Rotate Array DSA Approaches

Give DSA notes for this question: Given an integer array nums, rotate the array to the right by k steps, where k is non-negative.

2/2

Notes on Rotating an Array to the Right by k Steps

Problem Breakdown

1. **Input:**
   - An integer array, `nums`.
   - A non-negative integer, k.
2. **Output:**
   - The array, rotated k steps to the right.
   - Rotating right means that each element is shifted k positions to the right, with the last elements wrapping around to the beginning.
3. **Constraints:**
   - k can be larger than the length of the array.
   - Handle the operation efficiently in terms of time and space complexity.

Key Observations

1. **Effective Rotations:**
   - If $k > \text{length of nums}(n)$, then $k\%n$ rotations produce the same result. Example: Rotating `[1, 2, 3, 4, 5]` by $k = 7$ is equivalent to $k = 2$.
2. **Final Array Arrangement:**
   - The last $k\%n$ elements move to the front.
   - The first $n - k\%n$ elements shift to the back.

Approaches

1. Brute Force (Nested Loops)

- Rotate the array one step at a time k times.
- **Steps:**
  - For each rotation, take the last element and insert it at the beginning of the array.
  - Shift all other elements to the right by one position.
- **Complexity:**
  - Time: $O(k \cdot n)$, as each rotation takes $O(n)$.
  - Space: $O(1)$, as no extra data structures are used.

---

2. Using a New Array

- Use an auxiliary array to rearrange the elements.
- **Steps:**
  - Create a new array of the same size as `nums`.
  - Place each element from the original array into its new position in the auxiliary array:
    $$\text{newIndex} = (i + k) \% n$$
  - Copy the rearranged array back into the original array.
- **Complexity:**
  - Time: $O(n)$, as we iterate through the array twice.
  - Space: $O(n)$, for the auxiliary array.

---

3. In-Place Reversal (Optimal Solution)

- Rearrange the array without extra space using reversals.
- **Steps:**

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

- **Why It Works:**
  - Reversing the entire array aligns the last $k\%n$ elements at the front.
  - Reversing the first $k\%n$ elements restores their order.
  - Reversing the remaining $n - k\%n$ elements restores their order as well.
- **Complexity:**
  - Time: $O(n)$, as each reversal is $O(n)$.
  - Space: $O(1)$, as no additional space is used.

Code Skeleton (In-Place Reversal)

java

```java
public void rotate(int[] nums, int k) {
    int n = nums.length;
    k = k % n; // Handle cases where k > n

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

private void reverse(int[] nums, int start, int end) {
    while (start < end) {
        int temp = nums[start];
        nums[start] = nums[end];
        nums[end] = temp;
        start++;
        end--;
```

```
    }
}
```

Example Walkthrough

Input:

- $\text{nums} = [1, 2, 3, 4, 5, 6, 7]$
- $k = 3$

Steps (In-Place Reversal):

1. Reverse the entire array:
   $[7, 6, 5, 4, 3, 2, 1]$
2. Reverse the first $k = 3$ elements:
   $[5, 6, 7, 4, 3, 2, 1]$
3. Reverse the remaining $n - k = 4$ elements:
   $[5, 6, 7, 1, 2, 3, 4]$

Output:

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

Additional Edge Cases

1. **Empty Array or Single Element:**
   - No rotation is needed.
2. $k = 0$**:**
   - The array remains unchanged.
3. $k \geq n$**:**
   - Reduce k using $k \% n$.

Practice Questions

- Rotate the array to the left by k steps.
- Given two arrays, determine if one is a rotation of the other.