Introduction To Arrays

Space Complexity

↳ max space (worst case) that is utilized at any point of time during running the algo

Ex:-

```
fcen ( int N) {          // 4 bytes   → Not included
    int x;               // 4 bytes
    int y;               // 4 bytes
    long z;              // 8 bytes
}
```

16 bytes

* We don't account for input space because it's not in our control.

Hence, the space occupied = 16 bytes.

Our space is not depending on input, hence we can say it's Constant

∴ Space Complexity = 16 bytes

O(1)

QUIZ 1 :-

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

Space = ~~52~~ + 4*N

Space Complexity = O(N)

Quiz 2

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

Space = ~~16~~ + ~~4N~~ + $8N^2$

Space Complexity = $O(N^2)$

Question:- Find the space complexity of below code?

I

```
int maxAns(int arr[], int N) {
    int ans = arr[0];                    // 4 I
    for (i from 1 to N-1) {              // 4
        ans = max(ans, arr[i]);
    }
    return ans;
}
```

Space = 8 bytes

Space Complexity = O(1)

II

```
int maxAns(int arr[], int N) {
    int ans = arr[0];                    // 4 I
    for (i from 1 to N-1) {              // 4+4
        int x = 5;
        ans = max(ans, arr[i]);
    }
    return ans;
}
```

Space = 12 bytes

SC → O(1)

ARRAYS

↳ Sequential Collection of similar Data Types

Name of array

int arr [N];

data type — Size of Arrays:

⇒ We can access elements like this:-

arr [0], arr [1], ... arr [N-1]

* Indexing Start with 0.

Quiz → 0, N-1

↳ Printing all elements of array.

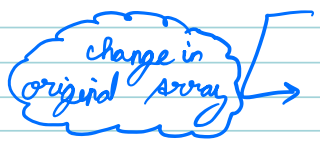
```
void printArray ( int[] arr ) {
    int N = arr.length;
    for ( i = 0; i < N; i++ ) {
        print ( arr[i] );
    }
}
```

TC ⇒ O(N) | SC ⇒ O(1)

Quiz 4

Quiz 5

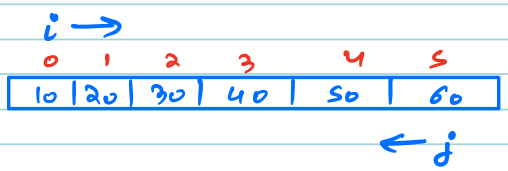
Question :- Given an array of size N, Reverse the entire array.



arr[] : { 1 2 3 4 5 }

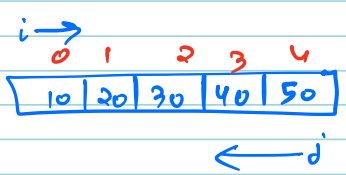
arr[] : { 5 4 3 2 1 }

Even :-



Need to run ⇒ $i <= j$

Odd :-



So I need to run → $i < j$

PSEUDO CODE

```
function reverse ( int [] arr, int N ){
    int i = 0;
    int j = N-1;
    while ( i < j) {
        int temp = arr [i];
        arr [i] = arr [j];
        arr [j] = temp;
        i++;
        j--;
    }
    return arr;
}
```

No. of iterations = $\frac{N}{2}$ (because i<j)

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

Question Given an arr of size 'N' and integers 'l' & 'r'. Reverse the array from 'l' to 'r'.

Ex:- N = 5

arr = { 1 2 3 4 5 }
(0 1 2 3 4)

if, l = 1 & r = 3

arr = { 1 4 3 2 5 }
(0 1 2 3 4)

PSEUDO CODE

```
function reverse ( int [] arr, int N, int l,
                                        int R){
    int i = l;
    int j = R;
    while ( i < j) {
        int temp = arr [i];
        arr [i] = arr [j];
        arr [j] = temp;
        i++;
        j--;
    }
    return arr;
}
```

TC => O(N) SC = O(1)

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

(i.e if K=1, last element will come at first position ....)

N=5 , arr = { 1 2 3 4 5 }

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

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

Brute force

↳ How to rotate 1 time

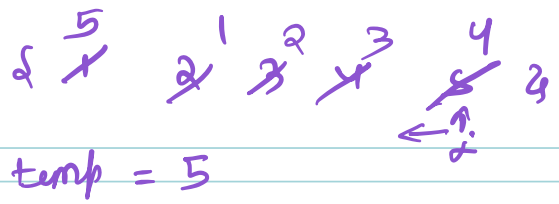
arr = { 1 2 3 4 5 }

1 time arr = { 5 1 2 3 4 }

Pseudo Code :-

```
int temp = arr[N-1];
for(j = N-1; j > 0; j--){
    arr[j] = arr[j-1];
}
arr[0] = temp.
```

DRY RUN :-



★ If we perform same task K times then the above question is solved.

PSEUDO CODE

```
function rotateK ( int arr[], int K, int N){
K →  for (i = 0 ; i < K; i++){
        int temp = arr[N-1];
        for (j = N-1; j > 0; j--){
TC:O(N)     arr[j] = arr[j-1];
        }
        arr[0] = temp.
     }
     return arr;
}
```

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

10 20 30 40 50

10 20 30 40 50

OPTIMIZATION

↳ arr[] = { 10 20 30 40 50 }

Ex:- arr[] = { 10 20 30 40 50 }

K=1 = { 50 10 20 30 40 }

K=2 = { 40 50 10 20 30 }

K=3 = { 30 40 50 10 20 }

K=4 = { 20 30 40 50 10 }

K=5 = { 10 20 30 40 50 }

OBSERVATIONS :-

→ If K=3, you are just pulling last 3 elements at front.

→ So, we want to pull K :-

→ Rotate the entire array.

→ Last K elements are the front, just Reverse the first K elements

→ Reverse the remaining elements.

K = 2, Let Repeat the Step.

arr = { 1 2 3 4 5 }

↓ Reverse entire array

arr = { 5 4 3 2 1 }

↓ Revers the first k elemet

arr = { 4 5 3 2 1 }

↓ Revese the remainig elemet

arr = { 4 5 1 2 3 }

PSEUDO CODE

```
function reverseK( int[] arr, int k, int N){
    //① Revers entire array
        reverse ( arr , 0 , N-1 , N);
    //② Reverse first k elements.
        reverse (arr , 0 , K-1, N) ;
    //③ Reverse the remaining elements
        reverse ( arr, K, N-1 , N);
}
```

(L) EDGE CASE

$K = 7$

Ex:- arr[] = {10 20 30 40 50} K = 5, 10, 15

K = 1 = {50 10 20 30 40} K = 6, 11

K = 2 = {40 50 10 20 30} K = 7, 12

K = 3 = {30 40 50 10 20} K = 8, 13

K = 4 = {20 30 40 50 10} K = 9, 14

K = 5 = {10 20 30 40 50}

OBSERVATION

$K = 0 \;\&\; K = 5$

★ Relation b/w K's

$$K = K \% N$$

PSEUDO CODE

```
function reverseK( int[] arr, int k, int N){
    K = K % N;
    //① Reverse entire array                          N/2 ->
    reverse (arr, 0, N-1, N);
    //② Reverse first K elements.                     K/2 ->
    reverse (arr, 0, K-1, N);
    //③ Reverse the remaining elements                (N-K)/2 ->
    reverse (arr, K, N-1, N);
}
```

TC = Total iteration :- $\frac{N}{2} + \frac{K}{2} + \frac{N-K}{2}$

$= \frac{2N}{2}$

$= N$

$= O(N)$

SC = $O(1)$ ⇒ Best

DYNAMIC ARRAYS

* Biggest Strength over normal array.
  ↳ Automatic resizing.

* We dont need to determine the size

[Behind the scene array is getting used).

STRENGTH

→ Fast lookup
→ variable size.

WEAKNESS (working)

Behind the Scene →

Array = 10 Size
(can store 10 elements)

↓ 11th element.

=> Create Double Size array
=> Copy the old array in this new array
=> Put 11th element.

Array (20)

Complexity → O(N)
(very Expensive)

CONCLUSION :- ① only Some iterations are very Expensive.

2 → 4 → 8 → 16 → 32
↓
64

So, the complexity of insertion in ArrayList amortized O(1)

NEXT CLASS

→ Prefix Sum