Wednesday, 27 March 2024 10:29 PM

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

k = 3 → sub arr starting at idx 3 till the right

output → [4, 5, 6, 1, 2, 3]

Approach 1 → Shift arr one step at a time.

In each iteration, shift the array by one step.

k = 1 → 1 iteration

k = 2 → 2 iterations

k = 3 → 3 iterations

k = n-1

TC → $O(n^2)$

k = n

k = n%n = 0

if (k < n) {

k = k + arr.length

}

k = k % arr.length

if (k ≥ n) {

k = k % arr.length;

}

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

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

Approach 2

sub arr n-k, k → n

arr → [1, 2, 3, 4, 5, 6, 7]

reverse whole array (k, n-1) → [7, 6, 5, 4, 3, 2, 1]

reverse first (0, k-1) → [4, 5, 6, 7, 1, 2, 3]

reverse second (k, n-1)

Algorithm → reverse whole array, then reverse first part, then second part.

Helper fn to reverse.

class Solution {

public void rotate(int[] nums, int k) {

int n = nums.length;

k = k % n;

reverse(nums, 0, n-1);

reverse(nums, 0, k-1);

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--;

}

}

}

class Node {

int data;

Node next;

Node(int d) {

this.data = d;

}

}

Node head;

void printList() {

Node temp = head;

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

}

public static void main(String[] args) {

LinkedList ll = new LinkedList();

ll.head = new Node(1);

Node second = new Node(2);

Node third = new Node(3);

ll.head.next = second;

second.next = third;

ll.printList();

}

}

head → 1 → 2 → 3 → null

temp → 1 → 2 → 3 → null

temp → 2 → 3 → null

temp → 3 → null

temp → null

Node temp = head;

temp = temp.next;

temp = temp.next;

temp = temp.next;

temp = null

temp = temp.next;

temp = null

temp = temp.next;

temp = null