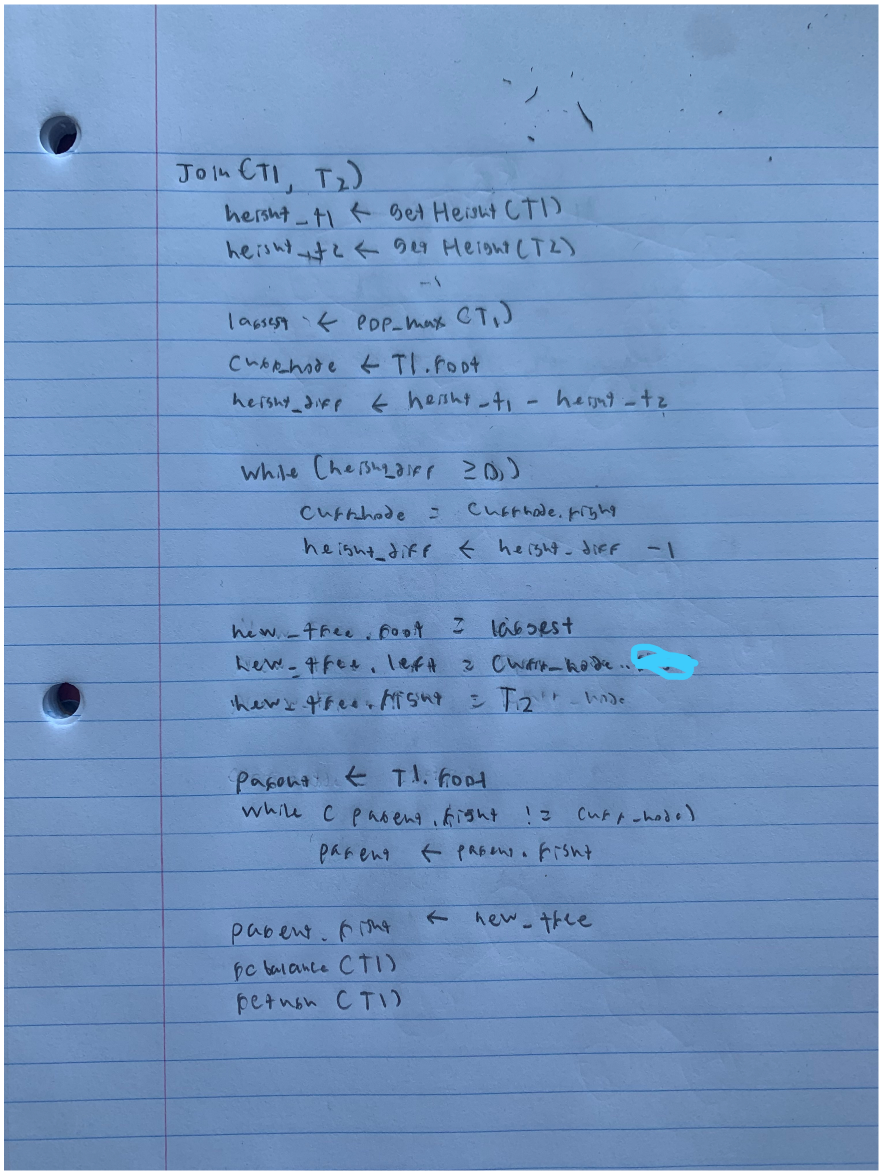
CS240, Spring 2022
Assignment 3: Question 4

Q4) The rational and explanation is on the next page



```
Join(T1, T2)
    height_t1 ← get Height(T1)
    height_t2 ← get Height(T2)
                      -1
    largest ← pop_max(T1)
    curr_node ← T1.root
    height_diff ← height_t1 - height_t2

    while (height_diff ≥ 0)
        currnode = currnode.right
        height_diff ← height_diff - 1

    new_tree.root = largest
    new_tree.left = curr_node..
    new_tree.right = T2'' _node

    parent ← T1.root
    while (parent.right != curr_node)
        parent ← parent.right

    parent.right ← new_tree
    rebalance(T1)
    return (T1)
```

Figure 1: Pseudocode for Join(T1, T2)

To begin we are going to get the heights of both T1 and T2, we will store these values inside height_t1 and height_t2. Since we are told that T1 has more nodes then T2 (and that all the values in T1 are smaller then those in T2) we will use the height later on to a node in T1 whose subtree is of an equal size as T2. [Getting the heights will take O(height_t1 + height_t2) time]

Before we do this however we are going to find delete the largest value in T1 (which we know will be smaller then all the value values in T2). Then we will traverse down to the right of T1 using a variable called curr_node which will find the height_t1 - height_t2 right most child. [popping and storing the largest value in T1 will take O(log(T1)) time, traversing T1 will also take O(log(T1)) time]

Now the curr_node will have a subtree with a height equal to T2, we will now create a new tree, using the largest value we stored at the start as the root (called variable largest). This root will be smaller then all values in T2 and larger then all values in curr_node subtree. Thus we will make the left side of the tree equal to curr_node and the right side equal to T2. [Creating a new tree will take O(1)]

We will then get the parent of current_node and from there add the new tree to the right side of it. We will then iterate up from the parent and balance the tree. [getting the parent will take O(log(T1)) time, and balancing the tree will take O(log(newTree))]

Therefore our total time complexity will be:

$O(height\_t1+height\_t2)+O(log(T1))+O(log(T1))+O(1)+O(log(newTree)) \in O(log(newTree)$

Which is the result as required