

Given an integer array `nums`, rotate the array to the right by `k` steps, where `k` is non-negative.

Example 1:

Input: `nums = [1,2,3,4,5,6,7]`, `k = 3`

Output: `[5,6,7,1,2,3,4]`

Explanation:

rotate 1 steps to the right: `[7,1,2,3,4,5,6]`

rotate 2 steps to the right: `[6,7,1,2,3,4,5]`

rotate 3 steps to the right: `[5,6,7,1,2,3,4]`

1 2 3 4 5 6 7

7 1 2 3 4 5 6

6 7 1 2 3 4 5

k=3

`[1, 2, 3, 4, 5, 6, 7]` $k=10$

$k=1$ 7 1 2 3 4 5 6

$k=2$ 6 7 1 2 3 4 5

$k=3$ 5 6 7 1 2 3 4

$k=4$ 4 5 6 7 1 2 3

$k=5$ 3 4 5 6 7 1 2

$k=6$ 2 3 4 5 6 7 1

$k=7$ 1 2 3 4 5 6 7

$k=8$ 7 1 2 3 4 5 6

$k=9$ 6 7 1 2 3 4 5

$k=10$ 5 6 7 1 2 3 4

basic

Time same

$k=10$

$142 \div 7 = 20.28$

7 14 21 28 35 42

4 5 6 7 1 2 3

0 1 2 3 4 5 6

$n=7$

1 2 3 4 5 6 7

6 7 1 2 3 4 5

6 1 2 3 4 5 6

```
public static void Rotate(int[] arr, int k) {
    int n = arr.length;
    k = k % n;
    for(int j=1; j<=k; j++) {
        int item = arr[n-1];
        for (int i = n-2; i >= 0; i--) {
            arr[i+1] = arr[i];
        }
        arr[0] = item;
    }
}
```

① Shifting $n-k$ elements reverse

② last k elements reverse

③ All elements reverse

$n-k$

$k=3$

1 2 3 4 5 6 7

0 1 2 3 4 5 6

4 3 2 1 7 6 5

5 6 7 1 2 3 4

5 6 7 1 2 3 4

5 6 7 1 2 3 4

```
public static void Rotate(int[] arr, int k) {
    int n = arr.length;
    k = k % n;
    // 1. reverse starting ke n-k element
    Reverse(arr, 0, n-k-1);
    // 2. last ke K element
    Reverse(arr, n-k, n-1);
    // 3. all element
    Reverse(arr, 0, n-1);
}
```

0 $n-k-1$

0 $n-1$

2+4+2+2+5=16

`[5, 3, 1, 2, 7, 4, 11, 6]`

0 1 2 3 4 5 6 7

→ L 5 5 5 5 7 7 7 7

→ R 7 7 7 7 6 6 6

$\min(S_1) - 5 = 0$

$\min(S_2) - 3 = 2$

$\min(S_3) - 1 = 4$

$\min(S_4) - 2 = 3$

$\min(S_5) - 7 = 0$

$\min(S_6) - 4 = 2$

$\min(S_7) - 11 = 0$

$\min(S_8) - 6 = 0$

5 3 1 2 7 4 11 6

0 1 2 3 4 5 6 7

5 3 1 2 7 4 11 6

0 1 2 3 4 5 6 7

5 3 1 2 7 4 11 6

0 1 2 3 4 5 6 7

5 3 1 2 7 4 11 6

0 1 2 3 4 5 6 7

`[5, 3, 1, 2, 7, 4, 11, 6]`

0 1 2 3 4 5 6 7

$\min(L_0, R_0) - arr[0] = 5 - 5 = 0$

$\min(L_1, R_1) - arr[1] = 3 - 3 = 0$

$\min(L_2, R_2) - arr[2] = 1 - 1 = 0$

$\min(L_3, R_3) - arr[3] = 2 - 2 = 0$

$\min(L_4, R_4) - arr[4] = 7 - 7 = 0$

$\min(L_5, R_5) - arr[5] = 4 - 4 = 0$

$\min(L_6, R_6) - arr[6] = 11 - 11 = 0$

$\min(L_7, R_7) - arr[7] = 6 - 6 = 0$

`[5, 3, 1, 2, 7, 4, 11, 6]`

0 1 2 3 4 5 6 7

$i=1$ $L(i) = \max(L(i-1), arr[i])$

$i=n-2$ $R(i) = \max(R(i+1), arr[i])$

$L(0) = \max(L(0), arr[0])$

$L(1) = \max(L(1), arr[1])$

$L(2) = \max(L(2), arr[2])$

$L(3) = \max(L(3), arr[3])$

$L(4) = \max(L(4), arr[4])$

$L(5) = \max(L(5), arr[5])$

$L(6) = \max(L(6), arr[6])$

$L(7) = \max(L(7), arr[7])$

$R(7) = \max(R(7), arr[7])$

$R(6) = \max(R(6), arr[6])$

$R(5) = \max(R(5), arr[5])$

$R(4) = \max(R(4), arr[4])$

$R(3) = \max(R(3), arr[3])$

$R(2) = \max(R(2), arr[2])$

$R(1) = \max(R(1), arr[1])$

$R(0) = \max(R(0), arr[0])$

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.

24

5 120

24 40 20 60

120 120 120 120

5 5 4 2

21 3 6 4 5

0 1 2 3 4

$L(0) = L[0] \times arr[0]$

$L(1) = L(0) \times arr[1]$

$L(2) = L(1) \times arr[2]$

$L(3) = L(2) \times arr[3]$

$L(4) = L(3) \times arr[4]$

$R(0) = R[0] \times arr[0]$

$R(1) = R(0) \times arr[1]$

$R(2) = R(1) \times arr[2]$

$R(3) = R(2) \times arr[3]$

$R(4) = R(3) \times arr[4]$

$answer[i] = L(i) \times R(i)$

360 240 120 180 144

1 2 6 3 4

0 1 2 3 4

360 240 120 180 144