**Before the Brute Force Approach: Rotating the Array by 1 Step**

**Idea:**

Before diving into rotating the array by k steps, it's helpful to understand how to rotate an array by just 1 step to the right. This is a simpler problem that lays the foundation for understanding the brute force approach.

**Steps:**

1. Save the last element of the array.
2. Shift all other elements one position to the right.
3. Place the saved last element in the first position.

**Example:**

Given nums = [1, 2, 3, 4, 5], after rotating by 1 step to the right, the array becomes [5, 1, 2, 3, 4].

```cpp
void rotateByOne(vector<int>& nums) {
    int n = nums.size();
    int last = nums[n - 1];

    for (int i = n - 1; i > 0; i--) {
        nums[i] = nums[i - 1];
    }

    nums[0] = last;
}
```

**Time Complexity: O(n)**

**Space Complexity**: **O(1)**

**1. Brute Force Approach**

**Idea:**

In the brute force approach, we simply rotate the array one step at a time, repeating this process k times.

**Steps:**

1. For each rotation, we move the last element to the front.

2. We repeat this process k times.

```cpp
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        int n = nums.size();
        k = k % n;  // In case k is greater than the size of the array
        for (int i = 0; i < k; i++) {
            int last = nums[n - 1];
            for (int j = n - 1; j > 0; j--) {
                nums[j] = nums[j - 1];
            }
            nums[0] = last;
        }
    }
};
```

**Time Complexity: O(n * k)**

**Space Complexity**: **O(1)**

**2. Better Approach (Using Extra Space)**

**Idea:**

We can create a new array and place each element in its rotated position.

**Steps:**

1. Create a new array of the same size as the original.
2. Place each element in the new array at its correct rotated position.
3. Copy the new array back to the original array.

```cpp
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        int n = nums.size();
        k = k % n;  // In case k is greater than the size of the array
        vector<int> rotated(n);

        for (int i = 0; i < n; i++) {
            rotated[(i + k) % n] = nums[i];
        }

        for (int i = 0; i < n; i++) {
            nums[i] = rotated[i];
        }
    }
};
```

**Time Complexity: O(n)**

**Space Complexity**: **O(n)**

**3. Optimal Approach (In-Place Reversal)**

**Idea:**

We can reverse parts of the array to achieve the rotation in-place with O(1) extra space.

**Steps:**

1. Reverse the entire array.
2. Reverse the first k elements.
3. Reverse the remaining n-k elements.

```cpp
class Solution {
public:
    void reverse(vector<int>& nums, int start, int end) {
        while (start < end) {
            swap(nums[start], nums[end]);
            start++;
            end--;
        }
    }

    void rotate(vector<int>& nums, int k) {
        int n = nums.size();
        k = k % n;  // In case k is greater than the size of the array

        // Reverse the whole array
        reverse(nums, 0, n - 1);
        // Reverse the first k elements
        reverse(nums, 0, k - 1);
        // Reverse the remaining n-k elements
        reverse(nums, k, n - 1);
    }
};
```

**Time Complexity: O(n)**

**Space Complexity**: **O(1)**

# Array

* left rotate the array by one place.

Q: arr[] = {1, 2, 3, 4, 5}

A: arr[] = {2, 3, 4, 5, 1}

Approach:

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

* left rotate an array by D place.

Q: arr[] = {1, 2, 3, 4, 5, 6, 7} d = 2

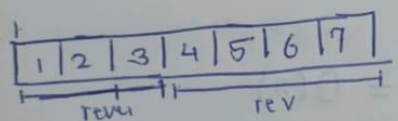
A: arr[] = {3, 4, 5, 6, 7, 1, 2}

Approach 1: Brut force approach

Approach 2:

d = 3

S1:



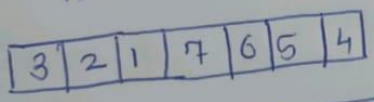

reverse (0, a + d)

revers (a+d, d+n)

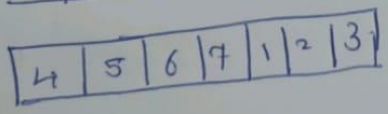
revers (a, a+n).

S2:

S2:




S3:




```
void Reverse (int arr[], int start, int end) {
    while (start <= end) {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
```

**Similar questions to Practice :**

- **GFG -** Rotate by 1