Problem Link:

https://leetcode.com/problems/product-of-array-except-self

Problem Description:

Given an integer array `nums`, return *an array* `answer` *such that* `answer[i]` *is equal to the product of all the elements of* `nums` *except* `nums[i]`.

The product of any prefix or suffix of `nums` is **guaranteed** to fit in a **32-bit** integer.

You must write an algorithm that runs in `O(n)` time and without using the division operation.

Problem Approach:

As hinted in the problem description, we'll calculate prefix and suffix of the elements and multiply them, and that's gonna be our answer.

Sample Test Case:

Input: nums = [1,2,3,4]
Output: [24,12,8,6]

Solution:

We start by creating a list that'll store our answer, which is a list of 1s (In case of a 0 based list/array in other languages, we'll need to initialize the first element by 1, otherwise, the result will always be 0). The approach is simple, first, we calculate the prefix-product. That is, the product of the elements before the current element in the iteration. We generalize a formula that calculates the prefix-product by, multiplying the element just before, with the prefix-product of the previous element. Once it's done, we initialize a variable called suffix-product, that calculates the product of elements after the current element. So, logically, we start by traversing the list from the end, towards the start. With every iteration, we first multiply the current element of the array, which is the prefix product of the original nums array, with the suffix product. Then, we multiply the suffix product with the current element in the nums array. Once this loop finishes, the answer array will have the product of all the elements except the current element from the nums array, and that is our answer. There can be a different approach where we calculate prefix and suffix products individually and then multiply them to give the answer, but this is more optimal as it uses less space.

Code (Python):

```
def productExceptSelf(self, nums: List[int]) -> List[int]:
        answer = [1] * len(nums)
        for i in range(1,len(nums)):
            answer[i] = answer[i-1] * nums[i-1]
        suffixProduct = 1
        for i in range(len(nums)-1,-1,-1):
            answer[i] *= suffixProduct
            suffixProduct *= nums[i]
        return answer
```