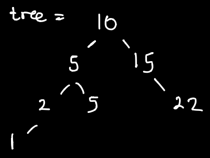
Input: BST
array

tree =

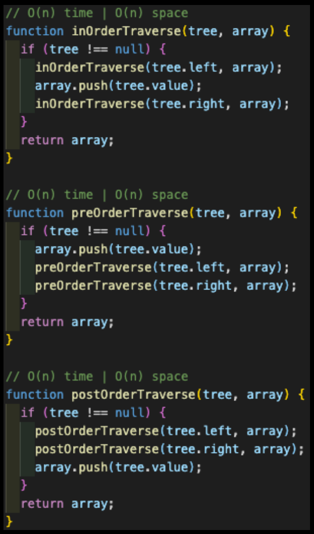


array = [ ]

Output: array

Traverse the BST, add its nodes' value to the input array
↳ inOrder, preOrder, postOrder

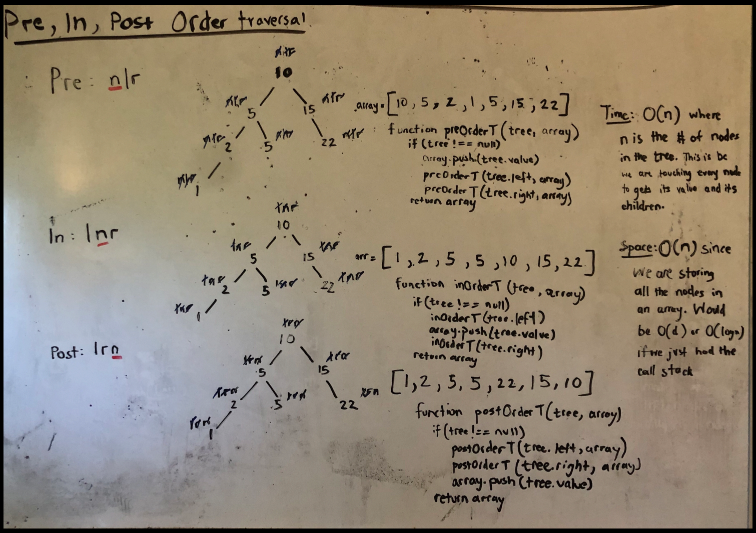
inOrder Traverse: [1, 2, 5, 5, 10, 15, 22]
preOrder Traverse: [10, 5, 2, 1, 5, 15, 22]
postOrder Traverse: [1, 2, 5, 5, 22, 15, 10]



```
// O(n) time | O(n) space
function inOrderTraverse(tree, array) {
  if (tree !== null) {
    inOrderTraverse(tree.left, array);
    array.push(tree.value);
    inOrderTraverse(tree.right, array);
  }
  return array;
}

// O(n) time | O(n) space
function preOrderTraverse(tree, array) {
  if (tree !== null) {
    array.push(tree.value);
    preOrderTraverse(tree.left, array);
    preOrderTraverse(tree.right, array);
  }
  return array;
}

// O(n) time | O(n) space
function postOrderTraverse(tree, array) {
  if (tree !== null) {
    postOrderTraverse(tree.left, array);
    postOrderTraverse(tree.right, array);
    array.push(tree.value);
  }
  return array;
}
```



Pre, In, Post Order traversal

Pre: nlr

array = [10, 5, 2, 1, 5, 15, 22]

```
function preOrderT(tree, array)
    if (tree !== null)
        array.push(tree.value)
        preOrderT(tree.left, array)
        preOrderT(tree.right, array)
    return array
```

In: lnr

arr = [1, 2, 5, 5, 10, 15, 22]

```
function inOrderT(tree, array)
    if (tree !== null)
        inOrderT(tree.left)
        array.push(tree.value)
        inOrderT(tree.right)
    return array
```

Post: lrn

[1, 2, 5, 5, 22, 15, 10]

```
function postOrderT(tree, array)
    if (tree !== null)
        postOrderT(tree.left, array)
        postOrderT(tree.right, array)
        array.push(tree.value)
    return array
```

Time: O(n) where n is the # of nodes in the tree. This is bc we are touching every node to gets its value and its children.

Space: O(n) since we are storing all the nodes in an array. Would be O(d) or O(logn) if we just had the call stack