# Documentation

The "Product of Array Except Self" problem is a common algorithm challenge where we need to return an array where each element at index (i) represents the product of all elements in the input array except for the element at (i) itself. This solution must avoid division to prevent issues with zero elements and requires a time complexity of (O(n)) for efficiency. Furthermore, achieving an (O(1)) space complexity, beyond the result array, is essential for scalability, especially with large input sizes.

To approach this, we use a two-pass strategy, accumulating the products of elements on either side (left and right) of each position in the array. In the first pass, we populate an output array where each position (i) stores the cumulative product of elements to the left of (i). By maintaining a running product variable that multiplies each element from left to right, we can efficiently accumulate these left products without needing an additional array, thus conserving space.

Once the left products are recorded in the output array, we proceed with the second pass, which works from the right side of the array toward the left. This time, we maintain a running product of elements to the right of each position and multiply it by the current value in the output array (which now holds the left product for each element). By combining the left and right accumulations, we give the final product of all elements except the one at each index.

The dual-pass approach enables us to solve the problem in (O(n)) time since each pass goes through the array only once. Additionally, the space complexity meets the (O(1)) requirement as we only use a few extra variables to track left and right products. This method is efficient for large arrays and circumvents the complications of handling zeros in the array, which would have posed issues if division were used.

This solution also efficiently handles edge cases. For instance, when zeroes are present in the input, they naturally result in zero in the output except where a single zero is excluded by the logic of multiplying only left and right products for each position. This robustness ensures that the solution can handle any valid input array length, as per the constraints, making it a preferred algorithm for similar product-based array manipulation tasks.