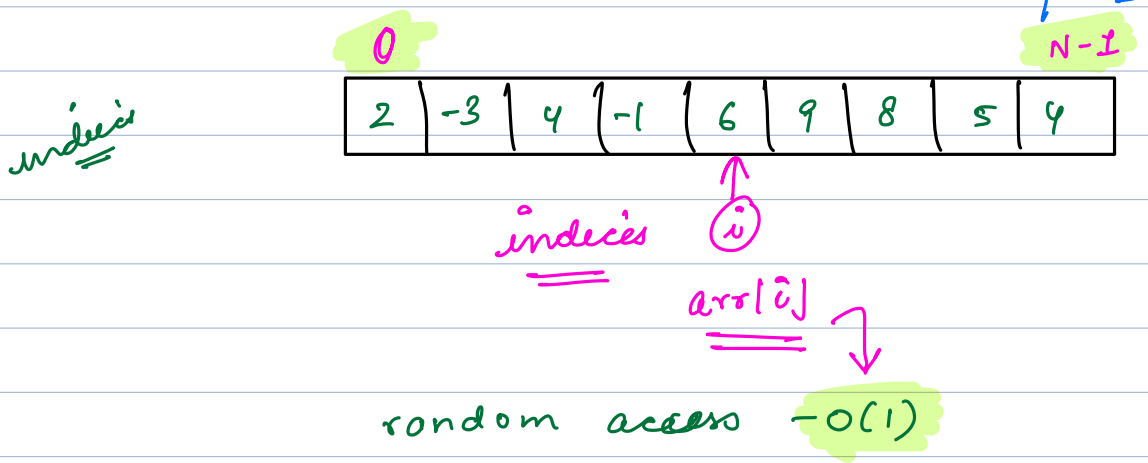
Arrays ? :-

list of homogenous items

- contiguous
- same type

```
int arr[3];
```

→ size is fixed



arr[i]

random access - O(1)

Traversal

```
for( i=0; i<n; i++)
    print(arr[i]);
```

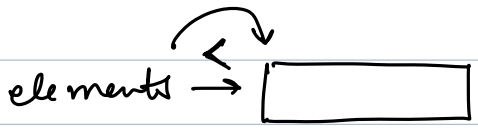
T.C: O(N)

Q Given an array of size N. Find count of elements which has atleast one greater element than itself.

-3 -2 6 8 4 8 5 ans = 5

2 5 1 4 8 0 8 1 3 8

ans = 7



{ maximum / greatest element don't have greater elements than itself

ans = N − count of max elements

size of array

get the freq → count of max elements

~~sort the array $\frac{1}{2}$ n log n~~

1) Find max element
2) find freq of max element → (x)
3) ans = N - x

```
int mx = -∞(INT_MIN) / arr[0]
for(int i=1; i<n; i++)
    mx = max(mx, arr[i]);
```

N

```
int cnt=0;
for(i=0; i<n; i++)
{
    if(arr[i]==mx)
        cnt++;
}
```

N

2N

T.C = O(N)
S.C = O(1)

we can do it in a single loop!

```
ans = n-cnt;
```

Q ≡ given an array of N elements. Find count of pairs (i, j) where i & j are indices such that $arr[i] + arr[j] = K$.

$i \neq j$

↳ given $i < j$

0	1	2	3	4	5	6
3	-2	1	4	3	6	8

$K = 10$

ans = 1

0	1	2	3	4	5	6	7	8	9
3	5	2	1	-3	7	8	15	6	13

$K = 10$

ans = 3

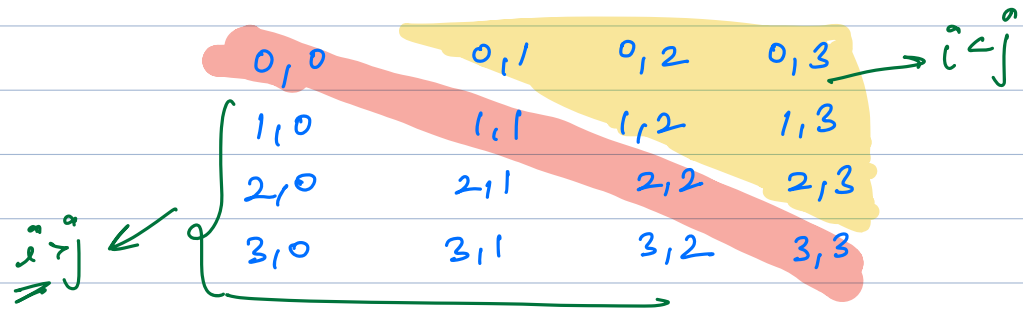
$K = 20$

0	1	2	3	4	5	6	7	8
2	7	3	14	6	1	0	10	14

ans = 2

Basic :- consider all possible valid pairs

$N = 4$



```
cnt=0
for( i=0; i<n; i++)
{
    for(j= i+1; j<n; j++)
    {
        if( arr[i]+arr[j]==K)
            cnt++;
    }
}
```

i	j	
0	1 → N-1	N-1
1	2 → N-1	N-2
2	3 → N-1	N-3
⋮		
N-2	N-1 → N-1	1
N-1	✗	0

$x * \frac{(n+1)}{2}$ ⟷ $x = n-1$

$\frac{(n-1)*(n)}{2} \Rightarrow \frac{n^2-n}{2} \Rightarrow O(n^2)$ T.C

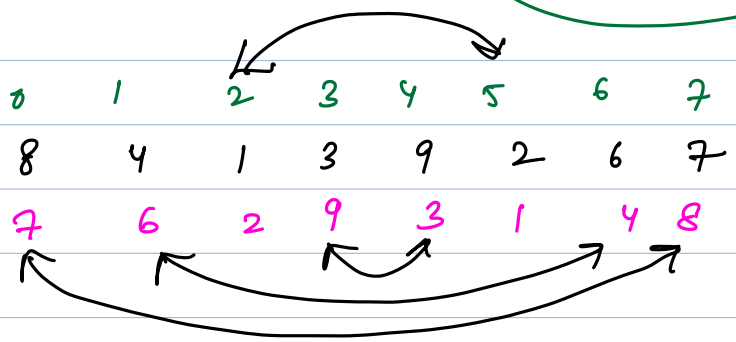
$O(1)$ S.C

HW: lower triangle

Q) given an array of size N.

Reverse the array without using extra space.

O(1)



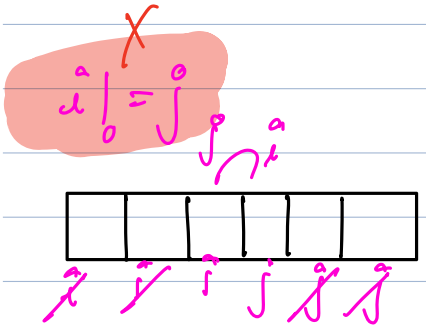
0th → 7th
1st → 6th
2nd → 5th
3rd → 4th
4th → 3rd
5th → 2nd
6th → 1st
7th → 0th

iterations = N/2

S.C: O(1)

T.C: O(N)

~~O(N/2)~~




```
void reverse ( int arr[], int n)
{
    i = 0, j = n-1;
    while ( i < j )
    { swap(arr[i], arr[j]);
      i++;
      j--;
    }
}
```

we can do it without j

#

2 points → (start, end) → reverse from start to end

0	1	2	3	4	5	6	7	8
4	6	1	3	9	2	8	7	10
4	6	1	8	2	9	3	7	10

3 → 6

```
void reverse ( int arr[] , int n , int start, int end)
{
    i = start ; j = end ;
    while ( i < j )
    { swap( arr[i], arr[j]);
      i++;
      j--;
    }
}
```

⇒ worst case T.C → O(N)

O(end - start + 1)

Q Given an array of size N. Rotate given array K times clockwise.

[constant space]

K < N

	0	1	2	3	4	5	6
	-1	2	3	6	5	4	8
K=1	8	-1	2	3	6	5	4
K=2	4	8	-1	2	3	6	5
K=3	5	4	8	-1	2	3	6

Obsf:- After rotating K times, last K elements will come at start of the array

-1 2 3 6 5 4 8 K=3

8 4 5 6 3 2 -1

reverse + reverse

5 4 8

S.C : O(1)

① reverse (0, N-1) ← n/2

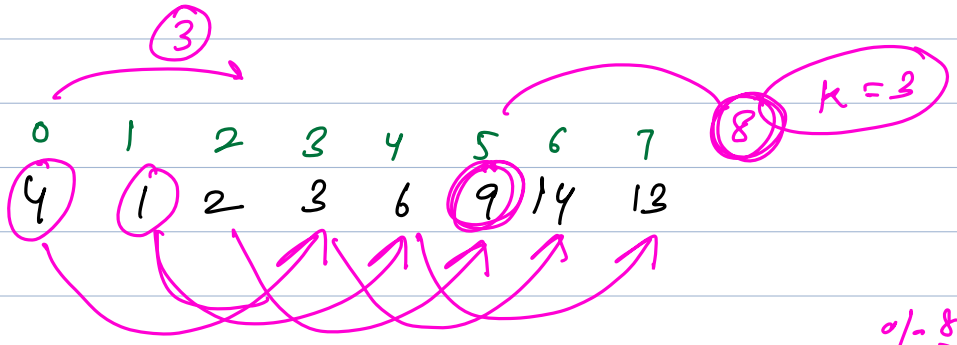
② reverse (0, K-1) ← K/2

③ reverse (K, N-1) ← n-k

2

N

n/2 + K/2 + n-k/2



8 %8 → 0
9 %8 → 1
10 %8 → 2

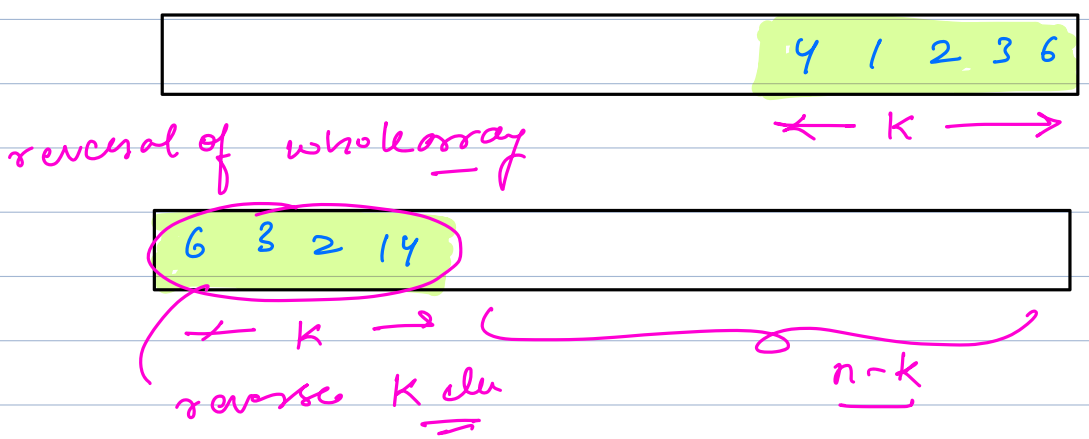
$(i + k) \% N$

if you directly swap → go wrong → extra space

0 → 3
1 → 4
2 → 5
3 → 6
4 → 7
5 → 8 → 0
6 → 9 → 1
7 → 10 → 2

T.C: O(N)
S.C: O(N)

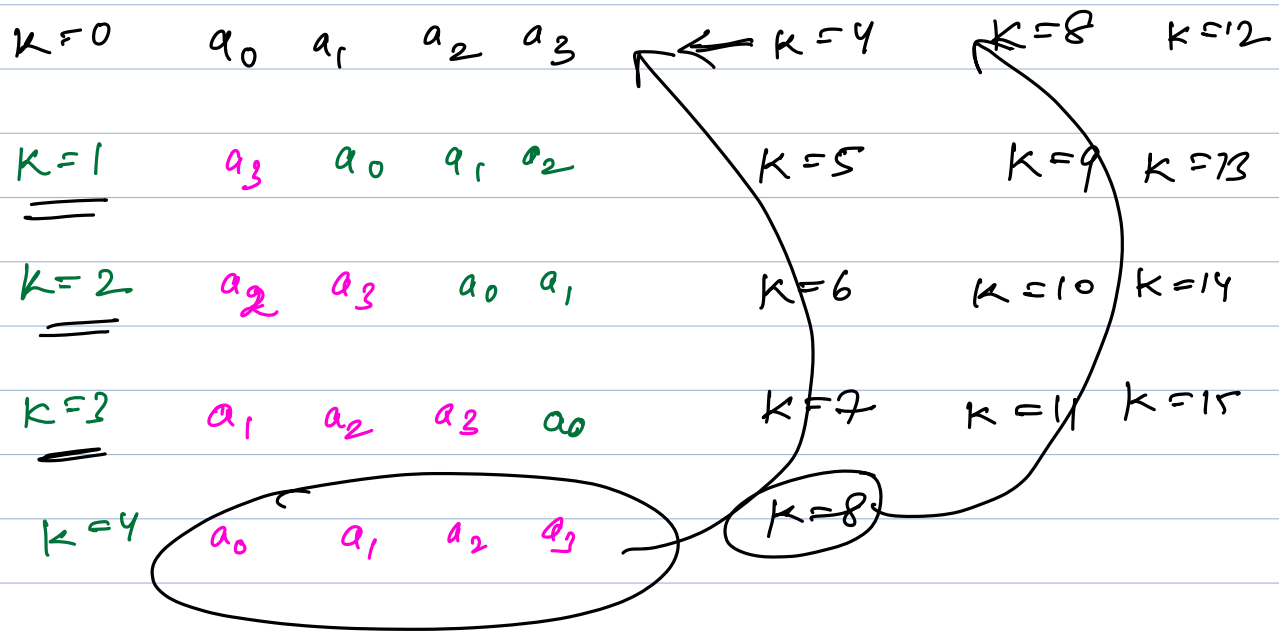
K rotation



last K elements to the beginning
↓
reverse

T.C: O(N)
S.C: O(1)

K > N



extra part

K > N

↳ K = K % N

K = N * x + K % N

N = 10

K = 2000

K = 1

2001

Dynamic Arrays

size is fixed ≡ static

C++	java	python	C#	Js/Ruby	C →
vectors	ArrayList	list	list ArrayList	array	malloc
	↓				
	.get				

static array

Behind the scenes → resizing

append() / add() / insert() / push_back()