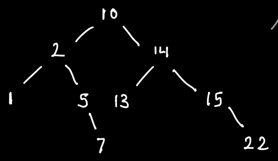
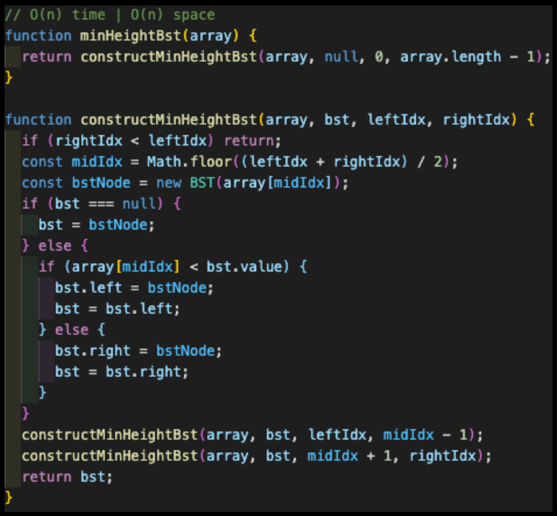
Input: Non empty sorted array of distinct integers

array = [1, 2, 5, 7, 10, 13, 14, 15, 22]

Output: Construct a BST from the integers and return the root of the BST



// this is just one example



```
// O(n) time | O(n) space
function minHeightBst(array) {
  return constructMinHeightBst(array, null, 0, array.length - 1);
}

function constructMinHeightBst(array, bst, leftIdx, rightIdx) {
  if (rightIdx < leftIdx) return;
  const midIdx = Math.floor((leftIdx + rightIdx) / 2);
  const bstNode = new BST(array[midIdx]);
  if (bst === null) {
    bst = bstNode;
  } else {
    if (array[midIdx] < bst.value) {
      bst.left = bstNode;
      bst = bst.left;
    } else {
      bst.right = bstNode;
      bst = bst.right;
    }
  }
  constructMinHeightBst(array, bst, leftIdx, midIdx - 1);
  constructMinHeightBst(array, bst, midIdx + 1, rightIdx);
  return bst;
}
```

Idea: Since the array is sorted and there are distinct integers, we can grab the mid value and set it as our root.

Then we can do a recursive call to grab the mid of the left side of the array (index 0 to index of currentMid - 1) and the right side of the array (index currentMid + 1 to end of array) to repeat the same steps.

We check if our new BST node is < our Curr Node value, if so we add to rootNode.left otherwise we add to rootNode.right

Time: O(n) (where n is # of elements in array) since we traverse the entire input array and create new Nodes (O(1) operations)

Space: O(n) since we end up with a BST of n nodes.

The recursive call stack has no affect as it is smaller than n