🎯 Rotate Array

https://leetcode.com/problems/rotate-array/

🛕 Temple Queue VIP Analogy – Rotate Array Problem 🎯

Imagine you're in a long *darshan* queue at a famous Indian temple like Tirupati or Vaishno Devi 🙏.
You're standing patiently in line with hundreds of others:
🧍‍♂️🧍‍♀️🧍‍♂️ … all waiting for your turn.

Suddenly...

📢 **"VIPs are coming! Make way!"**

Just like that, the **last k people** in the queue — usually some VIPs with passes 😎🎫 —
are moved straight to the **front**. No logic. No fairness. Just pure desi VIP treatment.

Example:

Original Queue: `[Amit, Bhavna, Chirag, Deepa, Esha, Farhan, Gita],k = 3`

VIPs from the end: `[Esha, Farhan, Gita]`

New Queue after VIP rotation: `[Esha, Farhan, Gita, Amit, Bhavna, Chirag, Deepa]`

🔁 What just happened?

This is *exactly* what array rotation by k steps to the right means.

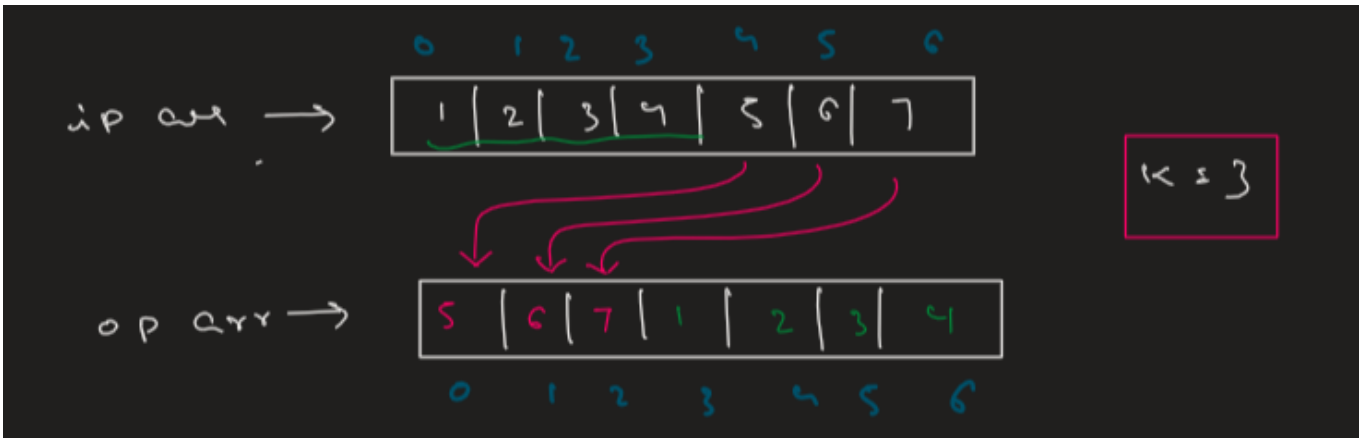
- The **last k elements** come to the **front**
- The remaining elements shift **back**

🍋 Moral of the story:

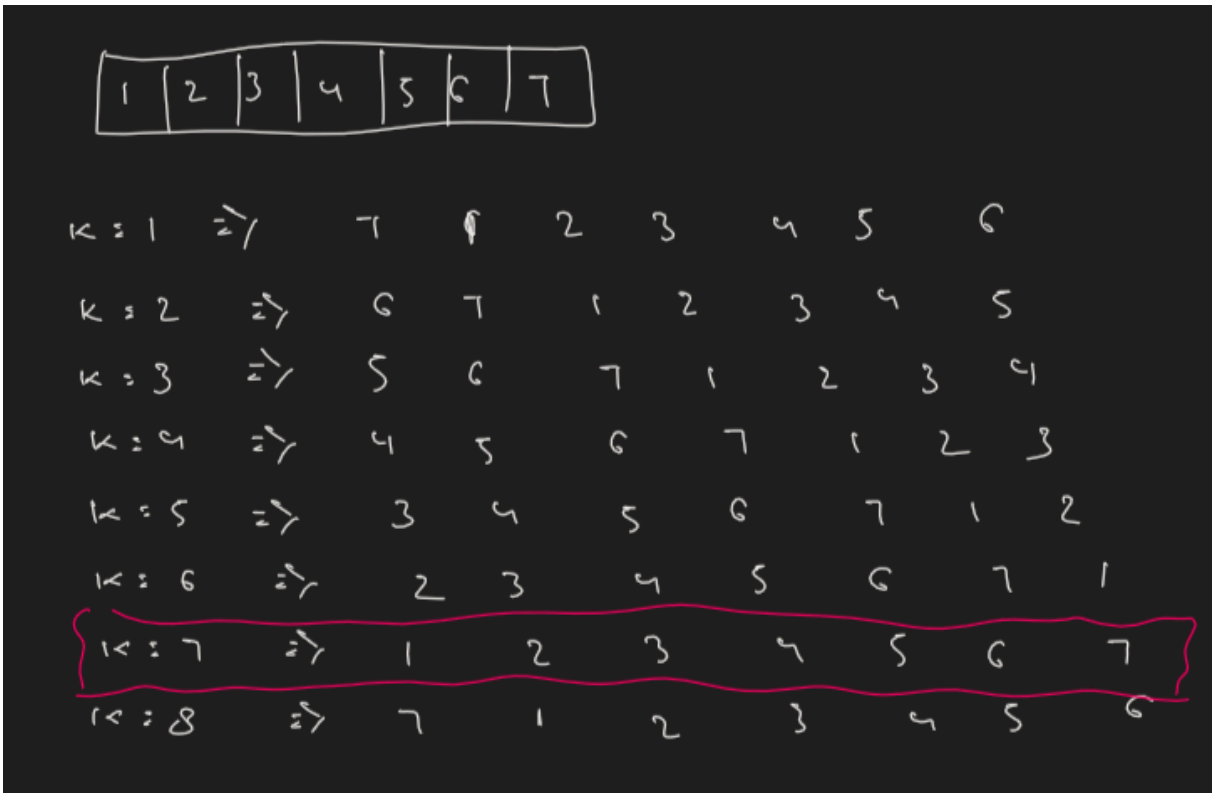
- VIPs get darshan first 🙌
- Life isn't fair… but the logic is clear 😅

Ex: Rotation in school life

- **Brute Force Solution:** Extra Array



The constraints don't mention that **k is always less than the size of the array**. So if **k = 200** and the **array size = 80**, it **is allowed** ?



- A **period** means something starts repeating after a certain interval.
- In this problem, once **k reaches or exceeds the array length**, the rotations start repeating, that is called periodic nature.

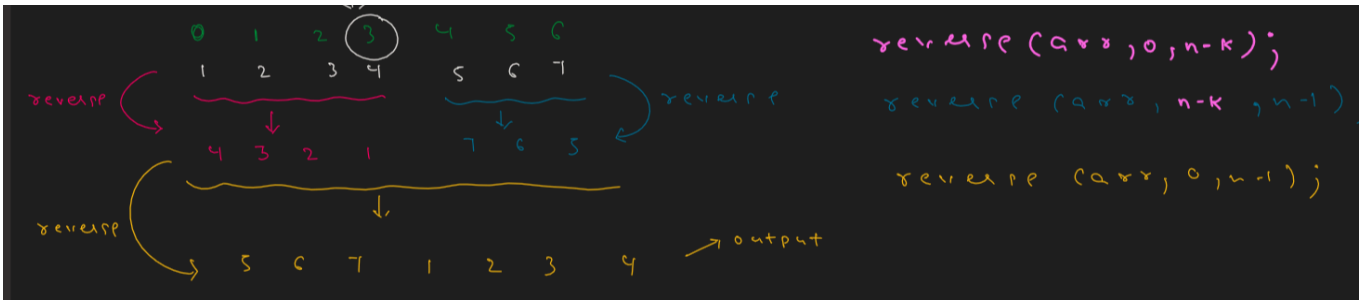
- For example, if the **array length is 7** and **k = 10**, rotating the array **10 times** gives the same result as rotating it **3 times**.
- How do we get 3? It's simply the modulus: **k % n = 10 % 7 = 3**
- That's why we use **k = k % n** in the code:
    - To handle cases where **k is greater than the array size**
    - To reduce it to an **equivalent smaller rotation**
    - To **avoid unnecessary rotations**
- N = 7, k = 143, I will find the 3rd rotation only.

```java
class Solution {
    public void rotate(int[] nums, int k) {
        int n = nums.length;

        // Normalize k to avoid unnecessary rotations if k > n
        k = k % n;

        // Create a new array to store the rotated result
        int[] numsExtra = new int[n];

        int j = 0;

        // Copy the last k elements of nums to the beginning of numsExtra
        for(int i = n - k; i < n; i++) {
            numsExtra[j] = nums[i];
            j++;
        }

        // Copy the first n - k elements to the remaining positions in numsExtra
        for(int i = 0; i < n - k; i++) {
            numsExtra[j] = nums[i];
            j++;
        }

        // Copy the rotated elements back into the original array
        for(int i = 0; i < n; i++) {
            nums[i] = numsExtra[i];
        }
    }
}
```

🔹 Steps to solve the problem

1. Understanding the problem requirements
2. Intuition,
3. Logic building,
4. Coding

🎯 Reversal algorithm



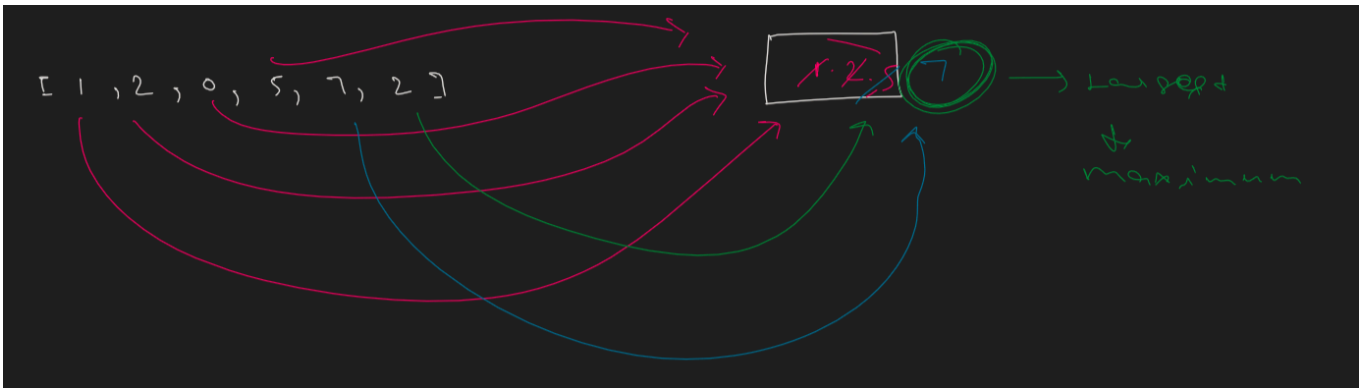

```java
public class Main {

	public static void main(String[] args) {

	  int[] arr = { 1, 2, 3, 4, 5, 6, 7 };
		int k = 3;
		rotate(arr, k);
		for (int i = 0; i < arr.length; i++) {
			System.out.print(arr[i] + " ");
		}
	}

	public static void rotate(int[] arr, int k) {
		int n = arr.length;
		k = k % n;

		// starting n-k part reversal
		reverseArrayPart(arr, 0, n - k - 1);

		// last k part reversal
		reverseArrayPart(arr, n - k, n - 1);

		// all element reversal
		reverseArrayPart(arr, 0, n - 1);
	}

	public static void reverseArrayPart(int[] arr, int i, int j) {
	       while (i < j) {
			int temp = arr[i];
			arr[i] = arr[j];
			arr[j] = temp;
			i++;
			j--;
		}
	}

}
```

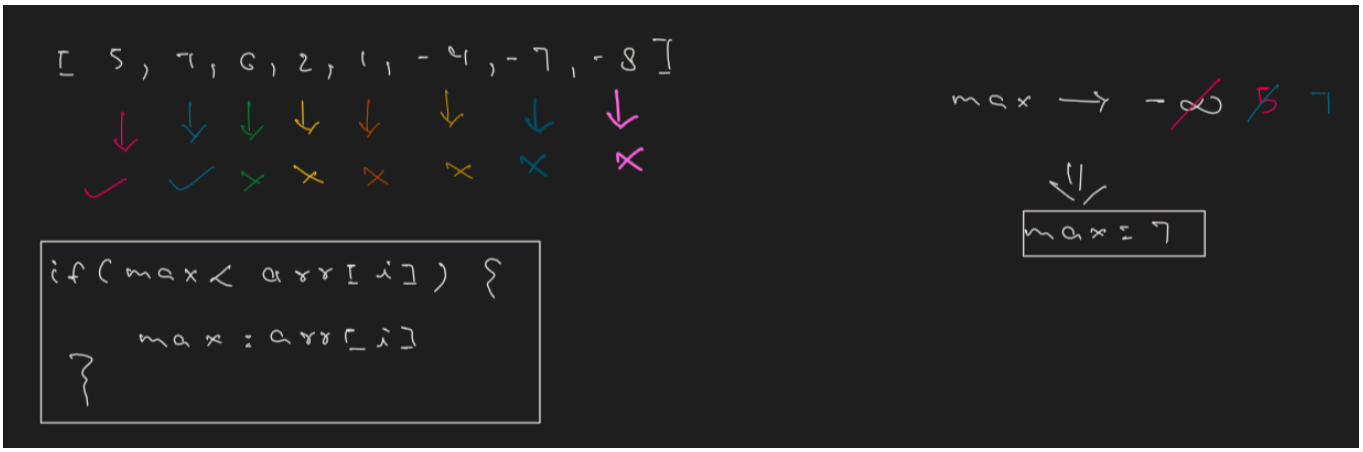
```java
class Solution {
    public void rotate(int[] nums, int k) {
        int n = nums.length;

        // Reduce k in case it's larger than the array size
        k = k % n;

        // Reverse the first part: from index 0 to n-k-1
        reverseArrayPart(nums, 0, n - k - 1);

        // Reverse the second part: from index n-k to n-1
        reverseArrayPart(nums, n - k, n - 1);

        // Reverse the entire array to get the final rotated version
        reverseArrayPart(nums, 0, n - 1);
    }

    // Helper method to reverse elements in array from index i to j
    public static void reverseArrayPart(int[] arr, int i, int j) {
        while (i < j) {
            // Swap elements at indices i and j
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++;
            j--;
        }
    }
}
```

🎯 Find Maximum in Array

To explain how to find the maximum value in an array, let's use the example of a piggy bank. Suppose you need to find the largest coin in the piggy bank. Here's how you'd do it:

You start by taking the first coin out and writing its value on a piece of paper. Then, you take out the second coin. If it's greater than the value written on the paper, you update the paper with this new value. You continue this process for each coin—if a coin is greater than the current value on the paper, you replace it; otherwise, you leave it as is.





```java
public class Main {
	public static void main(String[] args) {
		int[] arr = { 2, 3, 5, 1, 4, 11, 40, 51, 1, 15 };
		System.out.println(maxValue1(arr));
	}

	public static int maxValue3(int[] arr) {
		int max = Integer.MIN_VALUE;// -2^31
		for (int i = 0; i < arr.length; i++) {
			max=Math.max(max, arr[i]);
		}
		return max;
	}


	public static int maxValue2(int[] arr) {
		int max = Integer.MIN_VALUE;// -2^31
		for (int i = 0; i < arr.length; i++) {
			if (arr[i] > max) {
				max = arr[i];
			}
		}
		return max;
	}

	public static int maxValue1(int[] arr) {
		int max = arr[0];
		for (int i = 1; i < arr.length; i++) {
			if (arr[i] > max) {
				max = arr[i];
			}
		}
		return max;

	}
}
```

🎯 Product of Array Except Self

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

🧳 Family Packing for a Trip 🚗

👨‍👩‍👧‍👦 The Situation:

A family of 4 — **Papa, Mummy, Sonu, and Monu** — is packing bags for a road trip.
Each person is responsible for packing a **certain number of essential items**:

```
bags = [2, 3, 5, 4]
```

- Papa packed 2 items
- Mummy packed 3
- Sonu packed 5
- Monu packed 4

🎯 The Mission:

Mummy wants to know:

> "If I remove **each** person's contribution, how much effort did **everyone else** make *together*?"

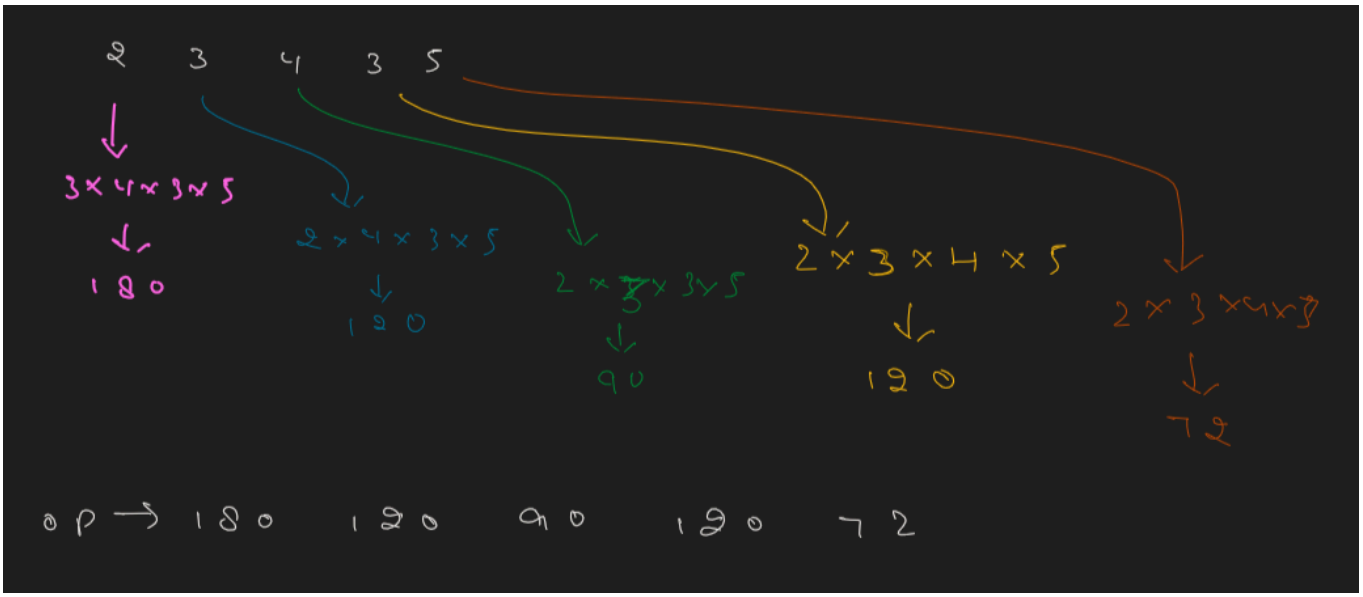
In other words: **product of all other contributions**, for **each** family member — but without using division!
(Because Indian moms don't divide — they **delegate smartly** 😎)

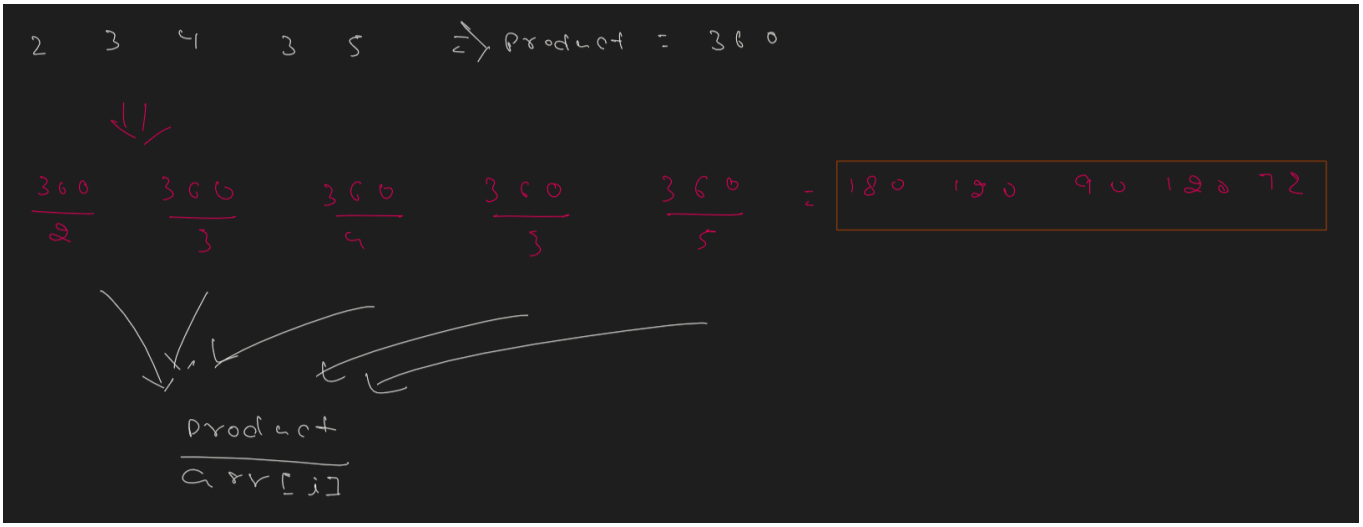
✅ Desired Output:

- Papa's contribution removed: `3 × 5 × 4 = 60`
- Mummy's removed: `2 × 5 × 4 = 40`
- Sonu's removed: `2 × 3 × 4 = 24`
- Monu's removed: `2 × 3 × 5 = 30`

Final result: `[60, 40, 24, 30]`

❌ **Brute Force Solution:** Using Division Operator





```java
public class Main {

	public static void main(String[] args) {
	int[] arr = {2, 3, 4, 3, 5};
	// System.out.println(product(arr));

	int[] nums = productOfArrayExceptSelf(arr);

	for(int i=0; i<nums.length; i++){
		System.out.print(nums[i]+" ");
	}

	}

	public static int product(int[] arr){
	int prod = 1;
	int n = arr.length;
	for(int i=0; i<n; i++){
		prod = prod*arr[i];
	}
	return prod;
	}

	public static int[] productOfArrayExceptSelf(int[] arr){
	int prod = product(arr);
	int n = arr.length;
	int[] op = new int[n];

	for(int i=0;i<n;i++){
		op[i] = prod/arr[i];
	}

	return op;
	}

}
```

🧠 Real-Life Jugaad: How Mummy Thinks

She doesn't divide. She uses **"left and right contribution tracking"**:

🧩 **Step 1: Left-side Contribution (before each person)**

```
left = [1, 2, 2×3=6, 6×5=30] → [1, 2, 6, 30]
```

🧩 **Step 2: Right-side Contribution (after each person)**

```
right = [3×5×4=60, 5×4=20, 4, 1] → [60, 20, 4, 1]
```

💥 **Final Output:**

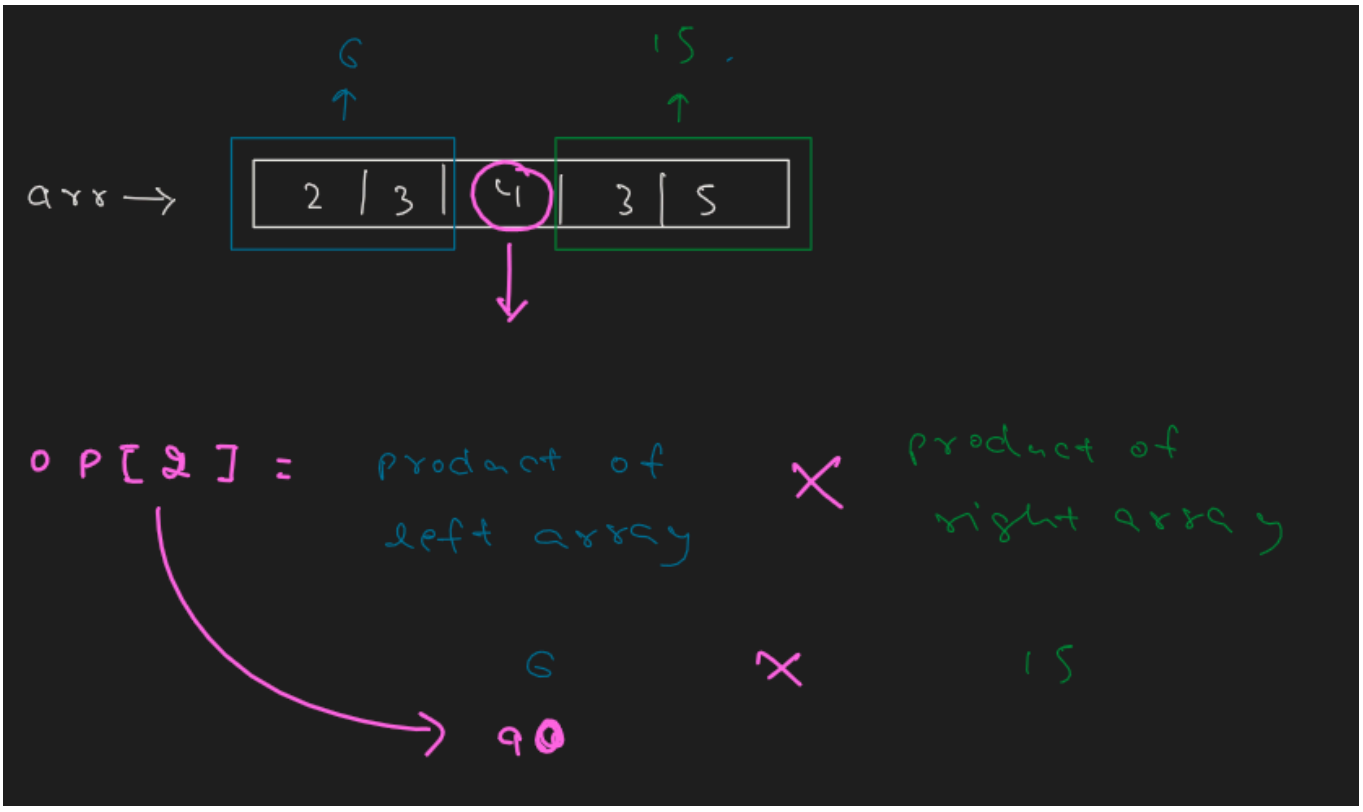
```
result = [1×60, 2×20, 6×4, 30×1] → [60, 40, 24, 30]
```

😄 Mummy's Wisdom:

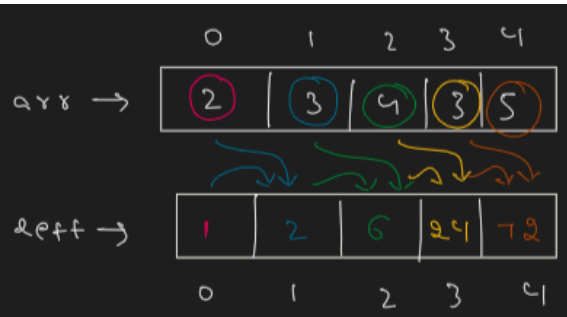
"Now I know who's slacking and who's actually pulling their weight! Next trip, I'm giving Papa fewer tasks!" 😂

✅ Why This Analogy Works:

- **Each person's role matters**, but we want to measure their absence's effect.
- In real life, we often want to know: **"If this person didn't help, how much would be left for others?"**

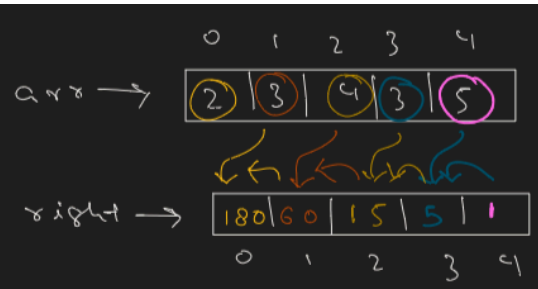


To solve this problem, I need to find the **prefix and suffix product arrays**. Suppose I'm calculating the left (prefix) array — **for each element, I will leave out that specific element and calculate the product of all the elements to its left**. I'll do the same for the right (suffix) array — leaving the current element and finding the product of all the elements to its right. The final result for each position will be the product of the corresponding values from the left and right arrays.




```
left[0] = 1
```

```
left[1] = left[0] * arr[0]
left[2] = left[1] * arr[1]
left[3] = left[2] * arr[2]
left[4] = left[3] * arr[3]

for(int i = 1; i < n; i++) {
    left[i] = left[i-1] * arr[i-1]
}
```




```
right[n-1] = 1
```

```
right[3] = right[4] * arr[4]
right[2] = right[3] * arr[3]
right[1] = right[2] * arr[2]
right[0] = right[1] * arr[1]

for(int i = n-2; i >= 0; i--) {
    right[i] = right[i+1] * arr[i+1]
}
```

```java
public class Main {

	public static void main(String[] args) {

	  int[] arr = { 1, 2, 3, 4 };
		int[] a = productOfArray(arr);
		for (int i = 0; i < a.length; i++) {
			System.out.print(a[i] + " ");
		}

	}

	public static int[] productOfArray(int[] arr) {

		int n = arr.length;
		int[] left = new int[n];
		int[] right = new int[n];

		left[0] = 1;
		for (int i = 1; i < n; i++) {
			left[i] = left[i - 1] * arr[i - 1];
		}

		right[n - 1] = 1;
		for (int i = n - 2; i >= 0; i--) {
			right[i] = right[i + 1] * arr[i + 1];
		}

		for (int i = 0; i < n; i++) {
			left[i] = left[i] * right[i];
		}

		return left;
	}

}
```

```java
class Solution {
    public int[] productExceptSelf(int[] nums) {
        int n = nums.length;

        // Arrays to store left and right products
        int[] left = new int[n];
        int[] right = new int[n];

        // left[i] contains product of all elements to the left of index i
        left[0] = 1;
        for (int i = 1; i < n; i++) {
            left[i] = left[i - 1] * nums[i - 1];
```

```
        }

        // right[i] contains product of all elements to the right of index i
        right[n - 1] = 1;
        for (int i = n - 2; i >= 0; i--) {
            right[i] = right[i + 1] * nums[i + 1];
        }

        // Multiply left and right products for the final result
        for (int i = 0; i < n; i++) {
            left[i] = left[i] * right[i];
        }

        return left; // left now contains the result

    }

}
```

🔹 Prefix-Suffix Concept

In the above question, the rainwater trapping problem uses the prefix-suffix concept. For example, if I write "Mr Akarsh", "Mr" is a prefix. Similarly, if I write "Mr Akash Jaiswal", then "Jaiswal" is the suffix — basically, the suffix is the ending and the prefix is the beginning. In this question, to calculate the value at the `i`th index, I used the prefix-suffix idea. I calculated all the maximum values *before* the `i`th index and stored them in the `left` array (which you can think of as the prefix). I then calculated all the maximum values *after* the `i`th index and stored them in the `right` array (which acts like the suffix). So, the `left` array is the prefix, and the `right` array is the suffix.