Intro to Arrays

Nov 27, 2023

AGENDA

- Space complexity
- Intro to Arrays
- Reverse an array / Reverse part of an array
- Rotate array K times

Space complexity

: Max space (memory) used by the program at any point of time.

: Denoted in Big-O notation.

```
func (int N)
{
    int x;  // 4 bytes
    int y;  // 4 bytes
    long z; // 8 bytes.
}
```

func (int N) → 4 bytes

* While calculating S.C., only consider the variables that you created.

Do not consider variables provided by input.

No. of bytes used = 4+4+8 = 16 bytes.

↓ Convert to Big O

↓

O(1)

```
func (int N)
{
    int arr = new arr[10]; // Array of size 10. → 10*4 bytes = 40
    int x;  → 4 bytes
    int y;  → 4 bytes.
}
```

Total bytes = 40+4+4

= 48 bytes.

↓

O(1)

```
func (int N)
{
    int arr[10];          → 40 bytes
    int x;                → 4
    int y;                → 4
    long z;               → 8
    int arr2[] = new int[N];   → 4*N bytes
}
```

4N + 56 bytes

↓

O(N) S.C.

```
func (int N) {

    int x = N          -
    int y = x * x      -
    int z = x + y      -
    int[] arr = new int[N]          // N* 4 bytes
    long[][] l = new long[N][N]     // N*N*8 bytes
}
```

4N + 8N²

↓

O(N²)

Q.

```
for(int i=0;i<n;i++)
{
    int s=i
    int j=i+3
}
```

These variables get created and destroyed in every iteration. Hence, max space utilized is O(1).

```
int s=0
for (int i=0 ; i<n;i++)
{
    s= s+i
}
```

s

The variable gets over-ridden every time, hence S.C. → O(1).

```
func ( int arr[] ) {
int   maxArr=0
for (int  i=0;i<n;i++)
{
    maxArr= max (maxArr, arr[i])
}
}
```

input.

4 bytes.

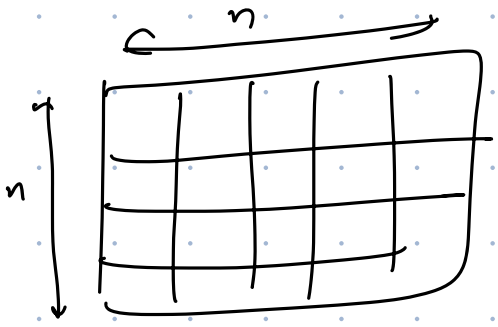
4 bytes

S.C. → O(1)

↓

You are not creating any extra array.

String → is an array.




```
for(int i=0; i<n; i++)
{
    int arr[]=new int[n];
}
```

↓

S.C → O(N)

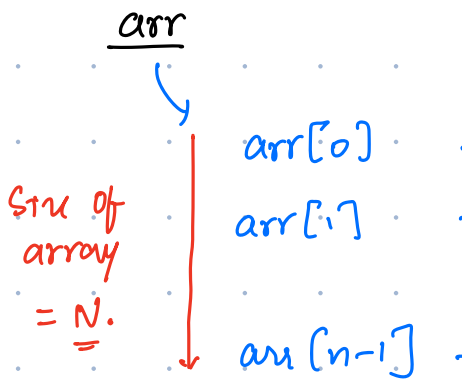
At any instant of time, max space utilization was O(N).

Arrays.

↳ collection of same type of data.

↳ stored in a contigous manner in memory.

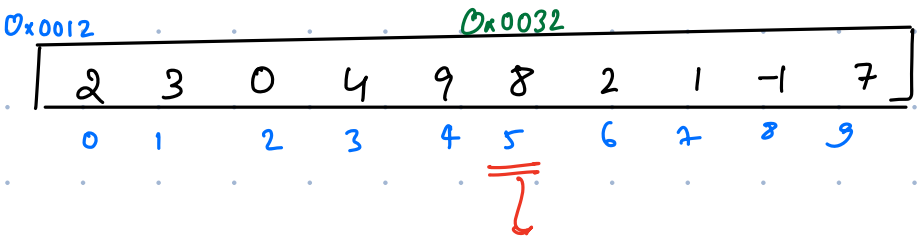
↳ Array indexing starts from 0.



* Print all elements of the array.

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

Time complexity to access ith index?



memory address → RAM → value stored there.

O(1) time :)

RAM → Random Access Memory

↓

It would give you random access in O(1) time.

* Conclusion : Random access in array is O(1) time.

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

print(arr[i]) → O(1)

T.C. → O(N)

S.C. → O(1)

Q. Given an array of size N, reverse the given array.

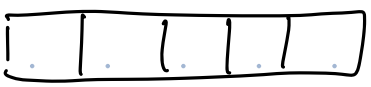
1 2 4 7 8

↓

8 7 4 2 1

Brute Force

1 2 4 7 8



* Create new array with same size.
* Iterate on the original array in reverse, and fill it here.

```
int[] temp = new int[n];
int j=0
for(int i=n-1; i>=0; i--)
{
    temp[j] = arr[i]
    j++
}
```

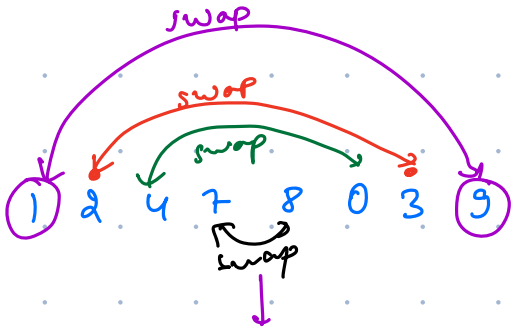
T.C. → O(N) ✓

S.C. → O(N)

You are creating an extra array.

not good enough ∴ optimise.

Optimised



9 3 0 8 7 4 2 1

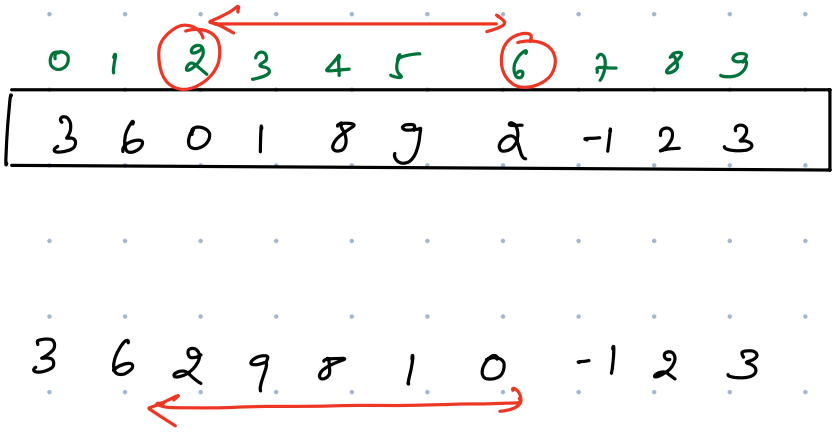
```
i = 0
j = n - 1
while ( i < j )  → Break as soon as i crosses j:
{
    int temp = arr[i]
    arr[i] = arr[j]
    arr[j] = temp
    i++
    j--
}
```

1 2 4 7 8 0 3 9
i → ← j

1 2 4 7 8 0 3 9
i j

T.C. → O(N)
S.C. → O(1)

Q. Reverse the array in a given range.




```
void reverseinRange( int arr[], int s, int e)
{
    i = s
    j = e
    while( i<j)
    {
        swap arr[i] & arr[j]
        i++
        j--
    }
}
```

T.C. → O(N)
S.C. → O(1)

Break till 8:15 AM

Q. Given an array of size N, rotate the array K times from right to left [last element will come at first].

e.g. 1 2 3 4 5

K=1 → 5 1 2 3 4

K=2 → 4 5 1 2 3

e.g. 1 0 3 9 2 5

K=3 → 9 2 5 10 3

↓

If I rotate the array 'n' times, the array is restored back.

1 2 3 4 5

K = 97

↓

same as rotating 2 times.

K = K % n

Array size

K = 32, n = 4 ⟶ 0 times. (32 % 4 = 0)

K = 12, n = 7 ⟶ 5 times. (12 % 7 = 5)

B.F.

1 2 3 4 5 6 7

* Do K rotations one by one.

Bring last element to the front K times.

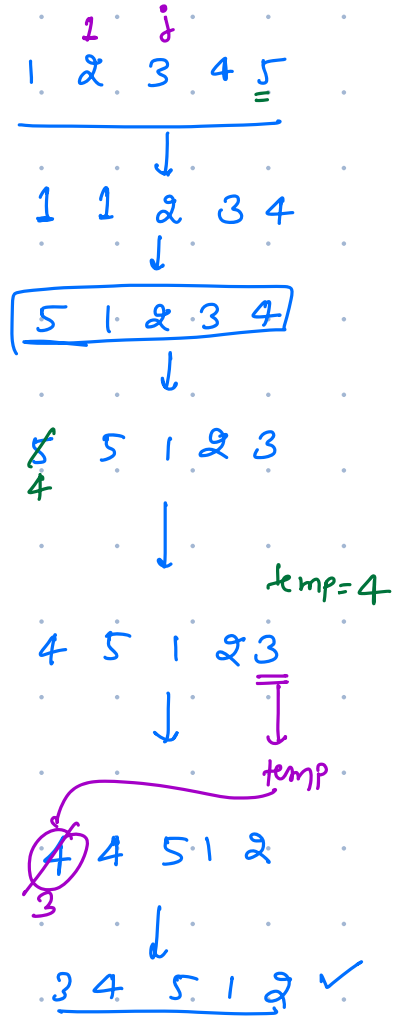
↓

temporarily save the variable arr[n-1].
Shift right all elements frm 0 to n-2.
arr[0] = put the temporarily saved no.

Code.

K = K % N

```
for(int i=0; i< K ;i++)   ← Running it K times.
                            ↳ should run K times.
{
    int temp = arr[n-1];
    for(int j=n-1; j>=1 ;j--)
    {
        arr[j] = arr[j-1]
    }
    arr[0] = temp
}
```



T.C. → O(K*N)

S.C. → O(1)

↳ Quadratic.

↓

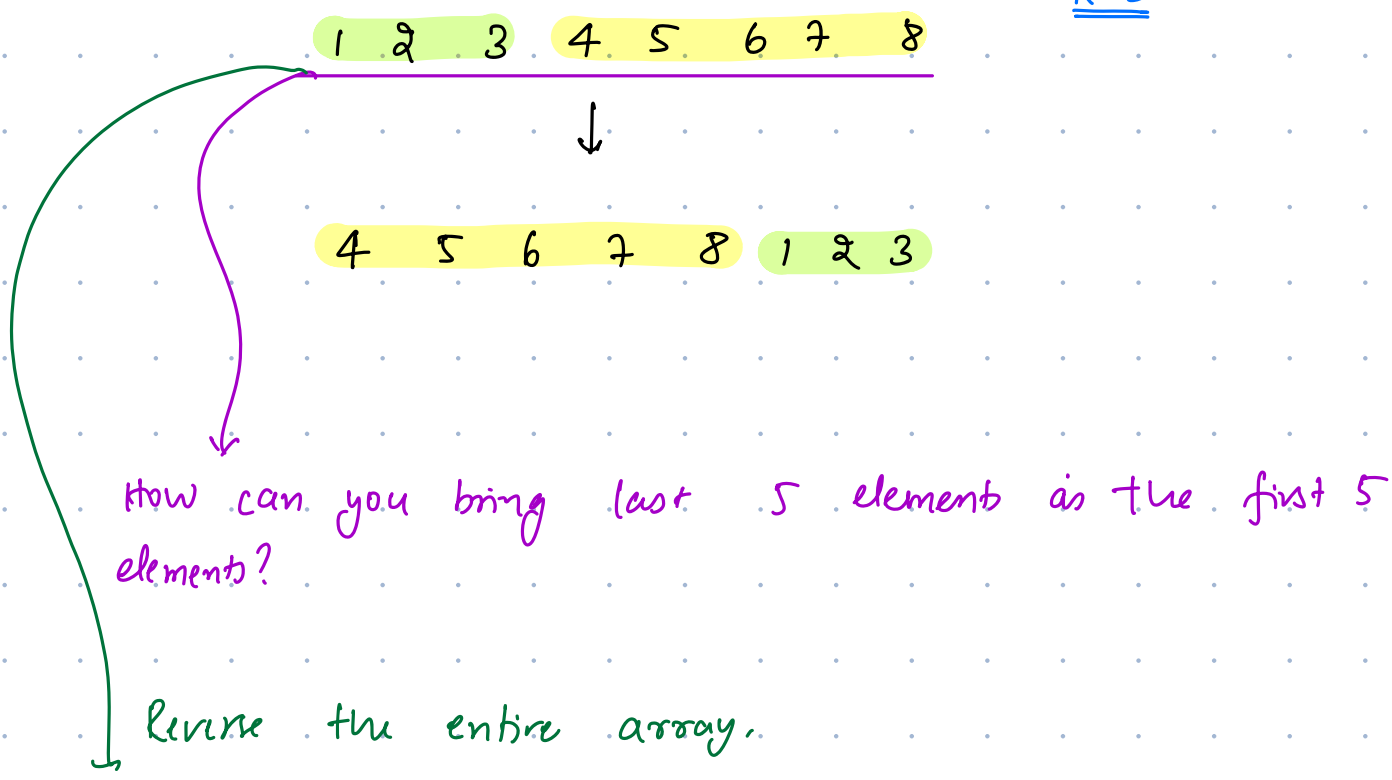
Not good enough!

Optimisation ?

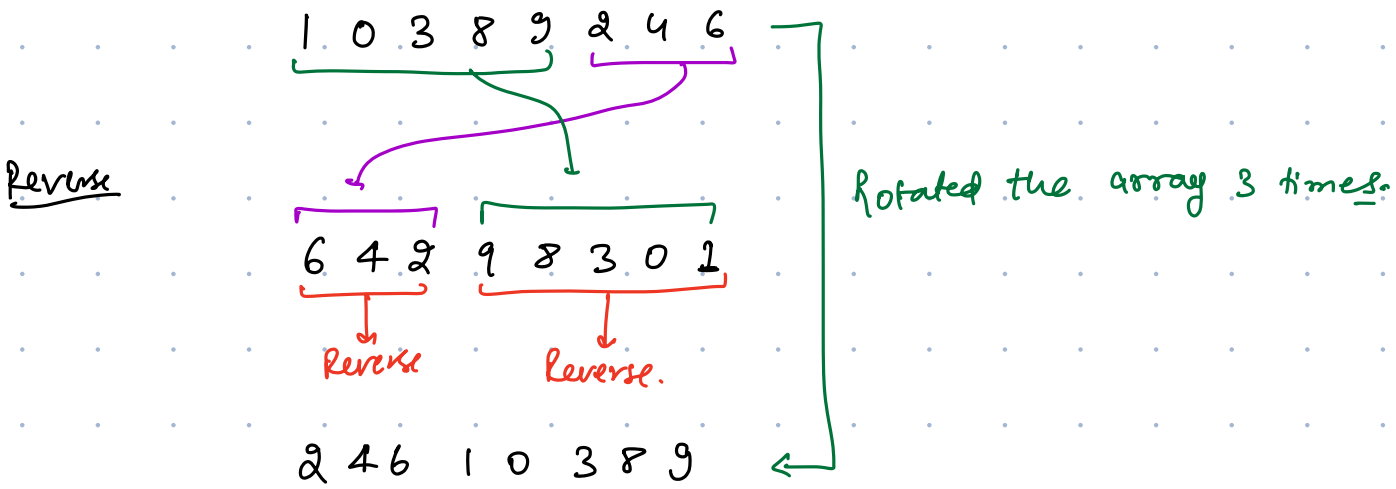
↳ Goal : T.C. → O(N)

S.C → O(1)

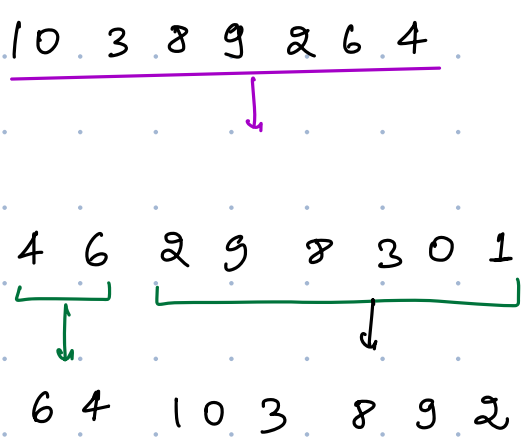
K=5



K = 3


1 0 3 8 9 2 4 6
Reverse
6 4 2 9 8 3 0 1
Reverse
Reverse.
2 4 6 1 0 3 8 9
Rotated the array 3 times.

K = 2


1 0 3 8 9 2 6 4
4 6 2 9 8 3 0 1
6 4 1 0 3 8 9 2

Code.

1 2 3 4 5 K=2

```
void rotateKtimes(int arr[], int n, int K)
{
    K = K % N
    T.C -> O(N)   reverseInRange(arr, 0, n-1)
    T.C -> O(K):  reverseInRange(arr, 0, K-1)   // Reverse 1st K elements.
    T.C -> O(N-K): reverseInRange(arr, K, n-1)
}
```

0

5 4 3 2 1

$$T.C. \rightarrow O(N + K + N - K)$$

$$= O(N)$$

$$S.C. = O(1)$$

Dynamic arrays.

```
int[] arr = new int[10];  // Static arrays.
```

↓

fixed length.

↓

Cons: You have to declare size before hand.

Some space might get wasted.

Dynamic arrays → automatic resizing.

↓

Size of arrays keeps growing as you add more elements.

(Amortized T.C. of adding into a Dynamic array is O(1).)

Random access and addition of an element at the end is O(1).

Addition / Deletion in middle won't be O(1).

Java → ArrayList ✓

C++ → vector

Python → list ✓

JS → list ✓

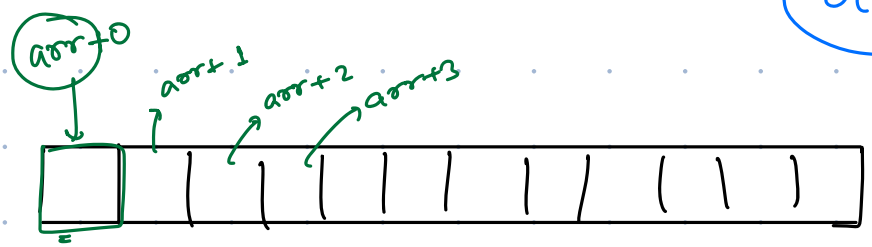
C# → list ✓

Doubt session

i	j	
i = 0	N 1	→ N
i = 1	N 2	→ + N-1
i = 2	N 3	→ + N-2
⋮		
i = N-1	N ..... N	→ + 1

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

$O(N^2)$

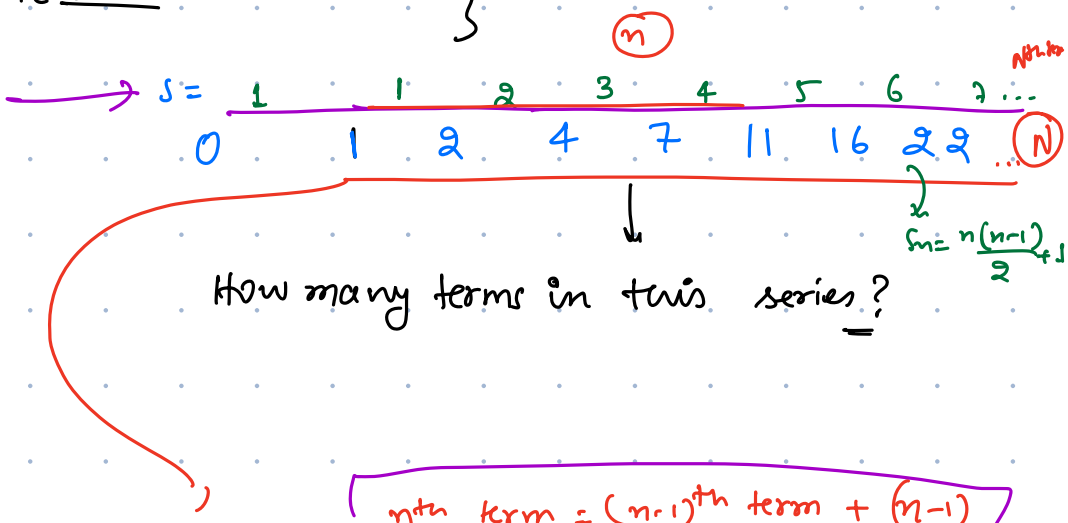


int arr[] = { }

P

```
i=1, s=1
while (s< N)
{
    s=s+i
    i++
}
```

```
public static void main()
{
    =
    =
}
```



S_1	S_2	S_3	S_4	S_5	S_6	S_7
1	2	4	7	11	16	22	30

$$S_n = \frac{n(n-1)}{2} + 1$$

$S_1 = 1$

$S_2 = 1 + 1 = 2$

$S_3 = S_2 + 2 = 4$

$S_4 = S_3 + 3 = 7$

$$S_7 = \frac{7 \times 6}{2} = 21$$

$S_n = S_{n-1} + (n-1)$

$S_{n-1} = S_{n-2} + (n-2)$

$S_n = S_{n-2} + n-2 + n-1$

$S_{n-2} \rightarrow S_{n-3} + n-3$

$S_n = S_{n-3} + n-3 + n-2 + n-1$

$S_n = \vdots$

$$S_2 = \underset{1}{S_1} + 1$$

$$S_n = 1 + 1 + 2 + 3 + 4 + \cdots + n-2 + n-1$$

$$= \frac{(n-1)n}{2} + 1$$

Series is going upto X.

1 2 4 7 11 16 22 X

How many terms are there in this series.?

Let it be n.

If there are n terms in the series.

$$S_n = X = \frac{n(n-1)}{2}$$

$$X = n^2$$

$$n = \sqrt{X}$$

```
( i=0 ; i<Math.pow(2,n) ; i++ )
{
    j=i
    'j>0
     j--
```

i=0	j
0	0
1	1
2	2
3	3
4	4
:	5
:	6
:	:
$2^N - 1$	2^{N-1}

$0+1+2+3+\cdots\cdots 2^{N-1}$

$= 1+2+3+4+\cdots\cdots 2^{N-1}$

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

```
while(a!=b)
{
    if(a>b)
        a=a-b
    else
        b=b-a
}
```

10 3

max(a,b) : worst ✓

gcd(a,b)

10-3=7

7-3=4

4-3=1

3-1=2

2-1=1

a/b

(10/,5)

O(2)

4^{N-1}

4^N 2^{2N}

$(4)^N$

2 ≠

Amortized :