Strategy – Recursion in Nested Structures

*/\* flattenDeep() \*/*

*// we declare an output array within the function;*

*// interating through the nested array,*

*// we check if the element is an array.*

*// and if so, we push the spread elements of the recursive call to the element.*

*// and else, we push the element into the array*

*// and outside of the iterative loop, we return the result array.*

*/\* sumNestedArrays \*/*

*// should be doing basically the same thing as above.*

*// We can return the result of reducing the array, with the accumulator initialized to zero.*

*// but within the callback of reduce, we will make a check if the "current" element is an array (and this can initialize a ternary conditional operator)*

*// and if so, we will add the result of a recursive call to that current element.*

*// and else, we will add the element to the accumulator.*

*/\* linkedListLength \*/*

*// declare "current" node to the head of the list*

*// and declare a count to 0 to keep track of the number of traversals.*

*// and while we the "curr" is defined -- at the tail, the current node will be null --*

*// we reassign current node to the next node (traverse)*

*// and increment the count by one.*

*// once we break out of our "while" loop,*

*// and we return the count.*

*/\* linkedListLengthRecursive \*/*

*// we pass in the count as a default parameter of the function*

*// if there is no head, then we return the count.*

*// and we make a recursive call, passing in the count with the prefix incrementor, to make sure we pass in the incremented value.*

*/\* BST height \*/*

*// if there's no root, we return -1 (doesn't exist)*

*// otherwise, we return one plus the maximum of recursive calls to the left and right branches.*

*// this way, if there is just one root, we return a height of 0.*