Agenda

1) TLE ⇒ Time limit Exceed
2) Importance of Constraint
3) Steps of writing code
4) Space Complexity
5) Intro. to Arrays
6) Reverse the Array
7) Rotate Array
8) Dynamic Arrays.

If $2^K = N$, $K = ??$

$$2^K = N$$

Take $\log_2$ both sides

$$\cancel{\log_2 2^K} = \log N$$

$$K = \log_2 N$$

Iterations ⇒ $\cancel{4}N^2 + \cancel{3N} + \cancel{1}$

$$= O(N^2)$$

Story

Amazon $\Rightarrow$ Online Test

$\Downarrow$

30 mins / problems.

2 problems $\Rightarrow$ 1 hour

$$\underbrace{1^{st} \text{ question} \Rightarrow \text{Solution} \Rightarrow \text{Code} \Rightarrow \text{Submit}}_{20 \text{ mins}}$$

$\Downarrow$

TLE (Time limit Exceeded)

$\Downarrow$

$$\underbrace{2^{nd} \text{ sol}^n \text{ (More optimal)} \Rightarrow \text{Code} \Rightarrow \text{Submit}}_{20 \text{ mins}}$$

$\Downarrow$

TLE (Time limit Exceeded)

$\Downarrow$

3rd Time $\Rightarrow$ TLE

Online Editors & TLE

Codechef / InterviewBit / LeetCode

⇓

⇒ Run the code on an online server with 1 GHz of processing power.

⇒ Code execution time is 1 sec

Information ⇒ A 1 GHz machine can perform 10^9 instructions per second.

⇓

(Single command to compiler)

Ex Code

```
int countfactors (N) {
    int count = 0;          +1
                +1          (+1)×N
N ← for (i = 1;  i ≤ N;  i++) {      ⟶ i = i + 1
                                +2×N
        if (N % i == 0) {  → +2×N
            count = count + 1;  → +2×N
        }
```

```
    }

    return count; + 1
}
```

Instructions inside loop $\simeq$ $7 \times N$ instructions

$\Rightarrow$ In one iteration of the loop, we have approximately 7 instructions.

Small code $\simeq$ 10 instructions.

Large code $\simeq$ 100 instructions

Range of instructions in 1 iteration $\simeq [10^1, 10^2]$

We can execute 10^9 instructions.

Small code $= \dfrac{10^9}{10} = 10^8$ iterations.

Large Code $= \dfrac{10^9}{10^2} = 10^7$ iterations

1 second $\simeq$ $[10^7, 10^8]$ max.

Importance of Constraints

Eg 1) $1 \le N \le 10^5$

Q ⇒ Which Big(O) notation will work.

Complexity	Iterations	Works?
N^3	$(10^5)^3 = 10^{15}$	NO
$N^2 \log N$	$(10^5)^2 \log(10^5)$ $\simeq 10^{10} \times$ __	NO
N^2	$(10^5)^2 = 10^{10}$	NO
$N \log(N)$	$(10^5)(\log 10^5)$	Yes

3.2×10^6 $10^5 \times (\sim 32)$

Max int $\Rightarrow$ $\underline{2^{32}-1}$

$$\log(2^{32}) = \underline{32}$$

How to approach a Problem ??

DSA $\Rightarrow$ Pen & Paper

Code $\Rightarrow$ Last Step (Code Editor)

1) Read the question, constraints & Examples.
2) Formulate the idea / logic / sol^n on paper.
3) Verify the correctness of logic
   (Dry run on Examples)
4) Mentally develop a pseudo code or rough idea of loops.
   (Initially write pseudo code)

5) figure out the time Complexity (Big O)

6) Assess whether this T.C. will be accepted or not.
($[10^7, 10^8]$ max iterations)

7) Re-evaluate / Optimise sol^n

8) Code once confident on the logic / sol^n.

Space Complexity.

⇒ Max auxiliary (Extra) space (Worst Case) that is utilised at any point of time during the execution of the program / algorithm / code.

⇒ It is calculated in terms of rate of growth of space ⇒ Big(O)

NOTE :- The Input & Expected Output are not considered while calculating the Space complexity.

Ex Codes

1)

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

Total space = 16 Bytes.

Constant Space Complexity. = O(1)

Ex 2

```
func (int N) {
    int arr[10];     // 4x10 = 40B
    int x;           // 4B
    int y;           // 4B
    long z;          // 8B
    int arr2[N];     // 4xN = 4NB
}
```

Total = ~~56~~ + ~~4~~N

S.C. = O(N)

```
func (int N) {
    int x = N          // 4B
    int y = x*x;       // 4B
    long z = x + y;    // 8B
    int arr2[N];       // 4xN = 4NB
    long l[N][N]       // 8x(N^2) = 8N^2 B
}
```

Total = $\cancel{16} + \cancel{4N} + \cancel{8}N^2$

S.C. = $O(N^2)$

Ex 3

```
                                  I/P
function maxArray (int arr[], int N) {
    int ans = arr[0];
    for( i=1; i<N; i++) {
        ans = max (ans, arr[i]);
    }
    return ans;
}
```

T.C. = $O(N)$

S.C. = $O(1)$

Introduction to Arrays.

⇒ Collection of same type of data

⇒ Datatype can be anything

Declaration

```
int    arr [N] ;
```

↓ Datatype ↓ name ↓ size

// H.W. ⇒ Check syntax for your preferred language.

```
int    arr [5]
```

Position ⇒ Index

0	1	2	3	4

NOTE :- for an array of size N.

Indexes ⇒ [0, N-1]

Advantage

⇒ O(1) random access.

Access an element on index i $\Rightarrow$ $\underline{arr[i]}$

$\downarrow$

T.C. = $O(1)$

Q Code to print all elements of a given array of size N

```
function printArray (int arr[], int N) {
    for (int i=0; i<N; i++) {
        print (arr[i]);
    }
}
```

Q

	0	1	2	3	4
arr[5] =	5,	-4,	8,	9,	10
	1st	2nd	3rd	4th	5th

Ans = arr[0] + arr[4]

Q Given an integer array of size N. Reverse the entire array. (S.C. = $O(1)$)

Eg

$N = 5$

$$\begin{array}{cccccc} & 0 & 1 & 2 & 3 & 4 \\ arr = & \{1, & 2, & 3, & 4, & 5\} \end{array}$$

O/P $\Rightarrow$ $\{5, 4, 3, 2, 1\}$

Sol^n 1) Insert elements into a new array in reverse order.

S.C. = $O(N)$ ✗

2)

$$\begin{array}{cccccc} & 0 & 1 & 2 & 3 & 4 \\ arr[5] = & \{7, & 1, & 3, & 2, & 4\} \end{array}$$

$$\begin{array}{cccccc} & 0 & 1 & 2 & 3 & 4 \\ \text{Reverse arr} = & \{4, & 2, & 3, & 1, & 7\} \end{array}$$

$i \longrightarrow N-1-i$

Initial Index	Final Index
0	4 $(N-1)$
1	3 $(N-2)$
2	2 $(N-3)$
3	1
4	0

Code

```
for (i = 0; i < N; i++) {
    // i → N-1-i;
    arr[N-1-i] = arr[i];
}
```

Dry Run

N = 5

$$arr[5] = \{ \overset{0}{\cancel{7}}, \overset{1}{\cancel{1}}, \overset{2}{\cancel{3}}, \overset{3}{\cancel{2}}, \overset{4}{\cancel{4}} \}$$

$$\{ 7, 1, 2, 1, 7 \}$$

i	(N-1-i)	(5-1-i) = (4-i)
0	4	
1	3	
2	2	
3	1	
4	0	

Not correct

3) Correct our sol^n

$i \longrightarrow N-1-i$

Initial Index	Final Index	
0	4	(N-1)
1	3	(N-2)
2	2	(N-3)
3	1	⋮
4	0	(0)

$4 \rightarrow 0$ & $0 \rightarrow 4$ (Swap)

Code

```
for (i = 0; i < N; i++) {
        temp = arr[N-1-i];
        arr[N-1-i] = arr[i];
        arr[i] = temp;
}
```

	0	1	2	3	4
arr[5] =	~~4~~	~~2~~	~~3~~	~~1~~	~~7~~
	7	1	2	4	

	i	$N-i-1$ (j)
✓	0	4
+1 ✓	1	3
+1 ✓	2	2
+1 ✓	3	1
+1 ✓	4	0

-1 -1 -1 -1

$i = N/2$

$$arr[6] = \{\overset{0}{1}, \overset{1}{2}, \overset{2}{3}, \overset{3}{4}, \overset{4}{5}, \overset{5}{6}\}$$

$N/2 = \underline{3}$

Code

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

```
for (i = 0;  i < N/2 ;  i++) {
        temp = arr[N-1-i];
        arr[N-1-i] = arr[i];
        arr[i] = temp;
}
```

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

One more way to write same code

```
i = 0,  j = N-1

while (i < j) {
    temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    i++;
    j--;
```

Reverse a part of array from any i to any j

}

Next Class ⇒ Monday.

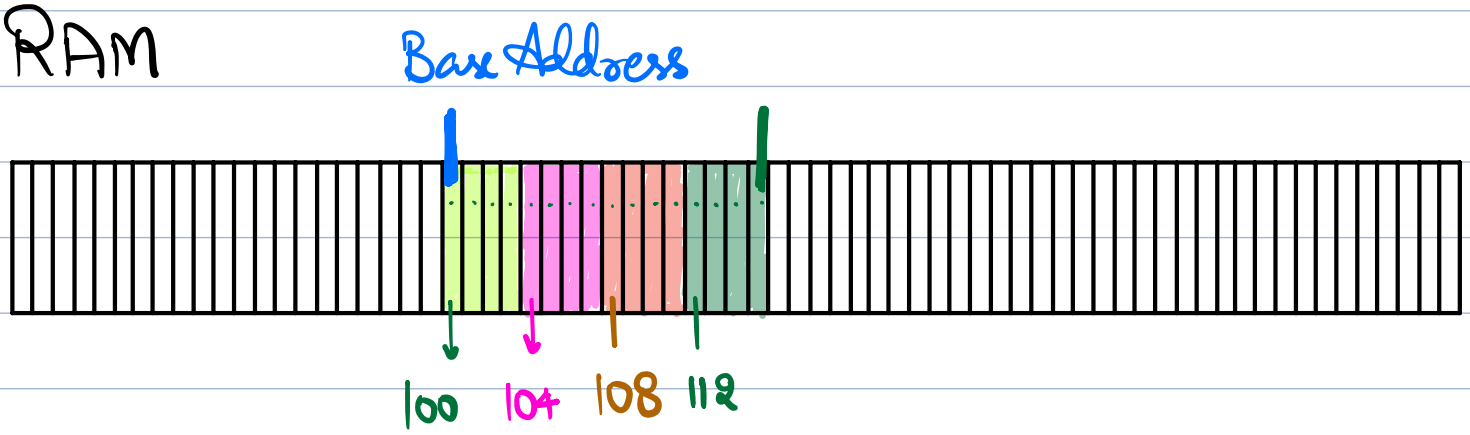
1) Rotate array.
2) Dynamic array.
3) Prefix Sum.

Array.

→ Homogenity (Same type of data)

→ Contiguous memory allocation.

RAM



! How many chunks will an it consume??

4

arr[4] ⇒ 4×4 = 16 chunks

arr[i]

Base Address

index	Address
0	100 (Base)
1	100 + 4 = 104
2	100 + 4×2 = 108
3	100 + 4×3

Address of ith element = Base Address + (Size of Datatype in Bytes.) × i

" + 4 × i

O(1)

10 (Initial capacity)

Load factor = 0.8

Added 8 elements

9th element.

hook for double the memory (90)

Copy the 9 elements

Doubt

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

[1, i]

↓

i

i	$j \rightarrow$	# iter.
0		0
1		1
2		2
3		3
$\vdots$		$\vdots$
2^N-1		2^N-1

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

$$\underline{2^N \times 2^N} \quad - \cancel{2^N}$$

$$\downarrow$$

$$2^{N+N} = 2^{2N} = (2^2)^N = \underline{\underline{4^N}}$$