Array of Array Products

Given an array of integers arr, write a function that returns another array at the same length where the value at each index i is the product of all array values except arr[i].

Solve without using division and analyze the runtime and space complexity

```
Example: given the array [2, 7, 3, 4]
your function would return: [84, 24, 56, 42] (by calculating: [7*3*4, 2*3*4, 2*7*4, 2*7*3])
```

Hints & Tips

- If your peer is stuck and can't think of anything, offer the brute force solution and ask how to improve it
- If your peer cannot improve the brute force solution, ask what part of the calculations is repetitive and can be re-used
- If still stuck, ask your peer to manually compute the solution to [1, 2, 3, 4, 5]. it's [120, 60, 40, 30, 24] by calculating [2*3*4*5,condition 1*2*4*5, 1*2*3*5, 1*2*3*4] Can you peer think of something to re-use from looking at the numbers? What is the repeating pattern?
- If still stuck, ask your peer how to create these sequences: [2*3*4*5, 1*3*4*5, 1*2*4*5, 1*2*3*5, 1*2*3*4] and its complement [2*3*4*5, 1*3*4*5, 1*2*4*5, 1*2*3*5, 1*2*3*4]
- The more hints needed, the more you should reduce your peer's rating on the knowledge on the problem solving section of the interview feedback
- Watch for forgotten indices on your peer's solution and for unnecessary iterations

Solution

A brute force approach would use two nested loops: for each index i in arr loop over arr while multiplying all other values and place the result on the **i**th element of the new array. This would give us a runtime complexity of $O(n^2)$. We can do better.

When we multiply all values of arr before and after each index, we get our answer — the product of all the integers except arr[i].

```
function calcProductArray(arr):
  n = length(arr)
   productArr = []
   for i from 0 to n-1:
      productArr[i] = 1
   product = 1
  for i from 0 to n-1:
      productArr[i] *= product
      product *= arr[i]
   product = 1
  for i from n-1 to 0:
      productArr[i] *= product
      product *= arr[i]
   return productArr
```

Runtime Complexity: two passes through arr and constant time work for each value in it, bring us to linear **O(n)** runtime complexity.

Space Complexity: from using an additional array of length n to hold the products, we get a linear **O(n)** space complexity.