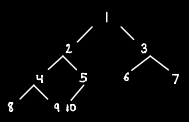
Branch Sums

Input: tree =



Output: [15, 16, 18, 10, 11]

15 = 1 + 2 + 4 + 8
16 = 1 + 2 + 4 + 9
18 = 1 + 2 + 5 + 10
10 = 1 + 3 + 6
11 = 1 + 3 + 7

Input: Takes in a binary tree

Output: returns a list of its branch sums ordered from left most branch sum to right most branch sum

Each binary tree node has an integer value, a left child node, and a right child node

children nodes can be BinaryTree nodes themselves or None/null

```
// O(n) time | O(n) space where n is the number of nodes in the Binary Tree
function branchSums(root) {
  const arr = [];
  let sum = 0;
  branchSumsHelper(root, sum, arr);
  return arr
}

function branchSumsHelper(node, sum, arr) {
  if (node === null) return

  sum += node.value;

  if (node.left === null && node.right === null) {
    arr.push(sum);
    return
  }

  branchSumsHelper(node.left, sum, arr)
  branchSumsHelper(node.right, sum, arr)
}
```

We first declare an empty array (arr) that will hold our summed values of each branch

The sum variable holds our running sum as we traverse down the a branch

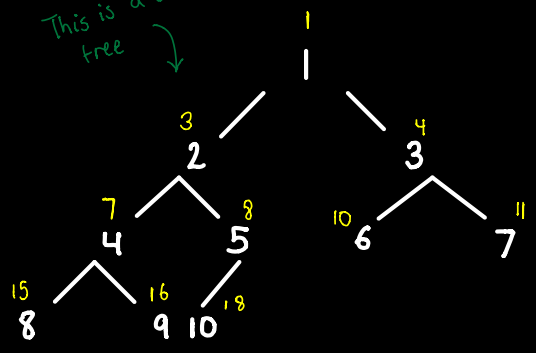
We then pass the root node, sum and arr to our recursive fn

Starting at the root node, we increment the sum by the value of the root node

We then check if the next node is null (if it is, we just return if it isn't, we add its value to the current sum

We then check if the current node has children or not. If no child nodes (left/right) exist then we are at a leaf node. We then push sum to arr

This is a balanced tree



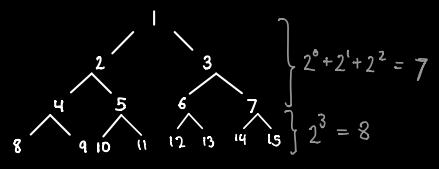
Time: O(n) where n is the number of nodes in the binary tree

Space: O(log n) for the recursive functions (bc of the implicit stack) → For a balanced binary tree. O(n) for a one branch tree

BUT

we won't have more terms in our arr than we do leaf nodes

Leaf nodes make up roughly half the items in the balanced binary search tree:



Therefore, our array will get, at worst, n/2 long or O(n)