Product Sum

Input: [5, 2, [7, -1], 3, [6, [-13, 8], 4]]

Output: 12 // 5 + 2 + 2(7 - 1) + 3 + 2(6 + 3(-13 + 8) + 4)

Input: A "special" array a non-empty array that contains either integers or other "special" arrays.

Output: The product sum the sum of its elements, where "special" arrays inside it are summed themselves and then multiplied by their level of depth

```
// O(n) time | O(d) space
function productSum(array, depth = 1) {
  let sum = 0;
  for (let i = 0; i < array.length; i++) {
    if (Array.isArray(array[i])) {
      sum += productSum(array[i], depth + 1);
    } else {
      sum += array[i];
    }
  }
  return sum * depth;
}
```

Idea is to keep track of the sum and the depth. Every time we encounter a new array, we increment the depth and call productSum again.

Time: O(n) where is the TOTAL # of elements in the original array and all elements in each sub array (in this case 12) since we are iterating over every element and it's contents

Space: O(d) where d is the largest depth of the array (we use constant space in every other case eg. for sum, depth) since we are using space on the call stack due to our recursive calls (the stack would be the max depth, 3 in this case)