Problem 2

1.

```
function FindPrev(T, t_k)
    node = T.root
    prev_node = NIL
    while node != NIL
      if node.key >= t_k
          node = node.left
      else
          prev_node = node
          node = node.right
    return prev_node
```

2.

If the key larger or equal to t_k, t_{k-1} must be in the left subtree, therefore we go left.

If the key is smaller than t_k, there are two possibilities:

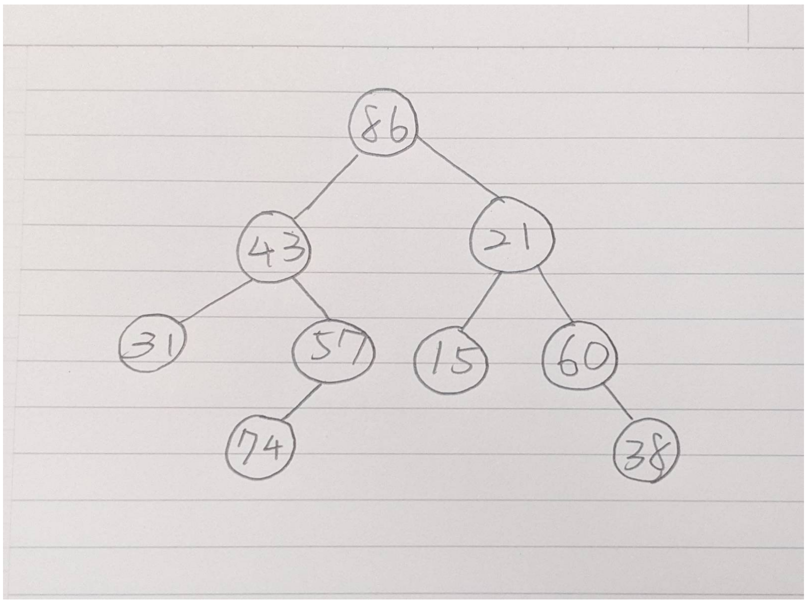
1. It's t_{k-1}
2. It is smaller than t_{k-1}

For case 1, because we go to the right subtree, every key is larger than t_{k-1} and larger or equal to t_k. `prev_node = node` won't be executed anymore and the algorithm works.

For case 2, t_{k-1} must be in the right subtree, therefore we go right.

If it receives t_1 as input, we always go left and `prev_node = node` won't be executed. The return value would be `NIL`.

3.



4.

Because `preorder[1]` is the root of the tree, by locating it in `inorder` we can extract (`inorder` , `preorder`) pair of left subtree and right subtree. Therefore if two trees have the same (`inorder` , `preorder`) pair, the root must be the same, two left subtrees have the same (`inorder` , `preorder`) pair, and two right subtrees also have the same (`inorder` , `preorder`) pair.

Doing this recursively for subtrees can show that this two trees must be the same.

5.

```
/* A.index(v) returns the index of v in array A */
function Reconstruct(inorder, preorder)
    if inorder.len == 0:
        return NIL
    rt = root()
    rt.key = preorder[1]
    l_size = inorder.index(rt.key) - 1
    r_size = inorder.len - l_size - 1
    l_inorder = inorder[:l_size+1]
    l_preorder = preorder[2:l_size+2]
    r_inorder = inorder[l_size+2:]
    r_preorder = preorder[l_size+2:]
    rt.left = Reconstruct(l_inorder, l_preorder)
    rt.right = Reconstruct(r_inorder, r_preorder)
    return rt
```

Time complexity: $O(n^2)$