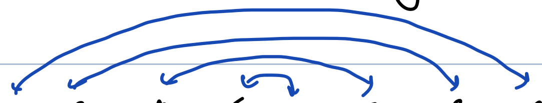
Todays Content

1. Rotate arr[] by k times
2. Vector pass by value & pass by reference.
3. Count no:f distinct elements

Q: Given an arr[], l & r reverse entire array from l..r



Ex: ar[12]

l=2 r=9

	0	1	2	3	4	5	6	7	8	9	10	11	
{	10	20	~~30~~	~~40~~	~~50~~	~~60~~	~~70~~	~~80~~	~~90~~	~~100~~	110	120	}
			100	90	80	70	60	50	40	30			
			P_1 →	P_1 →	P_1 →	P_1	P_2	← P_2	← P_2	← P_2			

Idea:

	0	1	2	3	4	5	6	7	8	9	10	11	
{	10	20	100	90	80	70	60	50	40	30	110	120	}

P_1	<	P_2	
2	<	9	swap(ar[P1], ar[P2]); P1++; P2--;
3	<	8	swap(ar[P1], ar[P2]); P1++; P2--;
4	<	7	swap(ar[P1], ar[P2]); P1++; P2--;
5	<	6	swap(ar[P1], ar[P2]); P1++; P2--;
6	<	5	P1> P2: Stop ar[P1... P2] is reversed.

```
void reverseRange(int ar[], int l, int r){
    int P1 = l, P2 = r;
    while(P1 < P2){
        //swap ar[P1] & ar[P2]
        int tmp = ar[P1];
        ar[P1] = ar[P2];
        ar[P2] = tmp;
        P1++; P2--;
    }
}
```

Iteration: WorstCase = Entire Array = N/2

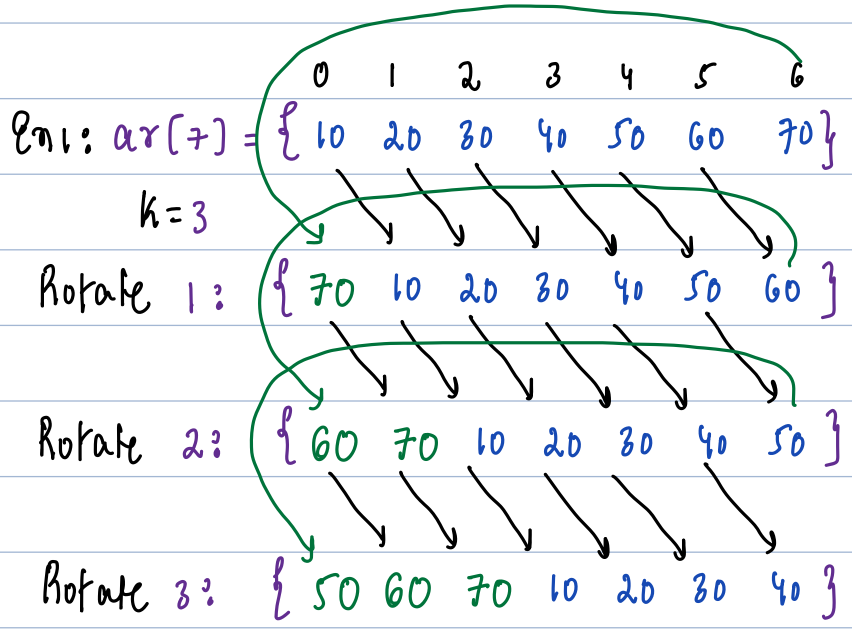
TC: O(N) SC: O(1)

Q: Given an arr[] rotate it by k times by Right → Left

Constraints:

$1 <= N <= 10^5$

$1 <= K <= 10^5$



0 1 2 3 4 5 6

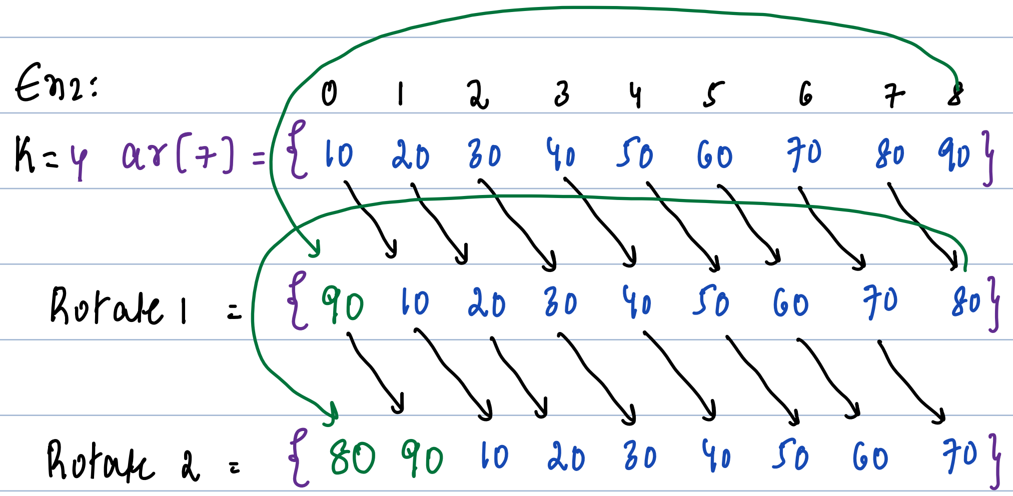
Ex1: ar[7] = { 10 20 30 40 50 60 70 }

k=3

Rotate 1: { 70 10 20 30 40 50 60 }

Rotate 2: { 60 70 10 20 30 40 50 }

Rotate 3: { 50 60 70 10 20 30 40 }



Ex2:

0 1 2 3 4 5 6 7 8

K=4 ar[7] = { 10 20 30 40 50 60 70 80 90 }

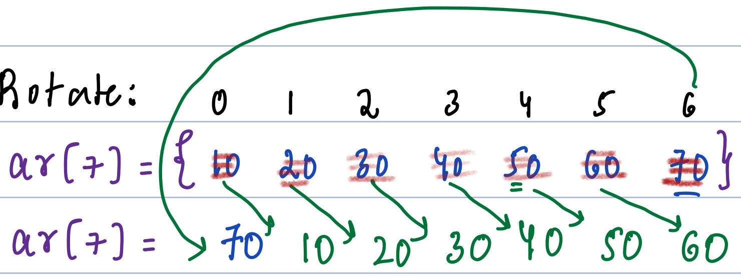
Rotate 1 = { 90 10 20 30 40 50 60 70 80 }

Rotate 2 = { 80 90 10 20 30 40 50 60 70 }

Idea: For k times: Estimated TC: O(k*N)

Rotate arr[] right to left.

Single Rotate:



0 1 2 3 4 5 6
ar[7] = { ~~10~~ ~~20~~ ~~30~~ ~~40~~ ~~50~~ ~~60~~ ~~70~~ }
ar[7] = 70 10 20 30 40 50 60

```
int tmp = ar[6]; //70 ✓
Shifting ✓
     i       i-1
   ar[6] = ar[5]
   ar[5] = ar[4]
   ar[4] = ar[3]
   ar[3] = ar[2]
   ar[2] = ar[1]
   ar[1] = ar[0]
ar[0] = tmp;
```

```
void rotatek(int ar[], int N, int k){
    for(int l=0; l<k; l++){
        int tmp = ar[N-1]; // tmp: last element
        for(int i = N-1; i>=1; i--){
        }   ar[i] = ar[i-1];
        ar[0] = tmp;
    }
}
```

Innerloop = NK Outerloop = K

TC: O(N*k) SC: O(1)

$1 <= N <= 10^5$

$1 <= k <= 10^5$

$O(10^5 * 10^5) = 10^{10} >> 10^8$ TLE

Optimized Approach

Ex1: K=3

	0	1	2	3	4	5	6
ar[7] =	10	20	30	40	50	60	70
Rotate 1 =	70	10	20	30	40	50	60
Rotate 2 =	60	70	10	20	30	40	50
Rotate 3 =	50	60	70	10	20	30	40

K=4

	0	1	2	3	4	5	6	7	8
ar[7] =	10	20	30	40	50	60	70	80	90
Rotate 1 =	90	10	20	30	40	50	60	70	80
Rotate 2 =	80	90	10	20	30	40	50	60	70
Rotate 3 =	70	80	90	10	20	30	40	50	60
Rotate 4 =	60	70	80	90	10	20	30	40	50

obs: Rotate arr[N] by k times

1. last k ele will shift to first
2. Remaining ele will shift to back

Ex4:

k=5

	0	1	2	3	4	5	6	7	8	9	10	11	12
ar[13] =	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}

Step1:

Reverse entire arr[]

0	1	2	3	4	5	6	7	8	9	10	11	12
a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0

Step2: Reverse k elements

Step3: Reverse rem ele

After Rotation =

0	1	2	3	4	5	6	7	8	9	10	11	12
a_8	a_9	a_{10}	a_{11}	a_{12}	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Idea: Given arr[N] & rotate k times

1. Reverse entire arr[] : reverseRange(arr, 0, N-1) : N/2
2. Reverse first k ele : reverseRange(arr, 0, k-1) : k/2
3. Reverse remaining ele : reverseRange(arr, k, N-1) : (N-k)/2

Iterations : N

TC : O(N)

```
void reverseRange(int ar[], int l, int r){
    int p1 = l, p2 = r;
    while(p1 < p2){
        int tmp = ar[p1];
        ar[p1] = ar[p2];
        ar[p2] = tmp;
    } p1++; p2--;
}
```

```
void rotateK(int ar[], int N, int k){
    K = K%N; // Edge Cases
    reverseRange(ar, 0, N-1)  →  reverseRange(ar, 0, 3) ✓
    reverseRange(ar, 0, k-1)  →  reverseRange(ar, 0, 5) ? Error.*
}   reverseRange(ar, k, N-1)
```

Ex: N=4 ar[4] = {10 20 30 40} k=6 ?
(indices: 0 1 2 3)

$P_1 = 0$, $P_2 = 5$

swap ar[0] & ar[5] // RTE.

Observation

	0	1	2	3
Ex: ar[4] =	10	20	30	40
rotate 0 =	10	20	30	40
rotate 1 =	40	10	20	30
rotate 2 =	30	40	10	20
rotate 3 =	20	30	40	10
rotate 4 =	10	20	30	40
rotate 5 =	40	10	20	30
rotate 6 =	30	40	10	20
rotate 7 =	20	30	40	10
rotate 8 =	10	20	30	40

Rotation observation

K:

0 → 4 → 8 → 12 → 16 → 20 . .

1 → 5 → 9 → 13 → 17 → 21 . .

2 → 6 → 10 → 14 → 18 → 22 → 26

3 → 7 → 11 → 15 → 19 → 23 → 27 → 31 → 35

K	N=4 ≠ K%4 ⇒ In general K%N
20	0
17	1
26	2
30	2
35	3

Con: Rotating k is same as Rotating K%N

```
void rotateK(int ar[], int N, int K){
    K = K%N; // Edge Cases
    reverseRange(arr, 0, N-1)  →  reverseRange(arr, 0, 3) ✓
    reverseRange(arr, 0, K-1)  →  reverseRange(arr, 0, 1) ✓
    reverseRange(arr, K, N-1)  →  reverseRange(arr, 2, 3) ✓
}
```

```
                          0  1  2  3
Ex: N=4  ar[4] = { 10 20 30 40 }  K=6 ⇒ 6%4=2
```