SCALER

Arrays

TABLE OF CONTENTS

Notes

Space Complexity

→ The maximum extra space (worst case) that is utilised by our function / algorithm at any point of time.

→ Big-O notation will be used.

int → 4 Bytes
long → 8 Bytes

] Data-types.

$16 \times N^0$
$+$
$N^0 \Rightarrow O(N^0)$
$\Rightarrow O(1)$

O(1) or constant

1.

```
void fun(int N){   // 4 B
    int x = N;     // 4B
    int y = 2*x;   // 4B
    long z = x+y;  // 8B
}
```

16 B

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
}
```

total extra space $= 40 + 4 + 4 + 8 + 4N$

$= 56 + 4N$

S.C $\rightarrow O(N)$

3.

```
void fun(int N){
    int x = N;                  → 4 B
    int y = x*x;                → 4 B
    long z = y+y;               → 8 B
    int[] arr = new int[N];     → 4N B
    long[][] b = new long[N][N]; → 8×N² B
}
```

Total extra space = $4+4+8+4N+8N^2$

$S.C \rightarrow O(N^2)$

i/p space.

4.

```
int maxArr( int arr[], int N){
    int ans = arr[0];
    for(i=0; i<N; i++){
        ans = Max(ans, arr[i]);
    }
    return ans;      o/p space.
}
```

Max element in the array

total extra space → 4B

$S.C \rightarrow O(1)$

$\boxed{\cancel{3}\ 4\ \cancel{0}\ \cancel{1}\ \cancel{2}\ 5}$

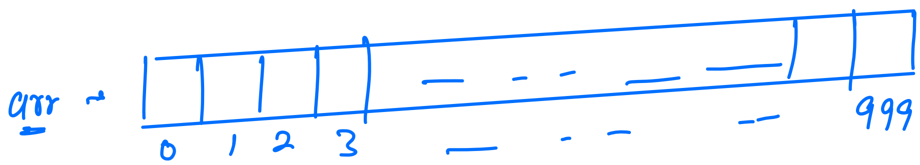
i

Introduction to Arrays

⇒ collection of same types of data.

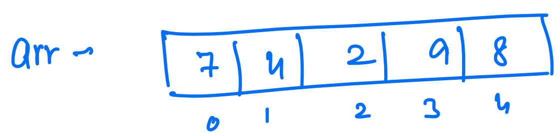
```
int marks1 = 50;
int marks2 = 100;
    |
    |
int marks1000 = 98
```

⇒ int arr[1000];



idx. → how much distance away is current element from the base address.

< Question > : Print all elements of the array



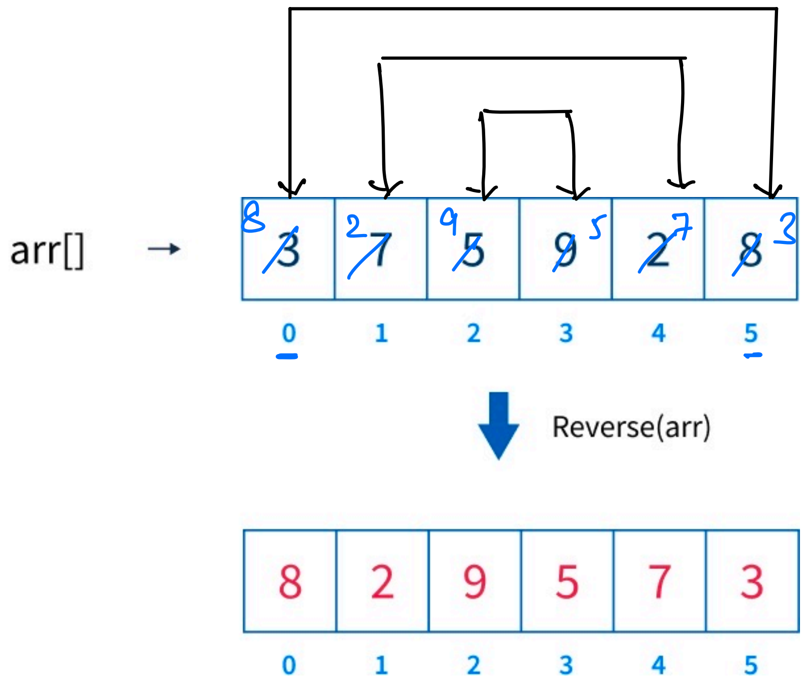
Exp o/p. →

7 4 2 9 8

```
void printArr( int[] arr, int n){
    for( i = 0; i < N ; i++){
        print( arr[i]);
    }
}
```

T.C → O(N)
S.C → O(1)

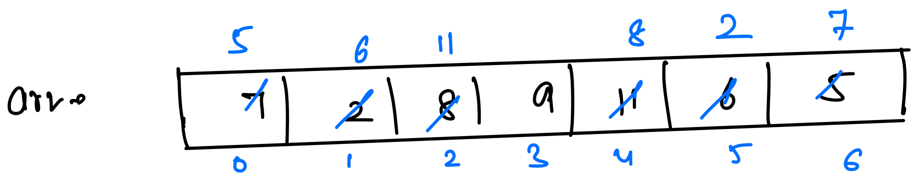
2. Reverse the given array



$l = 0 \to 1 \to 2 \to 3$

$r = 5 \to 4 \to 3 \to 2$

N = 7



$l = 0$, $r = 6$

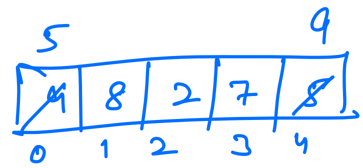
$l: 0 \to 1 \to 2 \to 3$

$r: 6 \to 5 \to 4 \to 3$

</> Code

```
void ReverseArray( int[] arr, int N){
    int l = 0 , r = N-1;
    while( l < r){
        //swap arr[l] with arr[r]
        int temp = arr[l];
        arr[l] = arr[r];
        arr[r] = temp;

        l++;
        r--;
    }
}
```

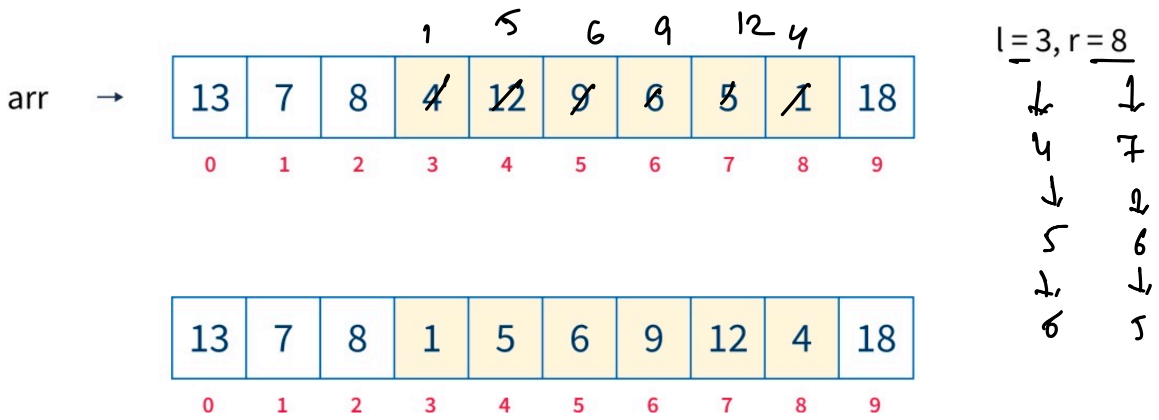


l = 0
r = 4

temp → 9

T.C → O(N)
S.C → O(1)

3. Reverse part of an array



Input

```
void reversePart(int []arr, int l, int r){
    while( l < r){
        //swap arr[l] with arr[r]
        int temp = arr[l];
        arr[l] = arr[r];
        arr[r] = temp;

        l++;
        r--;
    }
}
```

T.C → O(N)
S.C → O(1)

Rotate An Array

Given an array of size N & an integer K. Rotate arr[] by K.

$1 \le N \le 10^6$

$0 \le K \le 10^9$

N=7

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

last → arr[6] → 7

K=1

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

K=2

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

K=5.	3	4	5	6	7	1	2
K=6	2	3	4	5	6	7	1
K=7	1	2	3	4	5	6	7
K=8	7	1	2	3	4	5	6

B.f idea. → Keep bringing the last element at 0th index K times.

```
for( i = 0; i < K; i++){
        int last = arr[N-1];
        for( j = N-2 ; j >= 0; j--){
                arr[j+1] = arr[j];
        }
        arr[0] = last;
}
```

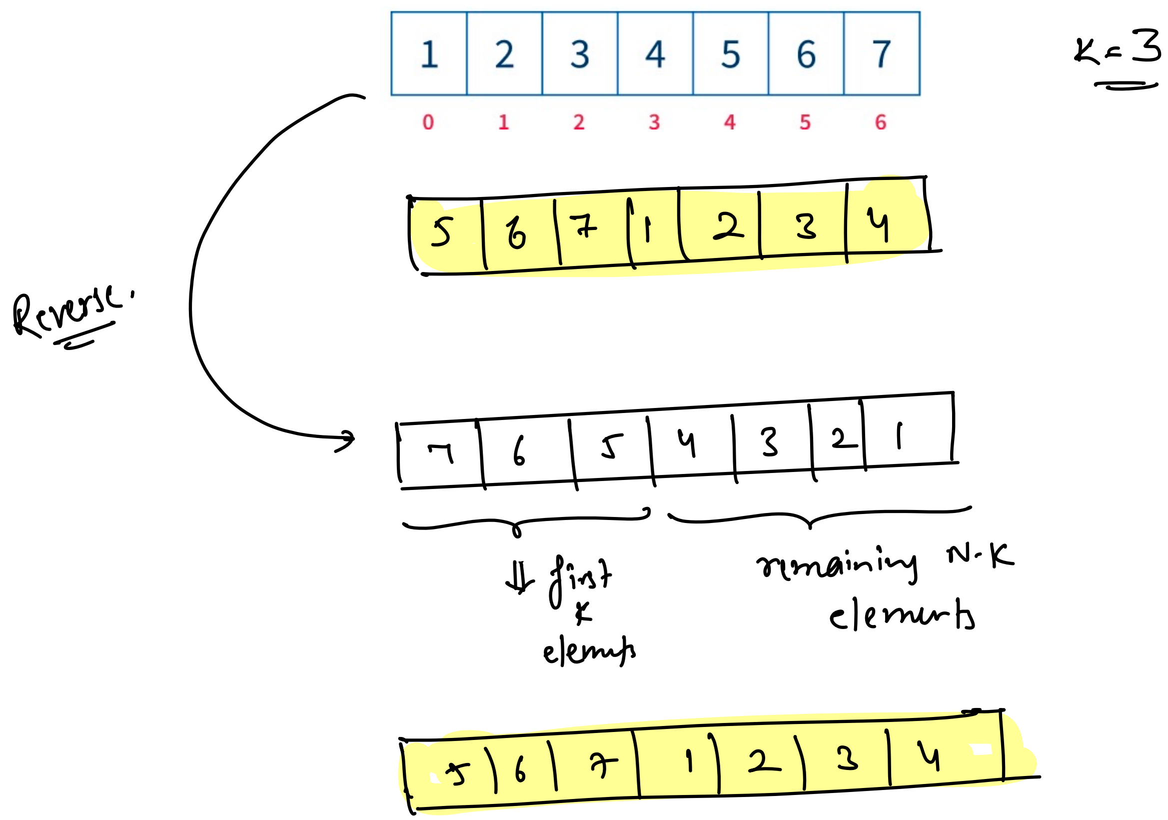
Last element will be shifted to 0th index.

T.C → O(K×N)
S.C → O(1)

→ T.L.E.

Optimisation

N=7



0. $K = K \% N$
1. Reverse the whole array
2. Reverse first K elements.
3. Reverse remaining N-K elements

```
void rotate( int[] arr, int N, int K) {
    K = K % N;
    reversePart( arr, 0, N-1 );     → N
    reversePart( arr, 0, K-1 );     → K
    reversePart( arr, K, N-1 );     → N-K.
}
```

T.C → O(N)
S.C → O(1)

N=7.

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

, K=8.

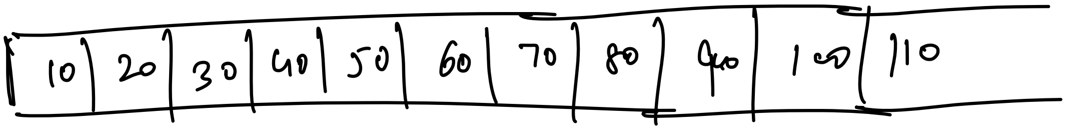
reversePart(arr, 0, 8) ✓

reversePart(arr, 0, 7);

Drawback of Arrays →

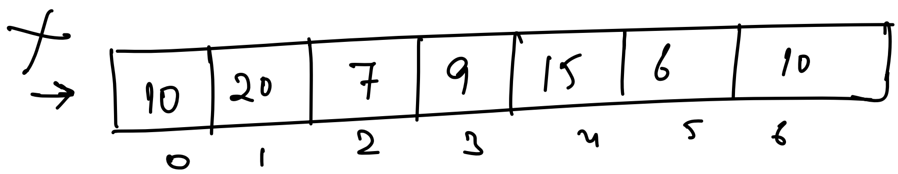
→ the size of the array has to be declared beforehand & that can't be changed.

Dynamic Array / List / Vector / Array List



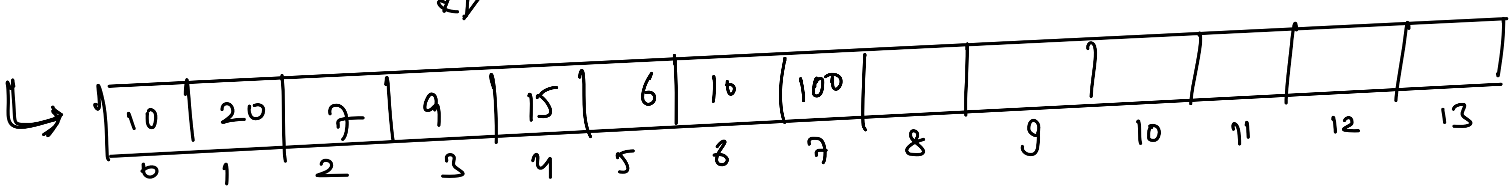
al.get(i);

al.add(100);



cap → ~~7~~ 14

size → ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ 8



ArrayList → Java

List → Python, C#

Vector → C++

⇒ Prefix - Sum

Memory management.

Time Limit → 1 sec

Space Limit → 256 MB

```
for( i = 0; i < 2^N; i++) {
    int j = i;
    while( j > 0) {
        j -= 1;
    }
}
```

i	j	iters
0	—	0
1	—	1
2	—	2
3	→	3
4	→	4 + . +
$2^N - 1$	→	$2^N - 1$

$$\Rightarrow \quad \underbrace{1+2+3+4+ \text{---} \text{---} (2^n-1)}$$

$$\frac{2^n(2^n+1)}{2} \Rightarrow \frac{2^n \times 2^n}{2} + \frac{2^n}{2} \qquad \frac{n(n+1)}{2}$$

$$\Rightarrow \frac{2^{2n}}{2} + \frac{2^n}{2}$$

$$\Rightarrow \rightarrow \frac{4^n}{\cancel{2}} + \cancel{\frac{2^n}{2}}$$

$$\underline{\underline{O(4^n)}}$$

```
int a=0, b=0;

for( i=0; i<N; i++) {
    for(j=0; j<N; j++) {
        a=a+j;
    }
}

for( K=0; K<N; K++) {
        b=b+K
}
```

$$\boxed{\begin{array}{l} T.C \rightarrow O(n^2) \\ S.C \rightarrow O(1) \end{array}}$$