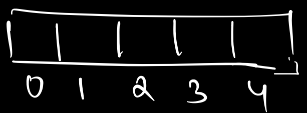
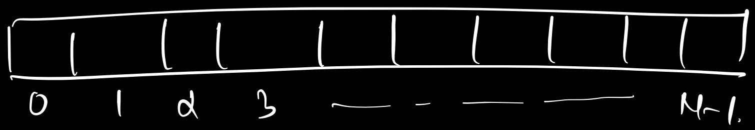
int arr[5] →



↓
integer array with 5 elements

int arr[N] →



↓
integer array with N elements

indexes => 0 to N-1

// print an array.

```
for(i=0; i<N; i++){
        print(arr[i])
}
```

] => O(N)

→ accessing the element at index i => O(1)

* accessing i^{th} index element => O(1)
arr[i]

Q1. Given N array elements, count no. of elements having at atleast 1 element, greater than itself? {no library fn}

ex => arr[7] => { -3 -2 6 8 4 8 5 }
✓ ✓ ✓ ✗ ✓ ✗ ✓

O/p = 5

arr[8] => { 2 3 10 7 3 2 10 8 }
✓ ✓ ✗ ✓ ✓ ✓ ✗ ✓

o/p = 6.

arr[10] => { 2, 5, 1, 4, 8, 0, 8, 13, 8, 9 }
✓ ✓ ✓ ✓ ✓ ✓ ✓ ✗ ✓ ✓

o/p = 9.

pseudo

```
c=0
for( i=0; i<N; i++) {
    e = arr[i]
    for( j=0; j<N; j++) {
        if( arr[j] > e) {
            c++;
            break;
        }
    }
}
return c;
```

TC => $O(N^2)$
SC => $O(1)$

Observation

* for maxm element, there can't be any element greater than itself. [count of maxm element won't be part of ans]

- max^{m} element is greater than all elements in the array, except itself
- for every element except max^{m}, we will definitely something bigger than the element.

ans = count of all elements of array except the count of max^{m} elements

ans = [N − count of max^{m} element]

1) N is known

11) count of max m → 1) max^{m} element

2) freq. or count of max^{m} ele.

```
              INT_MIN
maxV = 0  →  arr[0]
for( i=0; i<N; i++){
        if( maxV < arr[i])
                maxV = arr[i]
}
count=0
for ( i=0; i<N; i++){
        if ( arr[i]== maxV)
                count++
}
```

maxV = 0.

[-3 -2 -1 -1]

maxm = -1

maxV = 6

[6 3 2 2]

largest = 6

```
return N-count;
}
```

$$\boxed{\begin{array}{l} TC \Rightarrow O(2N) \cong O(N) \\ SC \Rightarrow O(1) \end{array}}$$

How

TO DO → Change the above code, so that it works with 1 for loop <u>only</u> ⇒ TC ⇒ O(N)

SC ⇒ O(1)

Q2. Given N array elements, check if there exists a pair (i,j) such that arr[i] + arr[j] == k, & i != j.

Note:- i & j are index values, k is the given sum.

arr[] ⇒	3	-2	1	4	3	6	8
	0	1	2	3	4	5	6

k = 10

o/p ⇒ true

$\left.\begin{array}{l} i = 3 \\ j = 5 \end{array}\right\}$ 4+6 = 10

arr[] =	2	4	-3	7
	0	1	2	3

k = 5

o/p = false

arr[] =	2	4	-3	7
	0	1	2	3

k = 8

o/p : false.

~~4+4 =~~ 8

Brutefore

check all pairs

N = 5 => { 0 1 2 3 4 }

(0,0)	(1,0)	(2,0)	(3,0)	(4,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)

pseudo

```
for( i=0; i<N; i++){
        for(j=0; j<N; j++){
            if ( i != j) {
                    if ( arr[i] + arr[j] == k)
                            return true
            }
        }
}
```

return false.

iterations => N^2

TC => $O(N^2)$

SC => $O(1)$

Optimised

```
for( i=0, i<N, i++ ){
        for( j=i+1; j<N; j++){
            if ( arr[i] + arr[j] == k)
                    return true
        }
}
```

arr => 3 -2 1 4 3 6 8

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

k = 10

i	j	sum
0	1	1
0	2	4
0	3	7
0	4	6
0	5	9
0	6	11
1	2	-1
1	3	2
1	4	1
1	5	4
1	6	6

i	j	sum
2	3	5
2	4	4
2	5	7
2	6	9
3	4	7
3	5	10

$N-1 + N-2 + N-3 + \ldots + 3+2+1+0$

$\Rightarrow 1+2+3+ \ldots + N-2 + N-1$

$\Rightarrow \dfrac{(N-1)(N-1+1)}{2}$ $\quad \left\{ \dfrac{N(N+1)}{2} \right.$

$\Rightarrow$ iterations $= \dfrac{N(N-1)}{2}$

TC ⇒ $O(N^2)$

SC ⇒ $O(1)$

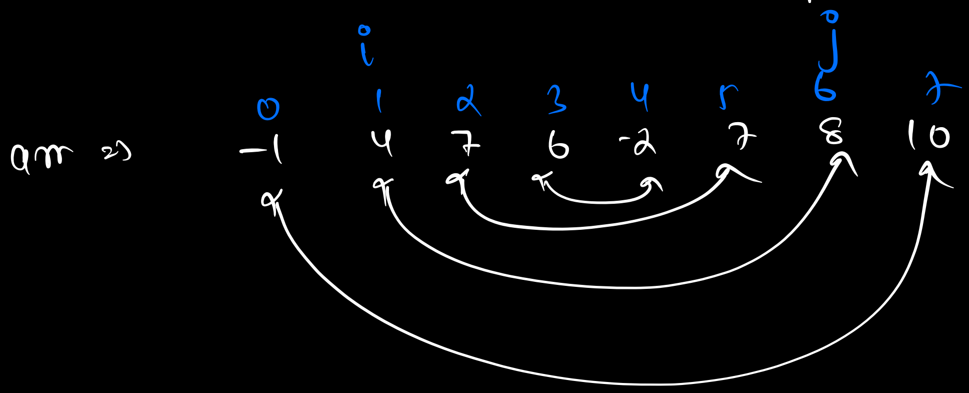
i	j	iterations
0	[1, N-1]	N-1
1	[2, N-1]	N-2
2	[3, N-1]	N-3
3	[4, N-1]	N-4
⋮		⋮
N	[N, N-1]	0

Q3. Given an array, reverse entire array | SC :- O(1)

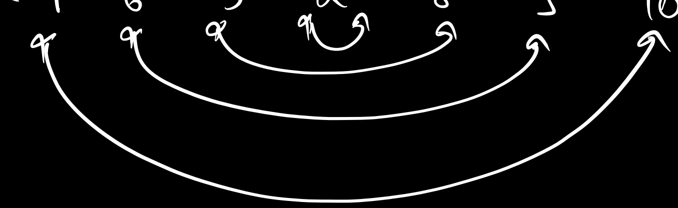
arr ⇒ -1 4 7 6 -2 7 8 10

rev. ⇒ 10 8 7 -2 6 7 4 -1

Since, SC O(1) → reverse the array in place.



arr => -1 6 3 2 8 9 10



	i	j		
	0	7		→ swap
+1	1	6	-1	→ swap
+1	2	5	-1	→ swap
	3	4		→ swap
	4	3		

pseudo

```
    reverseArray(arr, N) {
        i = 0
        j = N-1
    while ( i <= j) {
        swap(arr[i], arr[j])
        i++
        j--
    }.
```

Implement your own swap.

TC => O(N)

SC => O(1)

Q4. Given N array elements, & Si, Ei reverse array from Si to Ei, note Si ≤ Ei.

	0	1	2	3	4	5	6	7	8
arr =>	-3	4	2	8	7	9	6	2	10

Si = 3

Ei = 7.

	0	1	2	3	4	5	6	7	8
op =>	-3	4	2	2	6	9	7	8	10.

```
reversePart(arr, N, Si, Ei){
    i = Si
    j = Ei
  while( i <= j ){
     swap(arr[i], arr[j])
     i++
     j--
  }.
```

TC => O(N)

SC => O(1)

Q5. Given N array elements, rotate array from last to first by k times.

ex => arr[] = 3 -2 1 4 6 9 8

k = 3.

original → 3 -2 1 4 6 9 8

k = 1 ⇒ 8 3 -2 1 4 6 9

k = 2 ⇒ 9 8 3 -2 1 4 6

k = 3 ⇒ 6 9 8 3 -2 1 4 ⇒ o/p.

arr[9] ⇒ $\underbrace{4\ \ 1\ \ 6\ \ 9\ \ 2}\ \ \underbrace{14\ \ 7\ \ 8\ \ 3}$

k = 4. ⇒ $\underbrace{14\ \ 7\ \ 8\ \ 3}\ \ \underbrace{4\ \ 1\ \ 6\ \ 9\ \ 2}$

arr[10] ⇒ $\underbrace{-2\ \ 3\ \ 1\ \ 4\ \ 6\ \ 2\ \ 8}\ \ \underbrace{7\ \ 9\ \ 3}$

k = 3 ⇒ $\underbrace{7\ \ 9\ \ 3}\ \ \underbrace{-2\ \ 3\ \ 1\ \ 4\ \ 6\ \ 2\ \ 8}$.

arr[13] ⇒ $a_0\ \ a_1\ \ a_2\ \ a_3\ \ a_4\ \ a_5\ \ a_6\ \ a_7\ \ a_8\ \ a_9\ \ a_{10}\ \ a_{11}\ \ a_{12}$

k = 5 ⇒ $\underbrace{a_8\ \ a_9\ \ a_{10}\ \ a_{11}\ \ a_{12}}\ \ \underbrace{a_0\ \ a_1\ \ a_2\ \ a_3\ \ a_4\ \ a_5\ \ a_6\ \ a_7}$

reverse of ⇒ original array

$\underbrace{a_{12}\ \ a_{11}\ \ a_{10}\ \ a_9\ \ a_8}\ \ \underbrace{a_7\ \ a_6\ \ a_5\ \ a_4\ \ a_3\ \ a_2\ \ a_1\ \ a_0}$

k = k%N ↴

pseudo

```
1) reverse the entire array → reversePart(arr, N, 0, N-1)
2) reverse the first k element → reversePart(arr, N, 0, k-1)
3) reverse the last n-k elements → reversePart(arr, N, k, N-1)
```

$TC \Rightarrow O(N + (k) + (N-k)) = O(2N) \simeq O(N)$

$SC \Rightarrow O(1)$

reversePart(arr, N, 0, K−1) $K > N$

$k-1 > N-1$

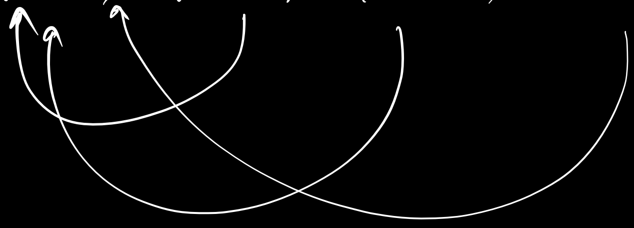
arr[5] K=0	=	a_0	a_1	a_2	a_3	a_4
K=1	⇒	a_4	a_0	a_1	a_2	a_3
k=2	⇒	a_3	a_4	a_0	a_1	a_2
k=3	⇒	a_2	a_3	a_4	a_0	a_1
k=4	⇒	a_1	a_2	a_3	a_4	a_0
k=5	⇒	a_0	a_1	a_2	a_3	a_4
k=6	⇒	a_4	a_0	a_1	a_2	a_3
k=7	⇒	a_3	a_4	a_0	a_1	a_2
k=8	⇒	a_2	a_3	a_4	a_0	a_1

TODO

rotate → first to last

Q) = k=1, k=6, k=11, k=16 _ _ _

N=5



k%N ⇒ k%5

1%5 = 1 21%5 = 1.

6%5 = 1

11%5 = 1

16%5 = 1

int arr[5] → 5

int arr[N] → N

] fixed length.

: dynamic arrays

arrays with dynamic length

C++	→	vector
Java	→	arraylist
Python	→	list
C#	→	arraylist / List
Js	→	array

C → change to C++/ Java.

list.size()

list insert(2)

list.get(0) ← index

O(1)

Doubts

1 6 3 4

k = 2

3 4 1 6.