

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 ] k=3

5 6 7 1 2 3 4 k=3

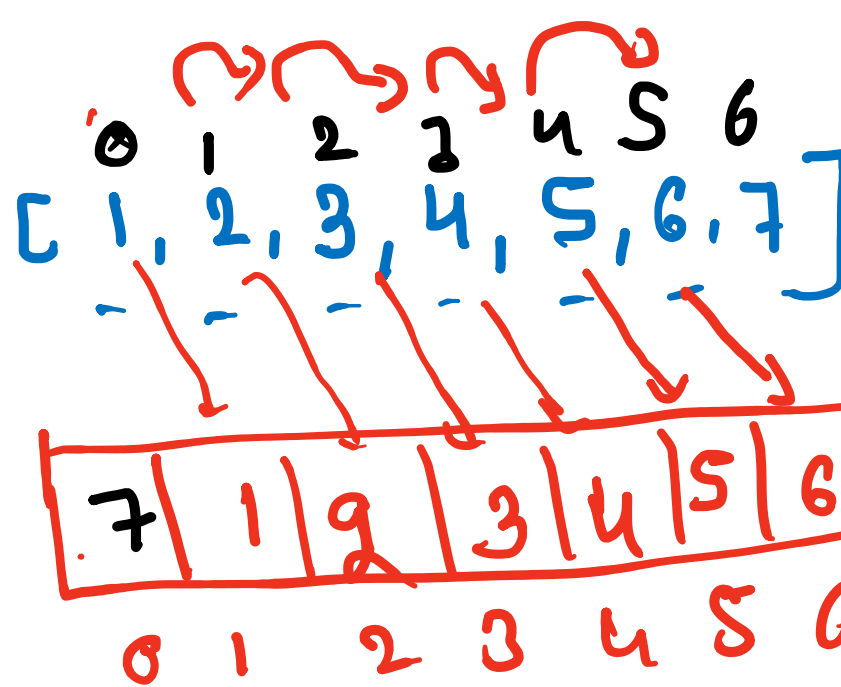
6 7 1 2 3 4 5 k=2

[ 1 2 3 4 5 6 7 ]  
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

k=143

k = k % n

n = arr.length -



n-2=5

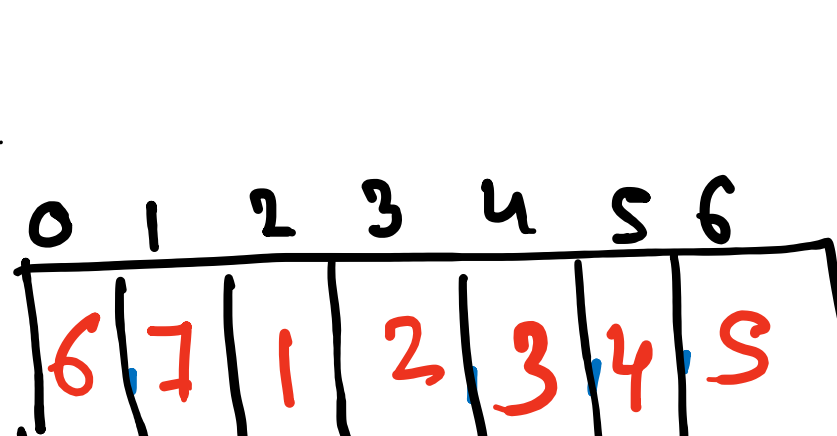
1 5 5 x 1 2 6 = 2 cr

item = arr[n-1] → arr[6]

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

i=5 arr[6] = arr[5]  
i=4 arr[5] = arr[4]  
i=3 arr[4] = arr[3]  
i=2 arr[3] = arr[2]  
i=1 arr[2] = arr[1]  
i=0 arr[1] = arr[0]  
arr[0] = item

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



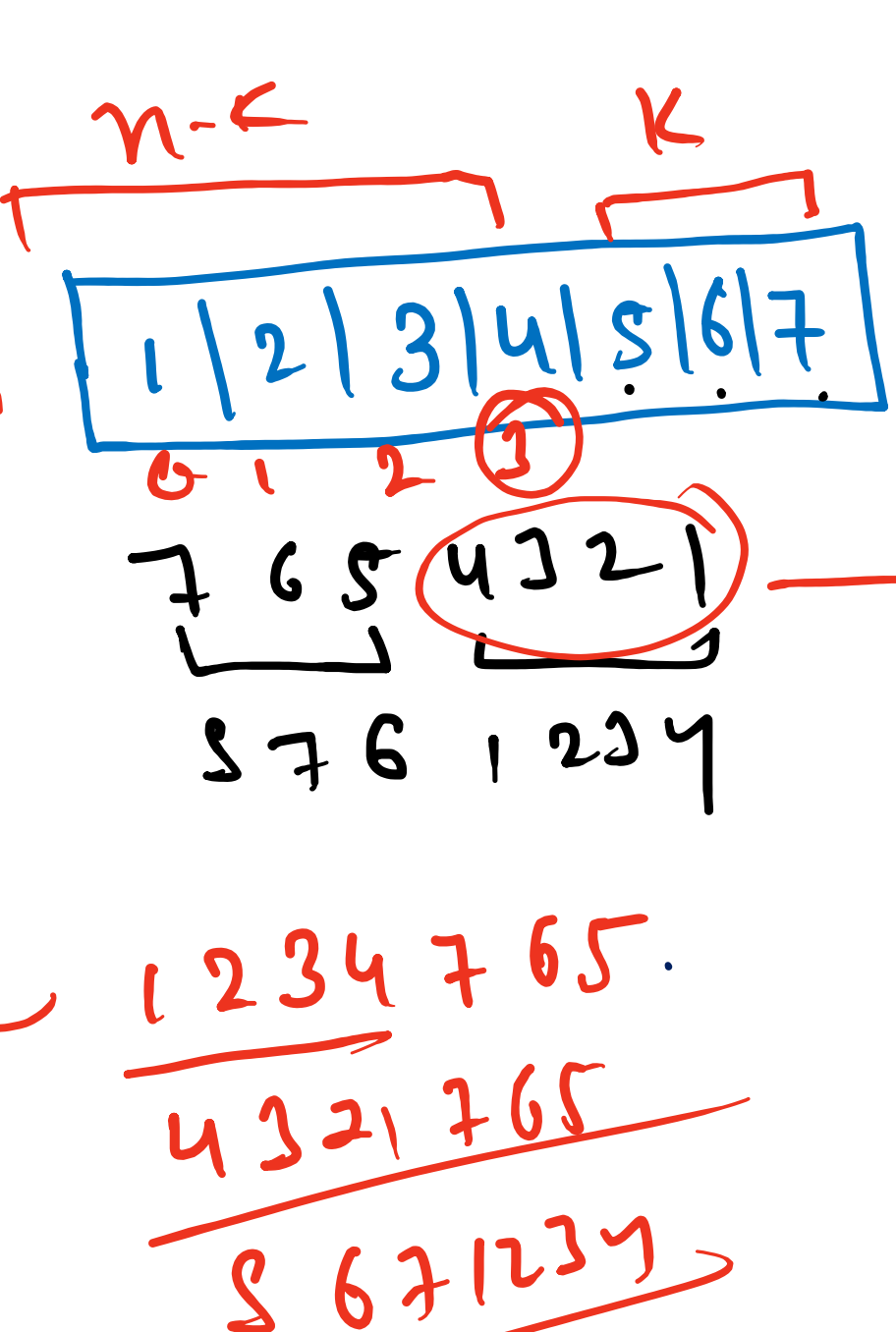
i=5 arr[6] = arr[5]  
i=4 arr[5] = arr[4]  
i=3 arr[4] = arr[3]  
i=2 arr[3] = arr[2]  
i=1 arr[2] = arr[1]  
i=0 arr[1] = arr[0]

1088 Kelen

Reversal Algo

7 6 5 4 3 2 1

Si=0  
ei=6  
Si=4 → n-1  
ei=6 → n-1



k=3

7 6 5 4 3 2 1

n=7

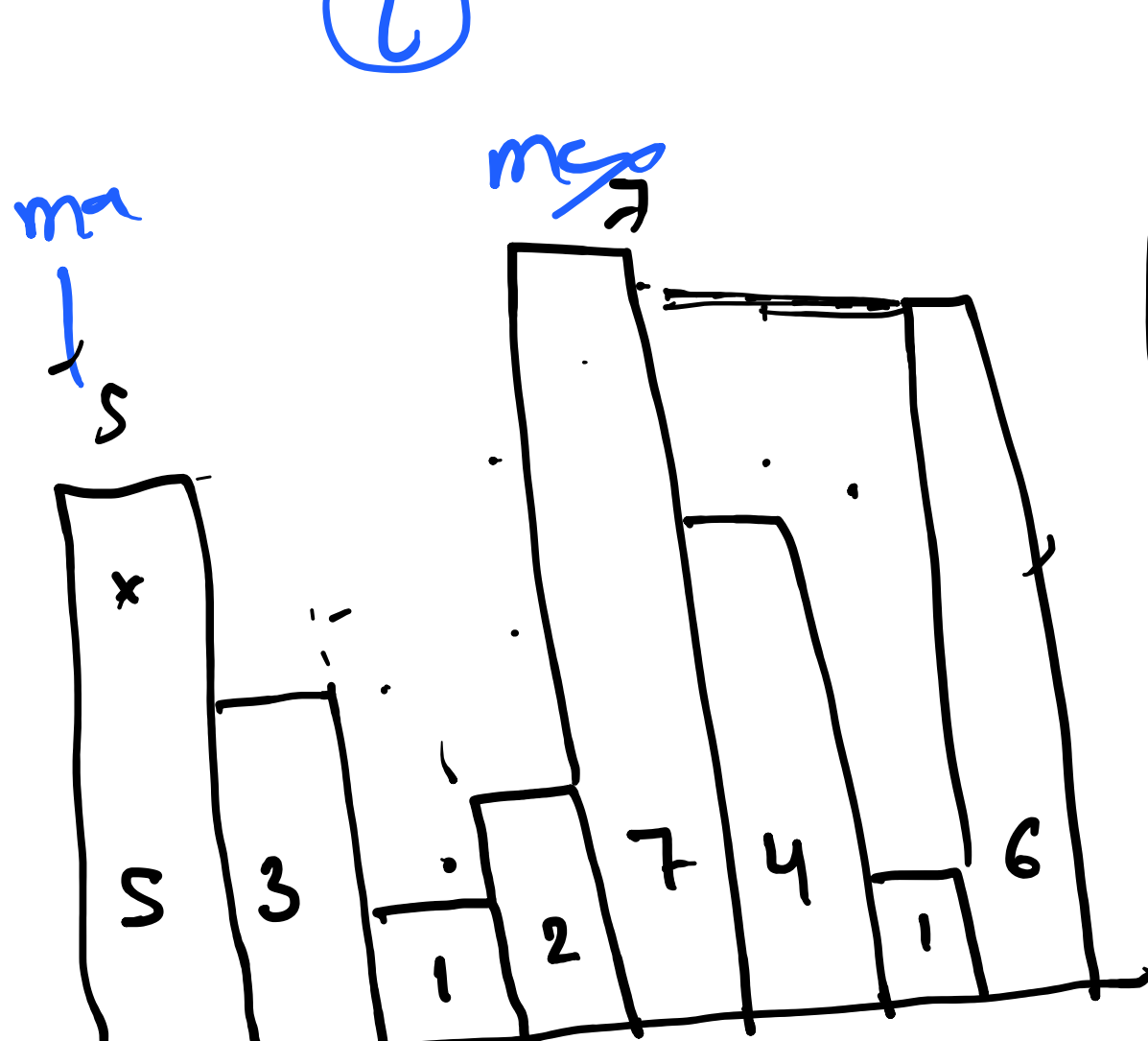
k=3

n/2

n-k=4

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

2x1=2  
4x1=4  
3x1=3  
2x1=2  
5x1=5



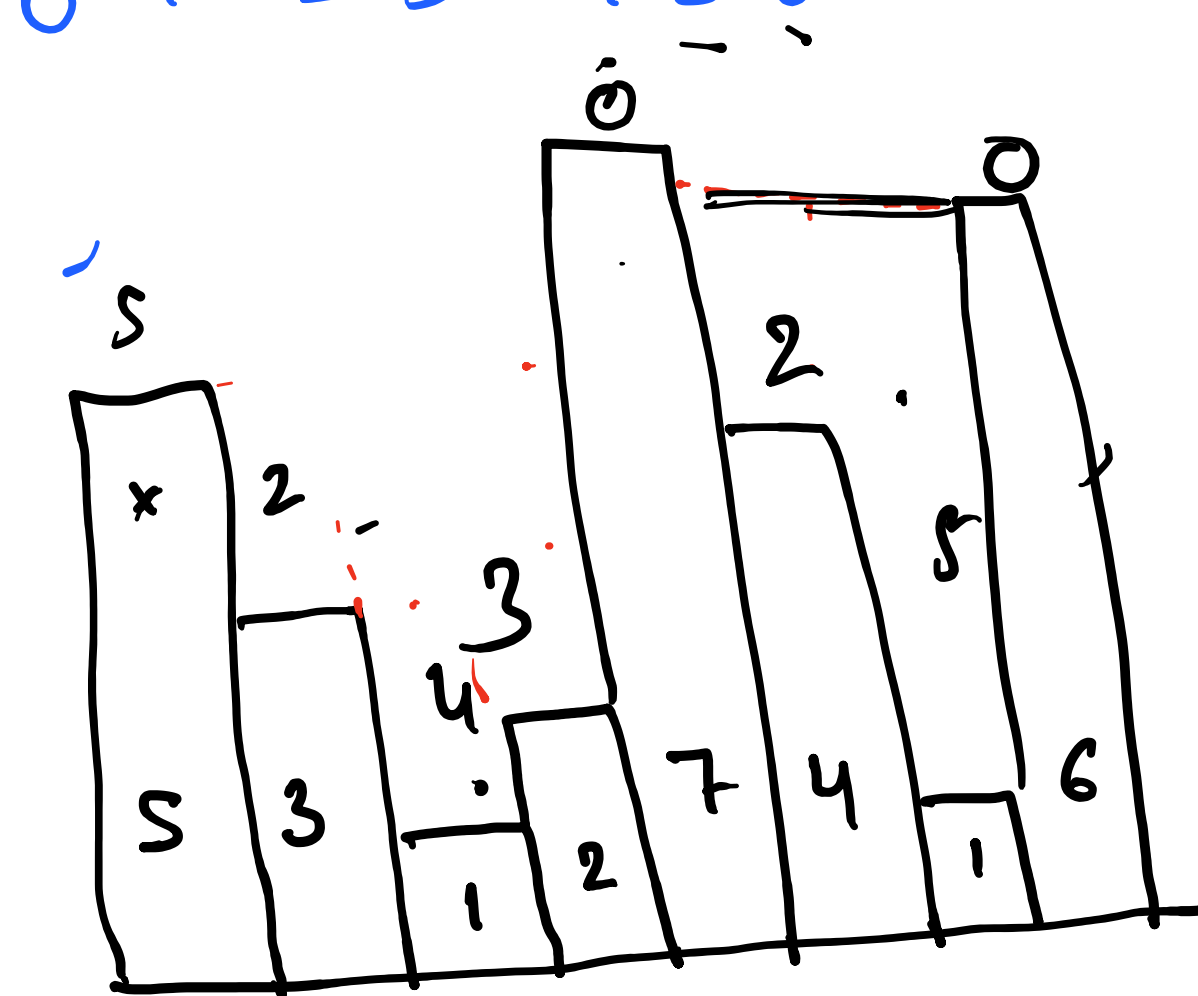
5 - 3 = 2x

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

5 3 1 2 7 4 1 6

i=0 → min(L[0], R[0]) - arr[0] = (5, 7) - 5 = 0  
i=1 → min(L[1], R[1]) - arr[1] = (5, 3) - 3 = 2  
i=2 → min(L[2], R[2]) - arr[2] = (1, 7) - 1 = 4  
i=3 → min(L[3], R[3]) - arr[3] = (3, 1) - 2 = 3  
i=4 → min(L[4], R[4]) - arr[4] = (7, 4) - 7 = 0  
i=5 → min(L[5], R[5]) - arr[5] = (1, 6) - 4 = 2  
i=6 → min(L[6], R[6]) - arr[6] = (3, 1) - 1 = 5  
i=7 → min(L[7], R[7]) - arr[7] = (7, 6) - 6 = 0

7 7 7 7 7 6 6 6



for (i = 0; i < n; i++) {  
sum = sum + min(L[i], R[i]) - arr[i];  
}

7 7 7 7 7 6 6 6

RC[n] = arr[n-1]

i=6 RC[6] = max(RC[7], arr[6])  
i=5 RC[5] = max(RC[6], arr[5])  
i=4 RC[4] = max(RC[5], arr[4])  
i=3 RC[3] = max(RC[4], arr[3])  
i=2 RC[2] = max(RC[3], arr[2])  
i=1 RC[1] = max(RC[2], arr[1])  
i=0 RC[0] = max(RC[1], arr[0])

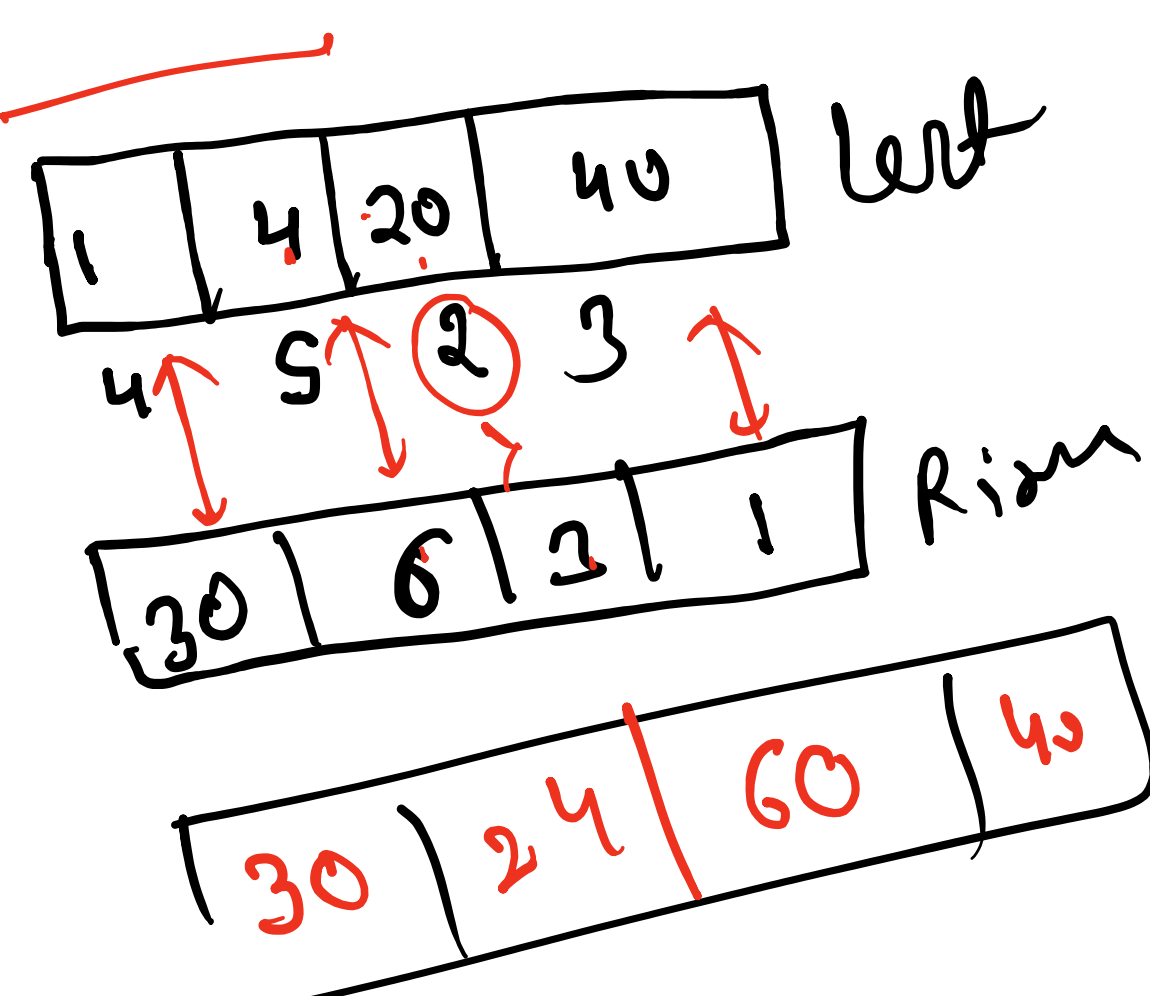
for (i = 1; i < n; i++) {  
left[i] = max(left[i-1], arr[i]);  
}

Left[0] = arr[0]  
Left[1] = max(left[0], arr[1])  
Left[2] = max(left[1], arr[2])  
Left[3] = max(left[2], arr[3])  
Left[4] = max(left[3], arr[4])  
Left[5] = max(left[4], arr[5])  
Left[6] = max(left[5], arr[6])  
Left[7] = max(left[6], arr[7])

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.



4 5 2 3 → 30 24 60 40

4 5 2 3

1 4 20 60

(0) = arr[0] \* arr[1]  
(1) = arr[0] \* arr[2]  
(2) = arr[0] \* arr[3]

RC[0] = RC[1] \* arr[2]  
RC[1] = RC[2] \* arr[3]  
RC[2] = RC[3] \* arr[4]

Left[1] = left[0] \* arr[1]  
Left[2] = left[1] \* arr[2]  
Left[3] = left[2] \* arr[3]