Agenda

1) Big O steps reasoning
2) Importance of constraints
3) space complexity
4) Arrays
   - reverse part
   - rotate by k

Big O (iteration count) → TC

i) find iteration count
ii) ignore lower order terms
iii) ignore constant coefficients

} why?

itr → ~~4~~N^2 + ~~3N~~ + ~~1~~

Big O → $O(N^2)$

1) Why we can ignore lower order terms.

itr → $N^2 + 10N$

N	total itr ($N^2 + 10N$)	itr due to lower order term ($10N$)	contribution of lower order terms in total iterations
10	200	100	$\frac{100}{200} \times 100 = 50\%$
10^2	$10^4 + 10^3$	10^3	$\frac{10^3}{10^4+10^3} \times 100 \approx 10\%$
10^4	$10^8 + 10^5$	10^5	$\frac{10^5}{10^8+10^5} \times 100 \approx 0.1\%$

Conclusion : for large input size, contribution of lower order term is almost negligible that's why we can ignore lower order terms.

---

1) Why we can ignore constant coefficients.

	Algo I	Algo II	winner
	$1000N$	N^2	Algo I
Big O →	$O(N)$	$O(N^2)$	

Issues with Big O ?

1) Algo I → $10N$ Algo II → N^2

Algo I is always better than Algo II ✗

Algo I is better than Algo II for large input size

N	Algo I ($10N$)	Algo II (N^2)
5	50	25
10	100	100
100	1000	10000
1000	10000	10^6

2) Algo I → $100N$ Algo II → $3N + 20$

↓ ↓

TC : $O(N)$ TC : $O(N)$

When two have same Big O, then Big O concepts fails to give the comparative analysis.

In this scenario, do comparison based on its count.

Winner → Algo II

* Importance of constraints

online editor ⟶ server

↓

1 sec

(10^7 to 10^8 iterations)

your code will be submitted

itr ⟶ < 10^7 to 10^8 itr

Otherwise we will get TLE

↳ time limit exceeded

Count Factors

Beginner	JSA
$1 <= N <= 10^6$	$1 <= N <= 10^9$
logic → itr : N	logic → itr : N

if (N == 10^6) {
itr → 10^6 ✓
}
↓
Submitted
	if (N == 10^9) {
	itr → 10^9 (TLE)
	}
	Optimised logic, itr → $\sqrt{N}$
	if (N == 10^9) {
	itr → $\sqrt{10^9}$ ≈ $10^{\sim 4}$
	}
	↓
	Submitted

Space complexity

Time complexity : apply Big O on iteration count of logic.

Space complexity : apply Big O on space taken by logic.

↓

Only consider the space taken by the logic and not the input space

int → 4 bytes

long → 8 bytes

```
int solve(int n) {
    long a = 50;
    int b = 6;
    return -1;
}
```

space → 12 bytes

SC → O(1) [constant]

Find the Space Complexity [Big(O)]

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

space → 40 + 4 + 4 + 8 + 4N

= ~~56~~ + 4N

SC → O(N)

Find the Space Complexity [Big(O)] of the below program.

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

space → 4 + 4 + 8 + 4N + 8N²

= ~~16~~ + ~~4N~~ + ~~8~~N²

SC → O(N²)

Big O

TC	SC
i) find itr count of logic	i) find space taken by logic
ii) ignore lower order terms	ii) ignore lower order terms
iii) ignore constant coefficient	iii) ignore constant coefficient

---

⟶ A.length = N

$1 <= N <= 10^5$

$1 <= A[i] <= 10^9$

```
int maxOfArray(int[] A) {
    int max = A[0];
    for (int i=1; i<A.length; i++) {
        if (A[i] > max) {
            max = A[i];
        }
    }
    return max;
}
```

TC	SC
itr: N	Space: 8 Bytes
TC → O(N)	SC → O(1)
if(N == 10^5)	
itr: 10^5	
↓	
Submitted	

Arrays

```
int [] arr = new int [5];
arr[2] = 50;
arr[3] = 90;
```

arr:

0	0	50	90	0
0	1	2	3	4

What will be the indices of the first and last elements of an array of size N?

N sized arr, index → 0 to N-1

What is the time complexity of accessing element at the ith index in an array of size N?

```
int [] arr = {10, 20, 30, 40};
```

10	20	30	40
0	1	2	3

```
sopln(arr[2]); → 30
```

↳ TC: O(1)

```
int arr[5] = {5,-4,8,9,10};
```

Which of the following statements correctly prints the sum of the 1st and 5th elements of the array?

[5 -4 8 9 10]
0 1 2 3 4

ans → 15

Q. Given an Array. Reverse it.

arr = [10 20 30 40 50 60]
0 1 2 3 4 5

↓ rev

[60 50 40 30 20 10]
0 1 2 3 4 5

60 50 40 30 20 10
[~~10~~ ~~20~~ ~~30~~ ~~40~~ ~~50~~ ~~60~~]
0 1 2 3 4 5

j i [STOP]

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

i
j

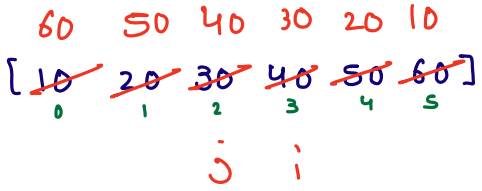
[STOP]

```
void reverse (int [ ] A ) {
    int i=0, j = A.length - 1 ;
    while (i<j) {
        || swap A[i] & A[j]
        int temp = A[i];
        A[i] = A[j];
        A[j] = temp;
        i++;
        j--;
    }
}
```

itr → $\frac{N}{2}$

TC : O(N)

SC : O(1)



i=0, j=5 1st time

i=1, j=4 2nd time

i=2, j=3 3rd time

Q. Given an array, rotate it right to left K times.

Constraints

$1 \le n \le 10^5$

$1 \le K \le 10^5$

don't create an extra array.

A[] = 1 2 3 4 5 K = 3
0 1 2 3 4

↓ 1st

5 1 2 3 4
0 1 2 3 4

↓ 2nd

4 5 1 2 3
0 1 2 3 4

↓ 3rd

3 4 5 1 2
0 1 2 3 4

K times [Right shift

```
int[] rotation(int[] A, int K) {
    int n = A.length;
    for (int t = 1; t <= K; t++) {
        int temp = A[n-1];
        for (int i = n-2; i >= 0; i--) {
            A[i+1] = A[i];
        }
        A[0] = temp;
    }
    return A;
}
```

1 2 3 4 5 K = 3
0 1 2 3 4

t = 1: 5 1 2 3 4 (1 2 3 4 5 struck) temp = 5
0 1 2 3 4

t = 2: 4 5 1 2 3 (5 1 2 3 4 struck) temp = 4
0 1 2 3 4

t = 3: 3 4 5 1 2 (4 5 1 2 3 struck) temp = 3
0 1 2 3 4

```
int[] rotation(int[] A, int K) {
    int n = A.length;
    for (int t = 1; t <= K; t++) {
        int temp = A[n-1];
        for (int i = n-2; i >= 0; i--) {
            A[i+1] = A[i];
        }
        A[0] = temp;
    }
    return A;
}
```

TC: $O(N*K)$

SC: $O(1)$

Constraints

$1 \le n \le 10^5$

$1 \le K \le 10^5$

if ($n == 10^5$, $K == 10^5$)

itr: $N*K$

$10^5 * 10^5 = 10^{10}$

↓

TLE