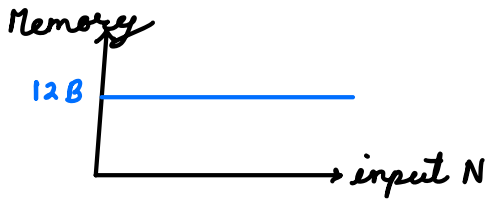
Space Complexity

TC = O(#iterations)

Rate of growth of space wrt input.

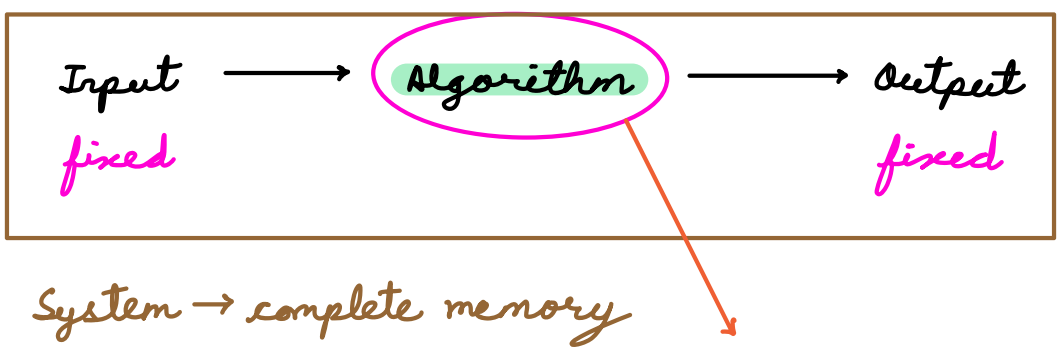
int → 4 Bytes long → 8 byte

1)
```
{ // N → input
  int x = N ;      ← 4B
  long y = x * x ; ← 8B
}
```
12 Bytes SC = O(1)



2)
```
{ // N → input
  int arr[10] ← 10 integers ← 10*4B
  int x ← 4B
  int y ← 4B
  long z ← 8B
  int a[] = new int[N] ← N*4B
}
```
Total = 40 + 4 + 4 + 8 + 4N = (56 + 4N) Bytes

SC = O(N)



System → complete memory

Space of algorithm i.e. extra space apart from input & output.

Q → Find max element of the given array.

```
          0   1   2   3   4   5
A = [ 2   8   5   3   10  6 ]      Ans = 10
```

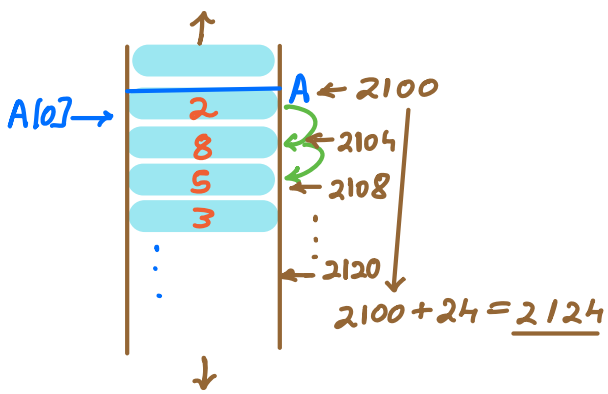
```
int maxElement ( A[N], N) {        // A[N] ← input
    ans = A[0]
    for i → 1 to (N-1) {
        if (A[i] > ans)    ans = A[i]
    }
    return ans                     SC = O(1)
}
```

```
          0   1   2   3   4   5
A = [ 2   8   5   3   10  6 ]
```

index → 0 to (N-1)

continuous part of memory

A[2] → 5



A[i] → memory of A + i*4 → TC = O(1)

Index of first & last array element → 0 , (N-1)

Q → Given an integer array of size N.

Check if there exists a pair s.t $a[i] + a[j] = K$ & $i != j$

```
          0   1   2   3   4
A = [ 9   1   3   5   9 ]      K = 12
```

Ans = true

```
        0   1   2   3   4
A = [ 3   5   2   7   3 ]      K = 6
```

A[0] + A[4] = 6 Ans = true

i != j

```
        0   1   2
A = [ 4   2   7 ]      K = 8
```

Ans = false

Bruteforce → ∀ i,j check a[i] + a[j] = K & i != j

try all possibilities

```
for i → 0 to (N-1) {
    for j → 0 to (N-1) {
        if ( A[i] + A[j] == K && i != j )
            return true
    }
} return false
```

TC = O(N²)

SC = O(1)

```
        0   1   2   3
A = [ 2  -6   8   3 ]
```

x + y = y + x

i \ j	0	1	2	3
0	(0,0)	(0,1)	(0,2)	(0,3)
1	(1,0)	(1,1)	(1,2)	(1,3)
2	(2,0)	(2,1)	(2,2)	(2,3)
3	(3,0)	(3,1)	(3,2)	(3,3)

i < j

i > j ✓

```
for i → 1 to (N-1) {
    for j → 0 to (i-1) {   // i > j
        if ( A[i] + A[j] == K )
```

return true

}

} return false

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

i	j	# iterations
1	0 — 0	1
2	0 — 1	2
3	0 — 2	3
⋮		⋮
N-1	0 — (N-2)	(N-1)

N-1 N-1

$\frac{N * (N+1)}{2}$

$1 + 2 + 3 + \ldots + (N-1)$

$= \frac{(N-1) * (N-1+1)}{2} = \frac{(N-1) * N}{2} \rightarrow O(N^2)$

N steps N/2 steps

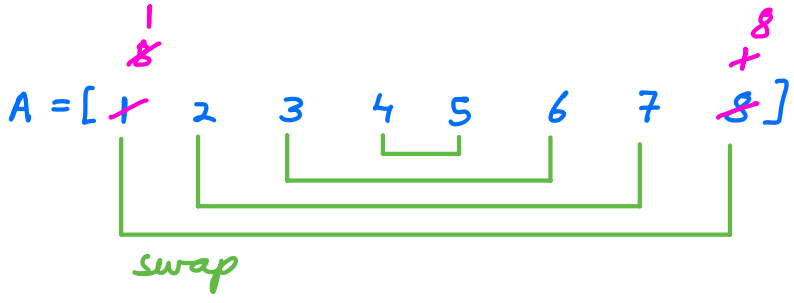
$O(N)$

10:30 PM

Q → Given an integer array, reverse the array.

A = [1 2 3 4 5]

5 4 3 2 1



for i → 0 to (N-1)/2 {

swap (A[i], A[N-1-i])

}

i = 0 → swap (A[0], A[7])

= 7 → swap (A[7], A[0])

0 N-1

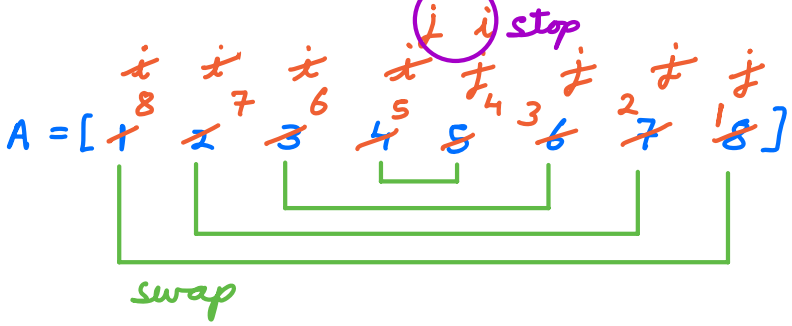
1 N-2

2 N-3

$t = A[i]$

$A[i] = A[N-1-i]$

$A[N-1-i] = t$




```
i = 0        j = (N-1)
while (i < j) {
    t = A[i]
    A[i] = A[j]        } swap(A[i], A[j])
    A[j] = t
    i++
    j--
}
```

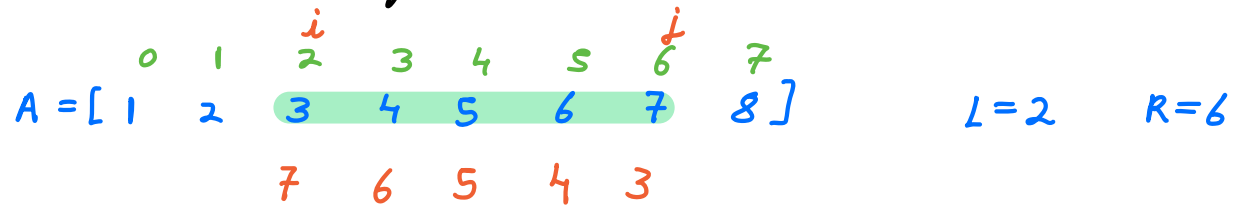
TC = O(N) SC = O(1)

```
// B[N]  → extra space
for i → 0 to (N-1) {
    B[i] = A[N-1-i]
}
for i → 0 to (N-1) {
    A[i] = B[i]
}
```

Reverse A using B

SC = O(N)

Q → Reverse array from index L to R



L = 2 R = 6

```
i = L      j = R
while (i < j) {
    t = A[i]
    A[i] = A[j]
    A[j] = t
    i++
    j--
}
```

swap (A[i], A[j])

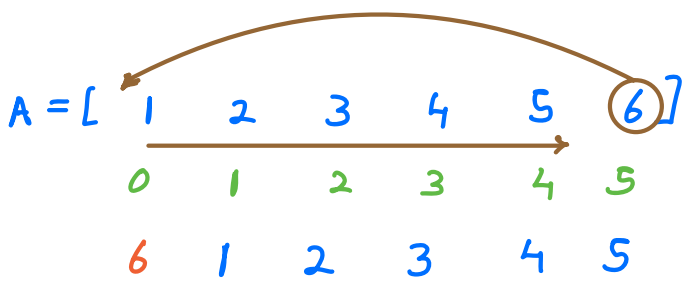
TC = O(N) SC = O(1)

Q → Given an integer array.
Rotate the array from right to left (forward) K times.

A = [1 2 3 4 5]

K = 1 → 5 1 2 3 4

2 → 4 5 1 2 3

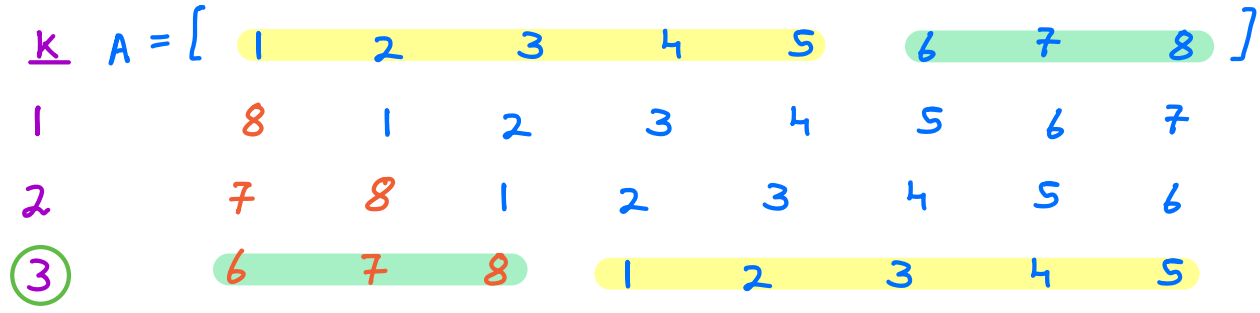


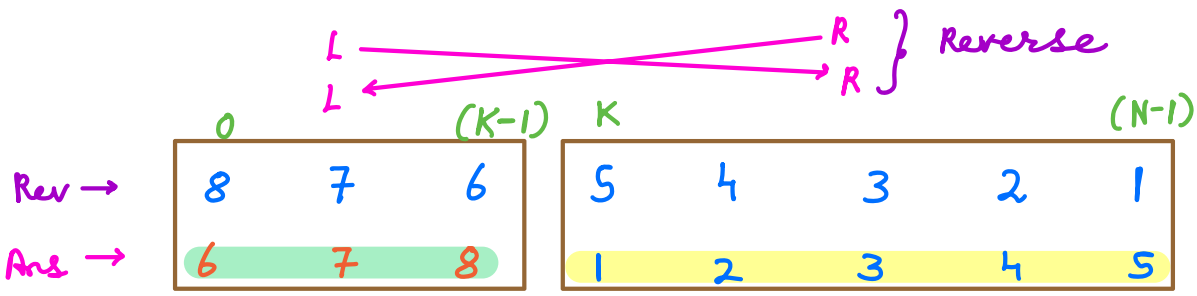
[Shift Right] ✓

```
for j → 1 to K {
    t = A[N-1]
    for i → (N-1) to 1
        A[i] = A[i-1]
    } A[0] = t
}
```

Rotate 1 time

TC = O(K * N) SC = O(1)



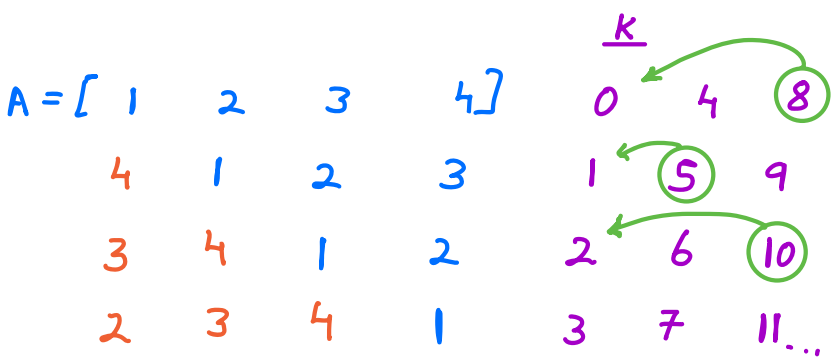


K = K % N

Sol →
1) reverse (A, 0, N-1)
2) reverse (A, 0, K-1)
3) reverse (A, K, N-1)

TC = O(3 * N) = O(N)
SC = O(1)

What if K is very large? (K >= N)



K = K % N
K = 10 2
10 % 4 = 2

H.W → Study dynamic array in your language.

no fix size

Java → ArrayList
C++ → Vector
Python → List
