Product of Array Except Self

Problem: 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.**

Example 1:

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

Explanation:

```
nums[i]=1      2 * 3 * 4 = 24
nums[i]=2      1 * 3 * 4 = 12
nums[i]=3      1 * 2 * 4 = 8
nums[i]=4      1 * 2 * 3 = 6
```

brute-force Solution(Using two for loops):

```
    for i = 0 -------> n
     for j = 0 --------> n
          int product = 1
          if i == j :
               Skip that
          calculate the product of remaining element
          product = product * product[j]
       answer[i] = product
    return Answer
```

It could be Time Limit Exceeded , we need to optimize this because it has TC : O(n^2) , SC = O(n) complexity.

Optimized Solution(using prefix and suffix array):

```java
class Solution {
    public int[] productExceptSelf(int[] nums) {
        int[] suffixP = new int[nums.length];
        int[] prefixP = new int[nums.length];
        int[] answer = new int[nums.length];

        prefixP[0] = 1;
        for(int i = 1; i < nums.length; i++){
            prefixP[i] = nums[i - 1] * prefixP[i - 1];
        }

        suffixP[nums.length - 1 ] = 1;
        for(int k = nums.length - 2; k >= 0; k--){
            suffixP[k] = nums[k + 1] * suffixP[k + 1];
        }
         for(int s = 0; s < nums.length; s++){
            answer[s] = prefixP[s] * suffixP[s];
        }
        return answer;
    }
}
// TC: O(n)
// SC: O(n)
```