Arrays

TABLE OF CONTENTS

Notes

Space complexity → Gives an idea about extra space used

1.

```
void fun(int N){
    int x = N;
    int y = 2*x;
    long z = x+y;
}
```

→ 4B

→ 4B

→ 8B   64 bit integer

total extra space ⇒ $4+4+8 = 16B$

whatever already given is NOT counted

$O(1)$

2.

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

$40+4+4+8+4N$

$4N+56$

$\Rightarrow 4N$

$\Rightarrow O(N)$

3.

```
void fun(int N){
    int x = N;                  ⇒   4
    int y = x*x;                ⇒   4
    long z = y+y;               ⇒   8
    int[] arr = new int[N];     ⇒   4N
    long[][] b = new long[N][N];  ⇒  8N^2
}
```

total = $8N^2 + 4N + 16$

$8N^2$

$O(N^2)$

4.

```
int maxArr( int arr[], int N){
    int ans = arr[0];           4B
    for(i=0; i<N; i++){         4B
        ans = Max(ans, arr[i]);
    }
        return ans;
}
```

$O(N)$ ✗

$O(1)$ ✓

$4 + 4 = 8B$

$\Downarrow$

$O(1)$

Introduction to Arrays

int arr [n]

first elem ⇒ arr [0]

last elem ⇒ arr (n-1)

< *Question* > : Print all elements of the array

```
for i : 0 ⟶ n-1
        print ( arr [i] )
```

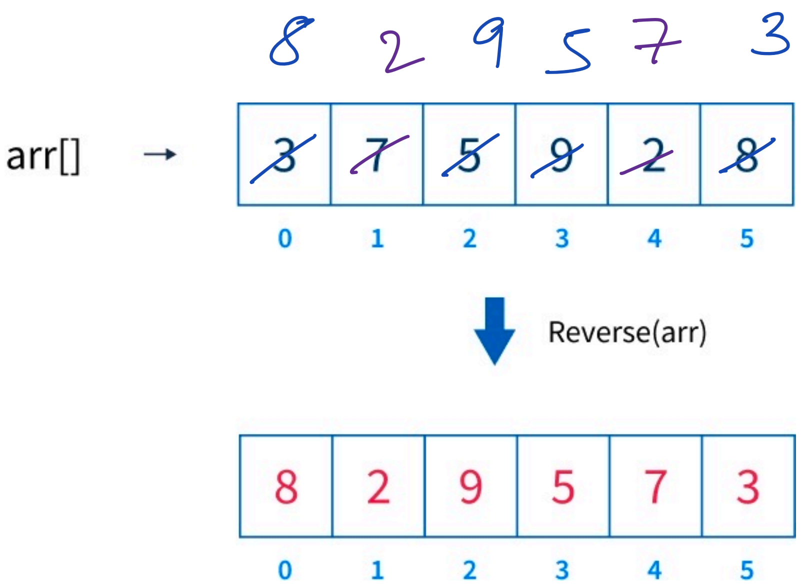
print (arr [53]) ⇒ O(1)

20 30 50

1st + 5th

a[0] + a[4]

2. Reverse the given array



i	j	
0	5	swap
1	4	swap
2	3	swap
3	2	done!

when $i > j$

Stop.

50 40 20 10

~~10~~	~~20~~	30	~~40~~	~~50~~
0	1	2	3	4

</> Code

i	j	
0	4	swap
1	3	swap
2	2	stop

stop condition is when $i \geq j$

running condition ⇒ $i < j$

```
i = 0        j = n-1
while  i<j  :
    swap  a[i]  and  a[j]
    i++
    j--
```

$a, b = b, a$

$a[i], a[j] = a[j], a[i]$

TC : $O(N)$

SC : $O(1)$

3. Reverse part of an array

l = 3, r = 8

arr →

			1	5	6	9	12	4	
13	7	8	~~4~~	~~12~~	~~9~~	~~6~~	~~5~~	~~1~~	18
0	1	2	3	4	5	6	7	8	9

13	7	8	1	5	6	9	12	4	18
0	1	2	3	4	5	6	7	8	9

```
void reversePart(int []arr, int l, int r){
    i = l          j = r
    while   i<j   :
        swap  a[i]  and  a[j]
        i++
        j--
}
```

a, b = b, a

a[i], a[j] = a[j], a[i]

TC: O(N)

SC: O(1)

Rotate the array right to left K times

N=7

1	2	3	4	5	6	7
0	1	2	3	4	5	6

K=1

7	1	2	3	4	5	6
0	1	2	3	4	5	6

K=2

5	6	7	1	2	3	4
~~6~~	~~7~~	~~1~~	~~2~~	~~3~~	~~4~~	~~5~~
0	1	2	3	4	5	6

K=3

5	6	7	1	2	3	4
0	1	2	3	4	5	6

K=4

4	5	6	7	1	2	3
0	1	2	3	4	5	6

Ideas: Perform the shifting k times

```
for i 0 → k-1 :
    last_num = ar[n-1]
    for( j n-2 → 0 , j--)
        ar[j+1] = ar[j]
    ar[0] = last_num
```

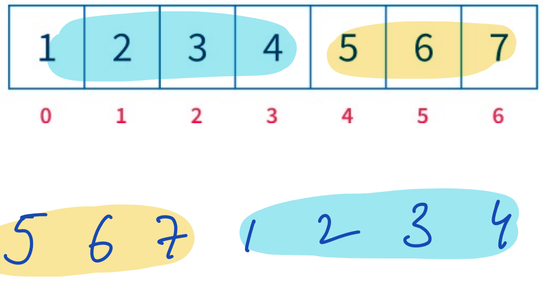
40	10	20	30	last_num = 40
~~10~~	~~20~~	~~30~~	~~40~~	
0	1	2	3	

TC: $O(nk)$

Optimisation

k = 3

N=7



1	2	3	4	5	6	7
0	1	2	3	4	5	6

5 6 7 1 2 3 4

1 2 3 4 5 6 7

Step-1 reverse the array 7 6 5 4 3 2 1

Step-2 reverse first k 5 6 7 4 3 2 1

Step 3 reverse last n-k 5 6 7 1 2 3 4

Code

$k = k \% n$

1) reverse(arr, 0, n-1)   N

2) reverse(arr, 0, k-1)   N

3) reverse(arr, k, n-1)   N

total time = $N+N+N = 3N = O(N)$

1	2	3	4	k = 0	0
4	1	2	3	k = 1	1
3	4	1	2	k = 2	2
2	3	4	1	k = 3	3
1	2	3	4	k = 4	0
4	1	2	3	k = 5	1
				⋮	2

$k \Rightarrow k \% n$

1) Dynamic arrays.

Java

```
ArrayList <Integer>
  arr = new ArrayList<>();

arr. clear ()
```

Python

```
mylist = []
mylist. append(10)
mylist. clear ()
```

{done}

6 % 4   10 % 4   14 % 4

2 % 4   18 % 4

1 2 3 4

4 3 2 1

4 1 2 3