Date: / /
Page:

✓ Array Rotation

1) left rotation - each element is shifted one poistion to the left, and the first element moves to the end.

2) rigth rotation - elements are shifted to the rigth and the last element moves to the fromt.

Ex. original - [1, 2, 3, 4, 5]
rigth rotation - [5, 1, 2, 3, 4]
left rotation - [2, 3, 4, 5, 1]

* Methods

1) Brute force (for left & rigth rotation)
2) In place reversal algo. (optimal)

✓ implenting using brute force method

code

```
def left_rotate(arr, k):
    n = len(arr)
    k = k % n  # in case k>n.
    return arr[k:] + arr[:k]
```

modulo based indexing

Ex. len = 7 & k = 8
so, k becomes 1. and gives one complete rotation +1 (i.e 1)

✓ $O(n-k+k) = O(n)$

arr[k:] & arr[:k]

1) time complexity - $O(n-k)$ & $O(k)$ & concatination - $O(n)$ — $O(n)$

2) space complexity - new list created for concatination which has same num ele.

so, $O(n)$

✓ Implementing using Inplace reversal

→ works for both left & right reversal

1) time complexity - $O(n)$

2) space complexity - $O(1)$

* Steps.

1) reverse first (k) ele
2) reverse remaining (n-k) element
3) reverse the whole array.

✓ code

```
def reverse(arr, start, end):
    while(start < end):
        arr[start], arr[end] = arr[end],
                                arr[start].
        start += 1
        end -= 1.
```

left kdun rotate

for left rotation

```
def left_rotati(arr, k):
    n = len(arr)
    k %= n
    rererse(arr, 0, k-1)  # first half
    rererse(arr, k, n-1)  # second half
    rererse(arr, 0, n-1)  # complete
```

let,

arr = [1, 2, 3, 4, 5]

if k = 2, rotate till -1 i.e n-1 from 0

1	2	3	4	5	← ele
0	1	2	3	4	← index.

be, k = 2 # first two element at back

rotate k, n-1

3	4	5	1	2	← ele
2	3	4	0	1	← old index
0	1	2	3	4	← new index

rigth kadun rotate

for Rigth rotation

```
def rigth_rotate(arr, k):
    n = len(arr)
    k %= n
```

~~reverse(arr, 0, k-1)~~
~~reverse(arr, k, n-1)~~
~~reverse(arr, 0, n-1)~~ X logic

reverse(arr, 0, n-1) # complete
reverse(arr, 0, k-1) # 1st half
reverse(arr, k, n-1) # 2nd half.

✓ **steps.**

1) rotate complete
2) rotate 0, to k-1
3) rotate k, to n-1

Note: don't change step order.

✓ **Very IMP Note**

1) passing array or list like
arr[:] → this makes the non-referencial copy of array and uses it so making changes to copy doesn't affect original.

2) Inplace reversal means reversal without creating extra space.
making execution faster in large arrays.

✓ it helps you loop back of an array once you reach & the end

✓ modulo based indexing

→ using the % modulo oprator to wrap & around indiecies when they go out of bonds.

Ex. arr = [10, 20, 30]
n = len(arr)

```
For i in Range(10):
    print(arr[i%n])
# aviods index out of range error.
```

Use cases

1) Circuler Queues.
2) rotation (left / rigth)
3) hashing (hash table index = hash (key) % size)
4) Round-Robin algo.
5) Handling over flow in array.

✓ Prefrebaly use Binary search on Rotated array

✓ **rigth** rotation using brute force

```
def rigth_rotate(arr, k):
        n = len(arr)
        k = k % n
        return arr[-k:] + arr[:-k]
```

1) time complexity - O(n)

2) space com. - O(n)