Space Complexity → Rate of growth of space wrt input size.

int → 4B

long → 8 B

1)
```
{
    int x = 2;  // 4 B
   int y = 8;  // 4 B
  long z = x+y;  // 8 B
}
```

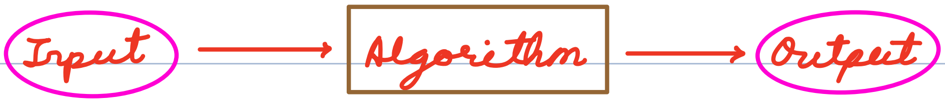
Total space = 16 B

SC = O(1)

2)
```
func (int N) {  // ~~4B~~
     int arr[10];  // 40 B
    int x;  // 4B
    int y;  // 4B
    long z;  // 8B
    int a = new int [N];  // 4*N B
}
```

Total space = (56 + 4N) B

SC = O(N)



SC is only wrt space apart from input & output.

3)
```
func (int N) {
    int x = N; // 4B
    int y = x * x; // 4B
    long z = x+y; // 8B
    int arr[] = new int [N]; // 4N B
    long a[][] = new long [N][N]; // 8 * N^2 B
}
```

Total = $(16 + 4N + 8N^2)$ B

SC = $O(N^2)$

4)
```
int maxArray (int a[], int N) {
    int m = A[0]
    for i → 1 to (N-1) {
        m = max (m, A[i])
    }
    return m  // Output space
}
```

Input space

4B

Algo Space   SC = $O(1)$

4B
```
for (int i = 1; i<(N-1); i++) {
    ⋮
}
```

Array → linear collection of same data type.

A = [5   8   3   4   1]

index → 0   1   2   3   4

A[3] = 4

0 to (N-1)

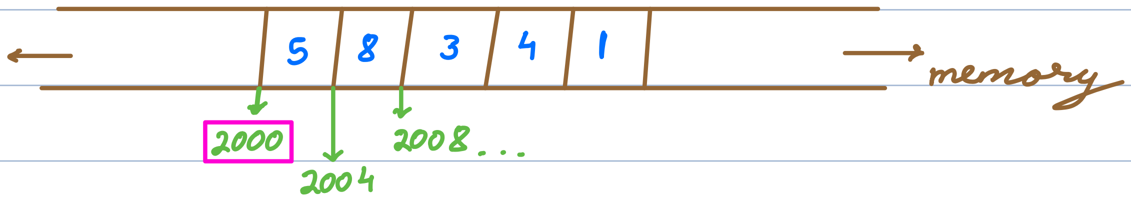
first   last

Q → Print all array elements.

```
for i → 0 to (N-1) {
        print (A[i])
}
```

Access i^{th} element → A[i]   TC = <u>O(1)</u>

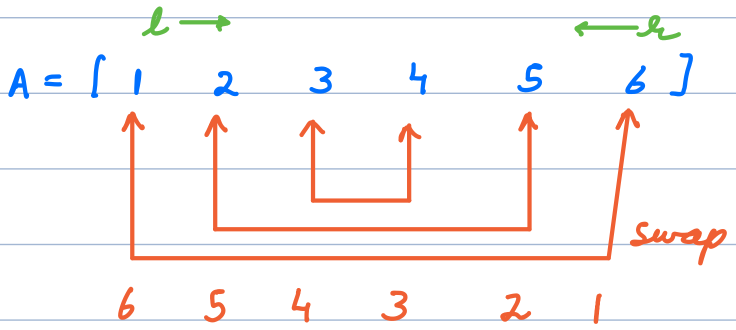
Array → continuous memory allocation



A[2] → 2000 + 2 * 4 = <u>2008</u>

Q → Given an integer array of size N.
Reverse the array.

A = [1 2 3 4 5]
↳ 5 4 3 2 1



```
swap (A[], i, j) {
    temp = A[i]
    A[i] = A[j]
    A[j] = temp
```

```
swap ( x, y ) { // without 3rd variable
    x = x + y      // 2+5 = 7
    y = x - y      // 7-5 = 2
    x = x - y      // 7-2 = 5
```

(above x: 2, above y: 5)

}   }

```
l = 0            r = (N-1)            O(N/2) → O(N)
while ( l < r ) {
    swap (A, l, r)
    l++                        TC = O(N)
    r--                        SC = O(1)
}
```

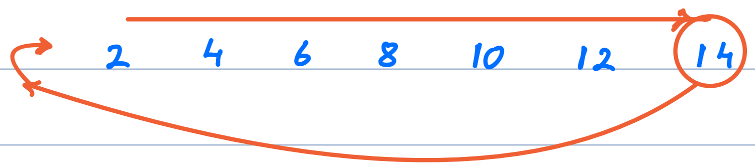
Q → Given an integer array & 2 integers L & R.
Reverse the array from index L to R.

		L			R		
A = [	2	4	6	8	10	12	14]
	0	1	2	3	4	5	6

└→ 2 10 8 6 4 12 14

```
l = L            r = R
while ( l < r ) {
    swap (A, l, r)
    l++                        TC = O(N)
    r--                        SC = O(1)
}
```

Rotate array left to right (clockwise) ↻



14 2 4 6 8 10 12

Q → Given an integer array, rotate the array left to right K times.

	0	1	2	3	4	K		
A =	[3	6	9	12	16]	0	5	10
K = 1	16	3	6	9	12	1	6	⋮
2	12	16	3	6	9	2	7	← 32
3	9	12	16	3	6	3	8	
4	6	9	12	16	3	4	9	
5								

K = 32 → 32 % 5 = 2

$\boxed{K = K \% N}$

Bruteforce →

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

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

Sol.

	0	1	2	3	4
A =	[3	6	9	12	16]
K = 2	12	16	3	6	9

Reverse A → [16 12] [9 6 3]

K = 2 12 16 3 6 9

first K elements (N-K) elements

<u>Sol</u> → 1) Reverse (A, 0, (N-1)) → O(N)

2) Reverse (A, 0, (K-1)) → O(K) } O(2N)

3) Reverse (A, K, (N-1)) → O(N-K)

TC = <u>O(N)</u> SC = <u>O(1)</u>

K = 3

	0	1	2	3	4	5	6	7	8
	1 →		(K-1)	K					(N-1)
A =	[3	6	9	12	16	18	20	15	17]
✓	17	15	20	18	16	12	9	6	3
✓	20	15	17	3	6	9	12	16	18

<u>Dynamic Arrays</u>

Limitation in array → fixed size

→ Resizing of the array is possible

→ continuous memory allocation → Access A[i] → TC = <u>O(1)</u>

	0	1	2	3		
DA =	[1	2	3	4]	↙ 4	← 5

↓ double the size ✓

[1 2 3 4 5 _ _ _] new memory add.

Eg → Java → Arraylist

C++ → Vector

Python → List

:

Next class ✓

Recap ✓

Doubts ✓

$$3^1 + 3^2 + 3^3 \ldots + 3^N$$

$$= \frac{3(3^N - 1)}{3 - 1} = 1.5 * (3^N - 1) \qquad TC = \underline{O(3^N)}$$

```
for (int a=1; a<10; a++) {

}
```