ARRAYS - 1

Start doing dry run, start using pen and paper while solving DSA problems. Trust me half of your problems will be solved.

Try this !!

Good Morning

Agenda

01. Space complexity
02. Arrays Introduction
03. Reverse Array
04. Rotate Array
05. Dynamic Array

Space complexity → Max space used at any point of time during the execution of algorithm

Space complexity is also measured by Big(O)

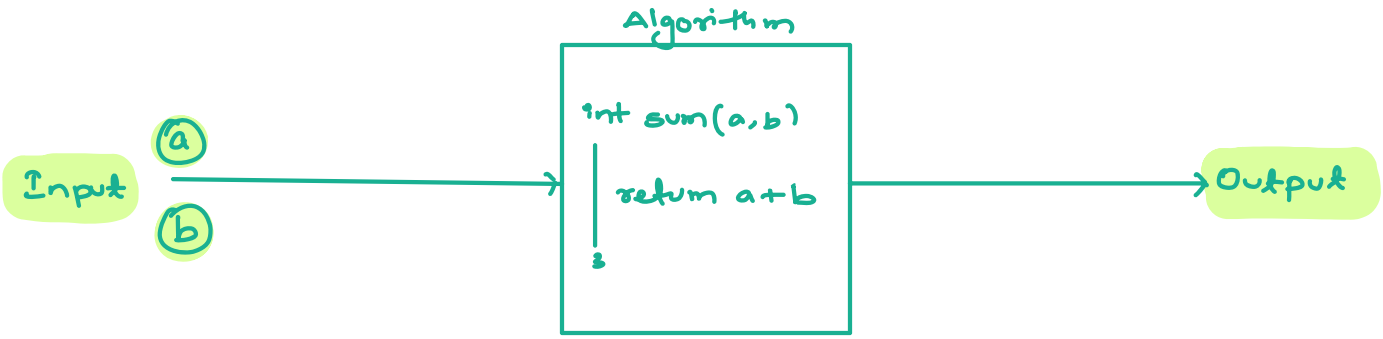
* function (int N){

```
    int x;    // 4 B
    int y;    // 4 B
    long z;   // 8 B
}
```

Space = 4 + 4 + 8

= 16 Bytes

SC : O(1) → independent of input



Obs 1 → Space complexity is the extra space taken by algorithm.

Input space ✗

Output space ✗

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

Space = 4 + 4 + 8 + 4N

= 16B + 4*N

SC: O(N)

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

Space = 4 + 4 + 8 + 4N + 8N^2

= 16 + 4N + 8N^2

SC: O(N^2)

```
int max (int [] ar, int N)
    int ans = 0
    for (int i = 0; i < N; i++) {
        if (ar[i] > ans) {
            ans = ar[i];
        }
    }
    return ans;
}
```

TC : O(N)

SC : O(1)

```
int[] max (int [] ar, int N)
    int ans = 0 // 4B
    for (int i = 0; i < N; i++) {   // 4B
        if (ar[i] > ans) {
            ans = ar[i];
        }
    }
    int [] A = new int [1];
    A[0] = max;
    return A
}
```

Space = 8B

SC : O(1)

Arrays

- Collection of data of same data type
- Contigous memory location
- Ordered linear data structure
- Memory is predefined (size is fixed)

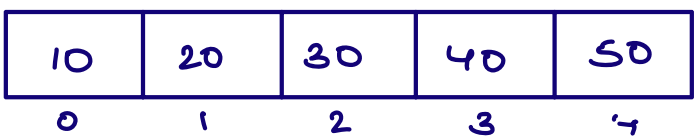
```
int arr[n];
```

n = size of array

arr = name of the array

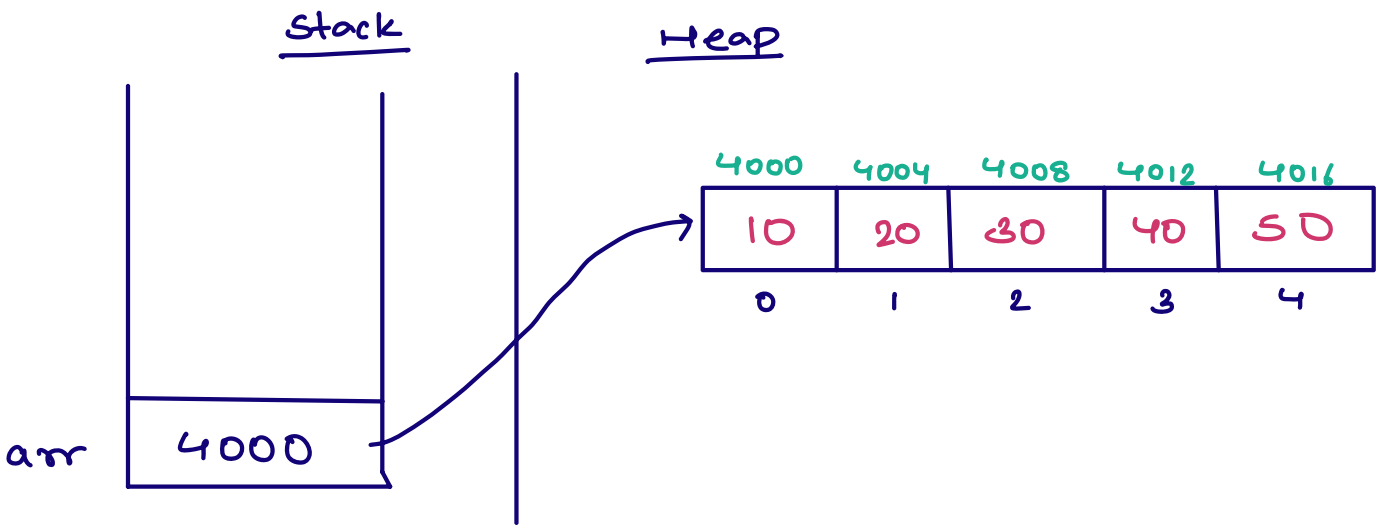
int = type of data this array is going to store

```
int [ ] arr = new int [5]
```



arr[0] = 10

arr[n-1] = 50



arr[0] = 4000 + 0 integers

= 4000 + 0 * 4

= 4000 ⟶ 10

arr[3] = 4000 + 3 integers

= 4000 + 3 * 4B

= 4000 + 12B

= 4012 ⟶ 40

To fetch value = base address + idx * size of datatype

print all elements in array

```
void print_array (int [] A, int n ){
    for (int i=0 ; i<n ; i++){
        print (A[i]);
    }
}
```

→ size of array

Tc: O(n)

Sc: O(1)

Accessing elements inside Array → O(1)

$A[5] = \{5, -4, 8, 9, 10\}$
0 1 2 3 4

sum of 1st & 5th element

$A[0] + A[4]$

element	idx
1	0
2	1
3	2
4	3
5	4

7:57 AM → 8:07 AM + 1 more min

Q Given an array of size N. Reverse the entire array

N = 5

$A[] = \{10, 20, 30, 40, 50\}$
0 1 2 3 4

Output: $A[] = \{50\ 40\ 30\ 20\ 10\}$
0 1 2 3 4

Brute force Approach = Take extra memory of array size & fill that array traversing from last

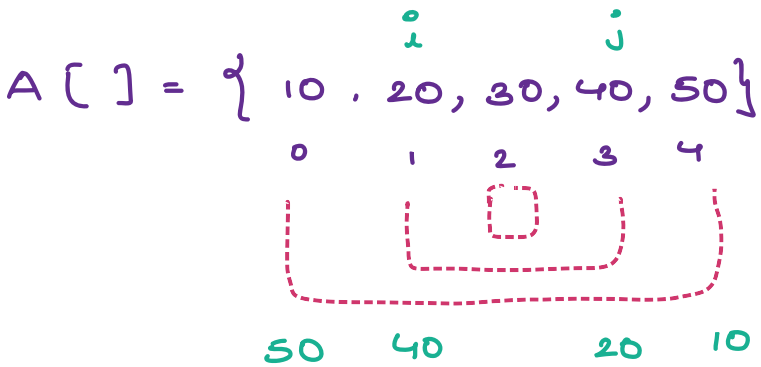
TC: O(n) SC: O(1)

```
int [] reverse (int [] arr){
    int [] ans = new int [n];   int j=0

    for (i= n-1; i ≥ 0; i--){
        ans[j] = arr[i];
        j++;
    }
    return ans;
}
```

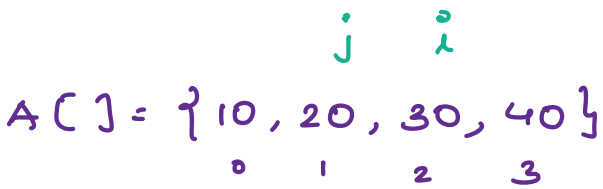
TC: O(N)
SC: O(1)

* Without taking extra memory



i	j	
0	4	swap A[0] & A[4]
1	3	swap A[1] & A[3]
2	2	⟶ stop

i<j



40 30 20 10

i	j	
0	3	swap A[0] & A[3]
1	2	swap A[1] & A[2]
2	1	⟶ stop

i < j

```
void reverse (int [] ar){
    int i = 0
    int j = n-1
    while (i<j){
        int t = A[i]
        A[i] = A[j]
        A[j] = t
        i++; j--;
    }
}
```

TC: O(N)

SC: O(1)

Q Given an array, st, e . Reverse the array from st to e .

```
void reverserange (int [] ar, int s, int e){
    int i = s
    int j = e
    while (i<j){
        int t = A[i]
        A[i] = A[j]
        A[j] = t
        i++; j--;
    }
}
```

TC: O(n)
SC: O(1)

* Rotate array

Q Given an array of size N. Rotate the array from right to left/anticlockwise K times.

A[] =

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

K=3

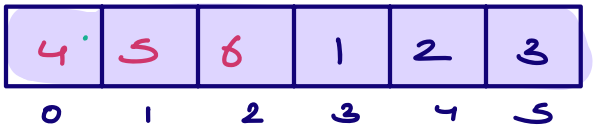
K=1

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

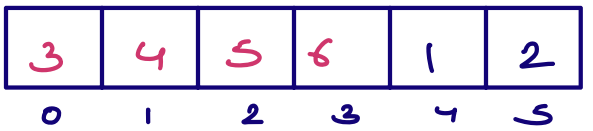
K=2

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

k=3



k=4



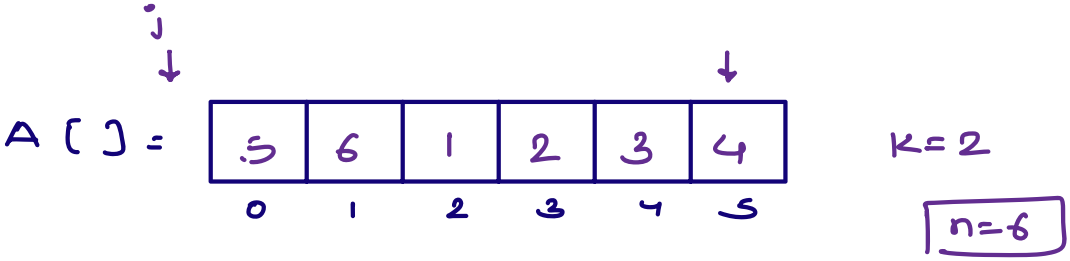
Brute force → Pick the last element, move it in the front & move rest of elements on the RHS k times

```
void rotate ( int [] A, int K)
{
    for ( i=0 ; i < K ; i++){
        int last = A[n-1]
        for ( j=n-2 ; j≥0 ; j--){
            A[j+1] = A[j];
        }
        A[0] = last;
    }
}
```

n → size of array

TC : O(K*n)

SC : O(1)



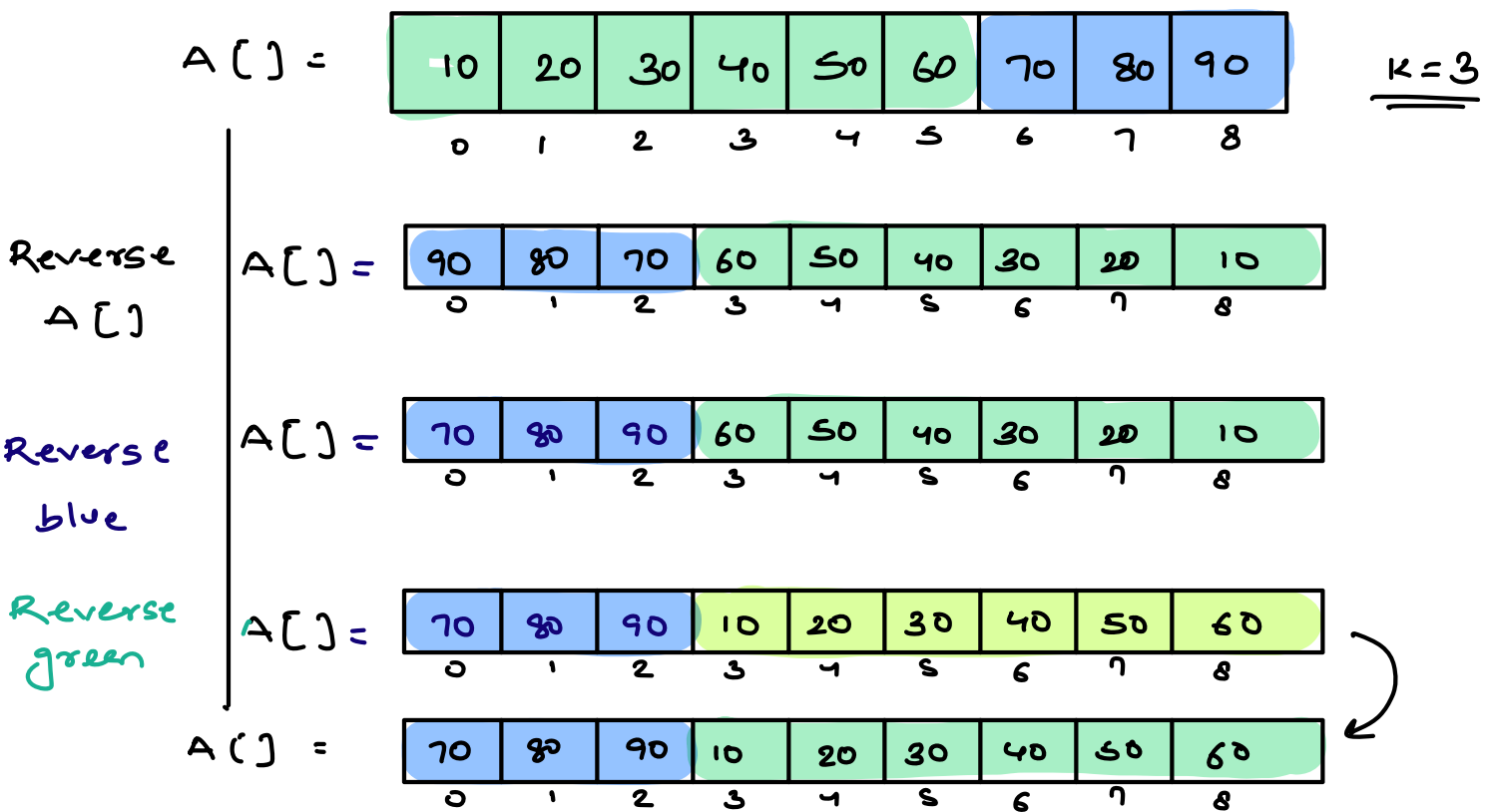
$i = 0$	$0 < 2$	
$+$	$1 < 2$	
2	$2 < 2$ ✗	

last = A[5] = 5

$j = 4$	$4 \geq 0$	$A[4+1] = A[4]$
3	$3 \geq 0$	$A[3+1] = A[3]$
2	$2 \geq 0$	$A[2+1] = A[2]$
1	$1 \geq 0$	$A[1+1] = A[1]$
0	$0 \geq 0$	$A[0+1] = A[0]$

$A[j+1] = A[j]$

Observation



```
void rotate (int [] A, int k){
    k = k % n
    if (k==0) return;
    reverserange (A, 0, n-1)
    reverserange (A, 0, k-1 )
    reverserange (A, k, n-1)
}
```

TC : O(n)

SC : O(1)

A[] = { 1, 2, 3, 4 }

K = 0 { 1, 2, 3, 4 }

K = 1 { 4, 1, 2, 3 }

K = 2 { 3, 4, 1, 2 }

K = 3 { 2, 3, 4, 1 }

K = 4 { 1, 2, 3, 4 }

K = 5 { 4, 1, 2, 3 }

K = 6 { 3, 4, 1, 2 }

K = 7 { 2, 3, 4, 1 }

K = 8 { 1, 2, 3, 4 }

K = 9 { 4, 1, 2, 3 }

K = 9 ⟶ 1

K = 10 ⟶ 2

K = 7 ⟶ 3

K = 9 % 4 ⟶ 1

K = 10 % 4 ⟶ 2

K = 7 % 4 ⟶ 3

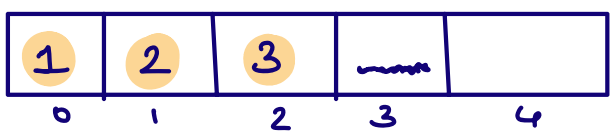
* Dynamic Arrays → Automatic resizing

Drawback of normal Arrays → fixed size

Strength → Accessing of elements is O(1)

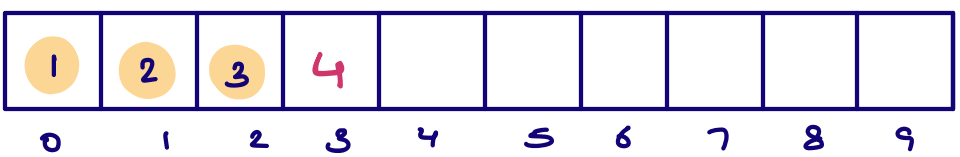
→ Resize itself whenever elements are increase

Weakness → Adding elements



Lf = 0.7

0.7 * 5
= 3.5



0.7 * 10
= 7 elements

TC to add elements ⇒ Amortized O(1)

Average O(1)

Please don't miss it.

Next class → prefix array → When?
How?
Problem based Prefix array