Topics Already Covered

- → Getting Started
- → Arrays & Strings - Level 1
- → Recursion Basics
- → Time & Space Complexity Analysis
- → Searching & Sorting (15 lectures)

Companies

① Service Based

→ TCS Digital

→ Infosys Power Programmer

→ Wipro ↑↑

(i) Arrays & Strings

(ii) Aptitude

② Product Based - Startups (MERN, MEAN, Data Science) (Backup)

→ DSA (Basic to Intermediate)

→ Web Dev / Mobile Dev (Intermediate +)

(3) Product Based → MNC's (Amazon, Microsoft, salesforce, ...)

- DSA (75-85%)
  - Sheet → Interview
  - OA → Leetcode contests or Competitive
- Core Subjects
  - OOPS, OS, DBMS, CN ⇒ Theory
  - Projects

(3.1) Some Product Based Companies

↳ Low level Design

eg Sprinklr, Uber, Walmart, Flipkart, Phonepe, Ola, Amazon (Bar raiser+), HFT Companies, Google (Googlyness), Salesforce

(3.2) Fin-tech Based companies

↳ Goldman Sachs, Morgan Stanley, JP morgan,

↳ Aptitude (maths 11th-12th → geometry, Algebra, Calculus)

(3.3) High Frequency Trading

eg Tower Research, Rippling, Qnance, etc ...

Highest packages in India

10-12 hours daily

LLD + HLD + DSA

Time & Space Complexity Analysis

Ideal Algorithm → Time ↓ && Space ↓
(Runtime or Execution time)
(RAM or main memory ↳ Process or CPU)

Tradeoff → Time ↓ Space ↑ ✓ it is preferred
→ Time ↑ Space ↓

(1) Runtime Analysis

Machine 1
(Windows i3)

Machine 2
(Macbook m2)

same code
Runtime ↑

same code
Runtime ↓

↳ Machine Dependent

LeetCode

code → cloud
code → cloud

] same Super computer]

Concurrent submissions

(2) Asymptotic Analysis

→ Machine independent

→ Rate of growth of runtime with respect to input size

(3) Amortized Analysis

eg

Machine 1 (win i3)

Machine 2 (mac m2)

Input size (N)

N = 10 $\xrightarrow{*10}$ N = 100 $\xrightarrow{*10}$ N = 1000

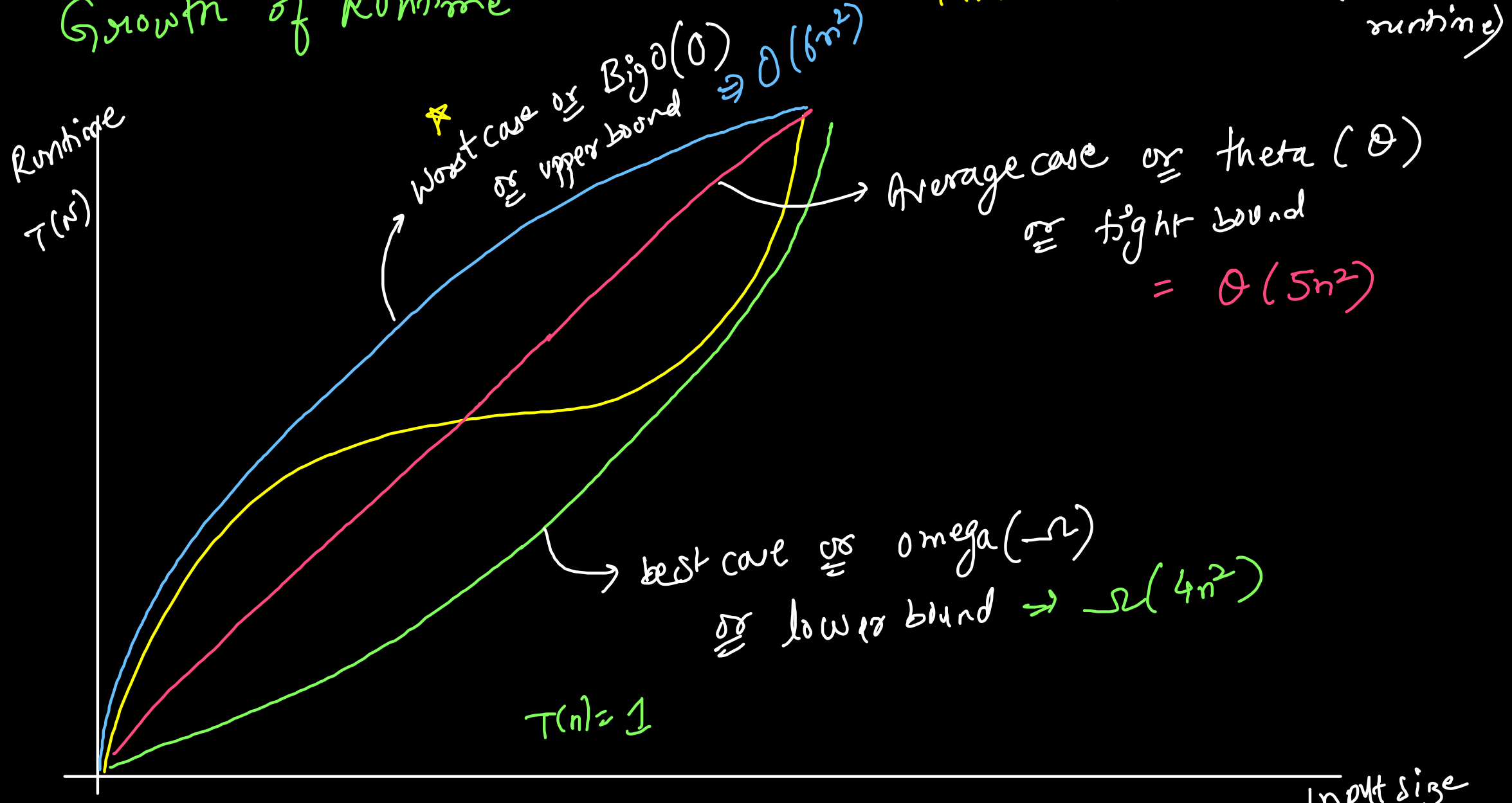
10 ms $\xrightarrow{*10}$ 100 ms $\xrightarrow{*10}$ 1000 ms

5 ms $\xrightarrow{*10}$ 50 ms $\xrightarrow{*10}$ 500 ms

Algorithm Runtime $\propto$ Input size

Growth of Runtime

$T(n) = 5n^2 + 3n + 6$ (actual running)



Worst case

Actual runtime

$O(n^2)$

$6n^2 > 5n^2 + 3n + 6$

$n = 1$: $6 * 1^2 = 6 \not> 5*1^2 + 3*1 + 1 = 14$

$n = 2$: $6 * 2^2 = 24 \not> 5*2^2 + 3*2 + 6 = 32$

$n = 3$: $6 * 3^2 = 54 \not> 5*3^2 + 3*3 + 6 = 60$

$n = 4$: $\not>$

$n = 5$: $\not>$ → we ignore smaller input size trend!

$n = 6, 7, 8, 9, 10, \ldots \infty$

$6n^2 > 5n^2 + 3n + 6$

input size

$O(n) \simeq O(3n) < O(n^2)$

100	100 ms	300 ms	10000 ms
↓ *n	↓ *10	↓ *10	↓ *100
1000	1000 ms	3000 ms	1000000 ms

we ignore constant values

$O(3n) = O(n)$

Runtime

$4n^3 + 10n^2 + 5n + 6$

$100n + 200n + 300n = 600n$

$200 + 400 + 1000 = 1600n^0$

$n^5 + n^4 * n^3 + n^2 * n^6 = n^5 + n^7 + n^8$

$n^5 + n^4 + 2^n + n^1 + 100$

Worst Case

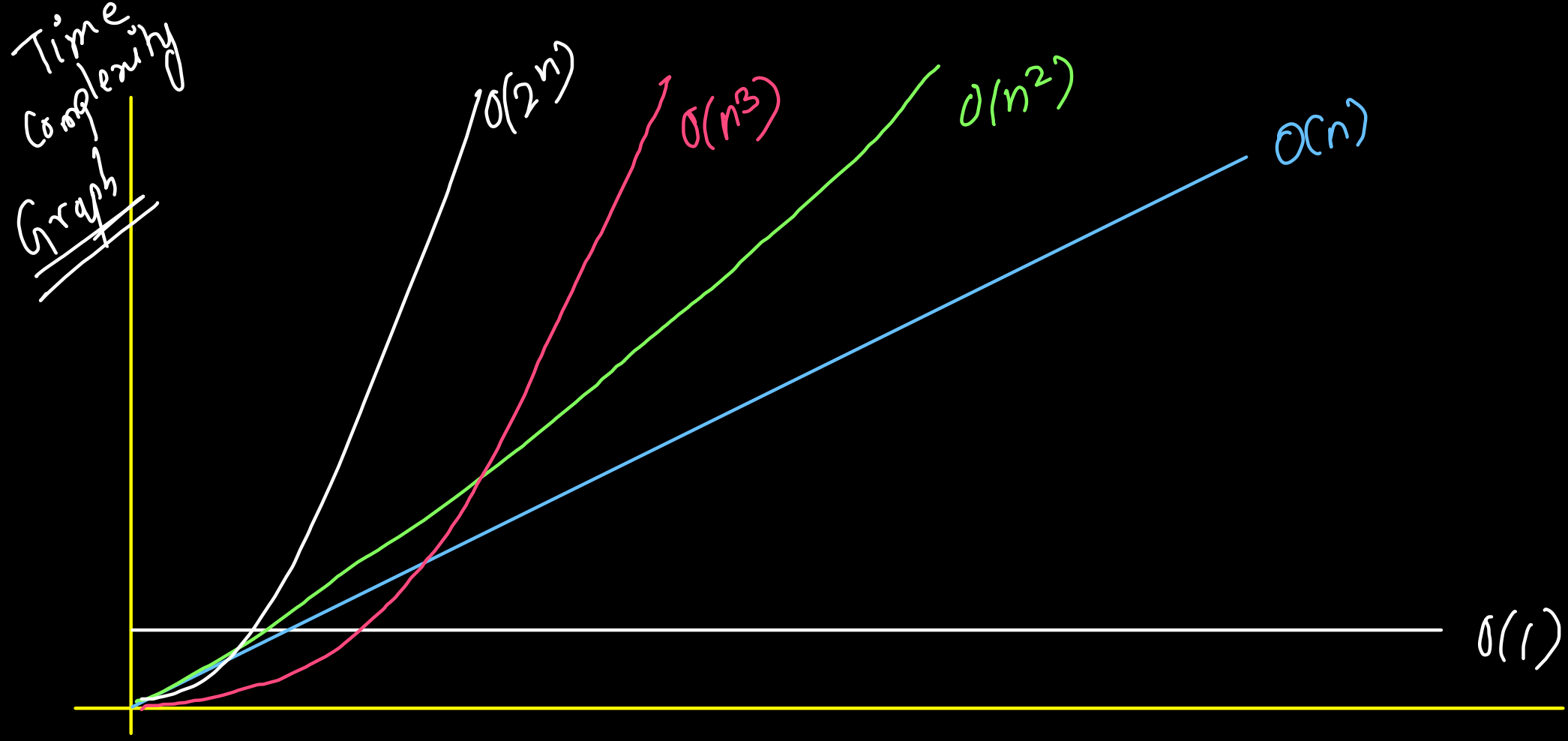
$O(n^3)$ cubic

$O(n)$ linear

$O(n^0) = O(1)$ constant

$O(n^8)$

$O(2^n)$ exponential


Time Complexity Graph
O(2^n)
O(n^3)
O(n^2)
O(n)
O(1)

Time Complexity Trend

$O(1) < O(\log_2 N) < O(\sqrt{N}) < O(N) < O(N * \log N)$

constant — logarithmic — root = $N^{1/2}$ — linear

$< O(n^2) < O(n^3) <<< O(2^N) < O(3^N) < O(N^N)$ or $O(N!)$

quadratic — cubic — polynomial — exponential

Constant Time Complexity

O(1)

(1) System.out.print/println (Primitive Variables)

(2) Scanner, scn, next(), nextLine(), nextInt() ....

(3) Operators (arithmetic (+, -, /, %, *), logical (||, &&, !, ^), comparison (<, >, <=, >=, ==), bitwise (|, &, ~, ^, <<, >>))

(4) If else statement, switch case

(5) Assignment operation (=)

(6) Ternary operator (()? :)

(7) Function call & Return statement

(8) Array/String indexing

Logarithmic Time Complexity

$O(1) \simeq O(\log_2 N) \simeq O(\log_3 N) \simeq O(\log_{10} N)$

$N = 1$	1	0	0	0
$N = 10$	1	3	2	1
$N = 100 = 10^2$	1	6	4	2
$N = 10^3$	1	12	8	4
$N = 10^{10}$ (int_max)	1	48	32	16
$N = 10^{18}$ (long_max)	1	60	26	18

→ $\log_a a = 1$

→ $\log_x 1 = 0$

→ $\log_x a^b = b \log_x a$

→ $\log_x (a \times b) = \log_x a + \log_x b$

$O(\log_2 N)$
- → power funcn (exponential)
- binary search
- divide & conquer based

$O(\log_{10} x)$

Digit Traversal

```java
public int reverse(int x) {
    int rev = 0;

    while(x != 0) {
        int digit = x % 10;

        if((rev > Integer.MAX_VALUE/10) || (rev < Integer.MIN_VALUE/10))
            return 0;

        rev = (rev * 10) + digit;
        x /= 10;
    }

    return rev;
}
```

$\log_{10} N$ = Number of digits in decimal no

10^{18} = 19 digits

$\log_{10} 10^{18} = 18$

$10^2 = 100$ → 2 digits + 1

$10^5 = 100000$ → 5 digits + 1

For Loops Time Complexity

(1)

```
for(int idx = 1; idx <= n; idx++){
    System.out.print(idx + " ");
}
```

$n = 5$

idx = 1 — k ms

idx = 2 — k ms

idx = 3 — k ms

idx = 4 — k ms

idx = 5 — k ms

$n * O(1) = O(n)$

5k ms = n * k ms

$\Rightarrow O(n)$

linear (Polynomial)

(2)

```
int n = scn.nextInt();

for(int idx = 1; idx <= n; idx++){
    System.out.print(idx + " ");
}

int m = scn.nextInt();
for(int idx = 1; idx <= m; idx++){
    System.out.print(idx + " ");
}
```

O(n)

O(m)

independent

↳ Time Complexity adds up

↓

They are not nested

linear

O(n) + O(m)

eg: merge 2 sorted arrays

③ Nested loops dependent

```java
int n = scn.nextInt();

for(int i = 1; i <= n; i++){
    for(int j = 1; j <= n; j++){
        System.out.print(i + " " + j);
    }
}
```

O(n²) quadratic

$N_{max} = 10^4$

n = 4

i = 1 + i = 2 + i = 3 + i = 4

j = 1, 2, 3, 4 (k ms) → nk

j = 1, 2, 3, 4 (k ms) → nk

j = 1, 2, ... 4 (k + k + ... k n times = nk)

= nk + nk + nk ---- n times = n^2k (k + k + ... k n times = nk)

(4)

```
int n = scn.nextInt();
int m = scn.nextInt();

for(int i = 1; i <= n; i++){
    for(int j = 1; j <= m; j++){
        System.out.print(i + " " + j);
    }
}
```

$n = 5, \; m = 3$

$i = 1$ $\quad j = 1,2,3 \Rightarrow O(m)$

$+$

$i = 2$ $\quad j = 1,2,3 \Rightarrow O(m)$

$+$

$i = 3$ $\quad j = 1,2,3 \Rightarrow O(m)$

$+$

$i = 4$ $\quad j = 1,2,3 \Rightarrow O(m)$

$+$

$i = 5$ $\quad j = 1,2,3 \Rightarrow O(m)$

$= m + m + m \cdots n$ times

$= O(m * n)$ quadratic

(5)

```
int n = scn.nextInt();

for(int i = 1; i <= n; i++){
    for(int j = 1; j <= i; j++){
        System.out.print(i + " " + j);
    }
}
```

$n = 5$

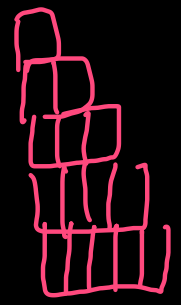
$i = 1$ $\quad j = 1 \Rightarrow k$

$i = 2$ $\quad j = 1,2 \Rightarrow 2k$

$i = 3$ $\quad j = 1,2,3 \Rightarrow 3k$

$i = 4$ $\quad j = 1,2,3,4 \Rightarrow 4k$

$i = 5$ $\quad j = 1,2,3,4,5 \Rightarrow 5k$



$k + 2k + 3k + \cdots nk$

$= k(1 + 2 + 3 + \cdots n)$

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

$= \frac{k}{2}n^2 + \frac{k}{2}n \Rightarrow O(n^2)$

(6)

```
int n = scn.nextInt();

for(int i = 1; i <= n; i++){
    for(int j = i; j <= n; j++){
        System.out.print(i + " " + j);
    }
}
```

$n = 5$

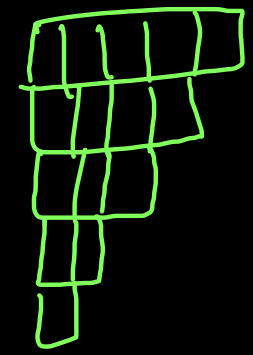
$i = 1$ $\quad j = 1, 2, 3, 4, 5$ — 5k

$i = 2$ $\quad j = 2, 3, 4, 5$ — 4k

$i = 3$ $\quad j = 3, 4, 5$ — 3k

$i = 4$ $\quad j = 4, 5$ — 2k

$i = 5$ $\quad j = 5$ — k



$k(1 + 2 + \cdots n)$

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

$= O(n^2)$

n = 10

(7)

```
for(int i = 1; i < n; i++){
    System.out.print(i + " " );
}
```

i = 1, 2, ... 9 ⇒ 9k ⇒ (n-1)k
= nk - k
= O(n)

(8)

```
for(int i = 1; i <= n / 2; i++){
    System.out.print(i + " " );
}
```

i = 1, 2, 3 ... n/2 = 5k = n/2
= O(n)

n is +ve integer {≥ 1}

```java
int n = scn.nextInt();

for(int idx = n; idx <= 0; idx++){
    System.out.println(idx);
}

for(int idx = n; idx >= 0; idx++){
    System.out.println(idx);
}
```

(9)

(10)

n = 5

idx = 5 5 ≤ 0 false

O(1) Constant

n = 5

idx = 5 5 ≥ 0

idx = 6 6 ≥ 0

idx = 7 7 ≥ 0

8 8 ≥ 0

⋮

+∞ ∞ ≥ 0

Runtime Error
↳ Crash

⇓

infinite loop

O(root N) ⇒ check No is Prime

```java
int n = scn.nextInt();

for(int idx = 1; idx * idx <= n; idx++){
    System.out.println(idx);
}
```

(11)

(or)

```java
int root = (int)Math.sqrt(n);
for(int idx = 1; idx <= root; idx++){
    System.out.println(idx);
}
```

n = 16

idx = 1 → 1*1 ≤ 16 ✓
idx = 2 → 2*2 ≤ 16 ✓
idx = 3 → 3*3 ≤ 16 ✓
idx = 4 → 4*4 ≤ 16 ✓
idx = 5 → 5*5 > 16 ✗

16 → 4 times

n = 49

idx = 1 → 1*1 ≤ 49
idx = 2 → 2*2 ≤ 49
idx = 3 → 3*3 ≤ 49
idx = 4 → 4*4 ≤ 49
idx = 5 → 5*5 ≤ 49
idx = 6 → 6*6 ≤ 49
idx = 7 → 7*7 ≤ 49

49 → 7 times

O(√n) = 10^16
O(n) = 10^8
O(n^2) = 10^4

int
-2^{31} to $2^{31}-1$
10^9

long
-2^{63} to $2^{63}-1$
10^{16}

$O(\log_2 N)$: Power-Logarithmic

```
int n = scn.nextInt();

for(int idx = 1; idx <= n; idx = idx * 2){
    System.out.println(idx);
}
```

12

$n = 16 = 2^4 \Rightarrow 4 = \log_2 16$

idx = 1	$1 \le 16$ ✓
idx = 2	$2 \le 16$ ✓
idx = 4	$4 \le 16$ ✓
idx = 8	$8 \le 16$ ✓
idx = 16	$16 \le 16$ ✓

16 → 5 times

$n = 49 (= 64 = 2^6) \Rightarrow 6 = \log_2 64$

idx = 1	$1 \le 49$
idx = 2	$2 \le 49$
idx = 4	$4 \le 49$
idx = 8	$8 \le 49$
idx = 16	$16 \le 49$
idx = 32	$32 \le 49$

49 → 6 times

$n = 100 (= 128 = 2^7) \Rightarrow 7 = \log_2 128$

idx = 1	$1 \le 100$
idx = 2	$2 \le 100$
idx = 4	$4 \le 100$
idx = 8	$8 \le 100$
idx = 16	$16 \le 100$
idx = 32	$32 \le 100$
idx = 64	$64 \le 100$

100 → 7 times

$n = 1024 = 2^{10}$

↓

10 times

$10 = \log_2 1024$

$O(\log_2 N) \approx O(1)$

$\log_2 2^{30} = 30$

$N = 10^9/10$

$\log_2 N = 30$

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

$2^5 = 32$

$\log_2 1 = 0$

$\log_2 2 = 1$

$\log_2 4 = \log_2(2^2) = 2 * \log_2 2 = 2$

$\log_2 8 = \log_2(2^3) = 3 * \log_2 2 = 3$

$\log_2 16 = 4$

$\log_3 32 = 5$

13

```
for(int idx = n; idx >= 1; idx = idx / 2){
    System.out.println(idx);
}
```

$\Rightarrow O(\log_2 N)$

n = 100

100 / 1 ✓
50 / 1 ✓
25 / 1 ✓
12 / 1 ✓
6 / 1 ✓
3 / 1 ✓
1 / 1 ✓
0 < 1 ✗

100 = 7 times

n = 1000 ≒ 1024 = 2^{10}

1000 / 1
500 / 1
250 / 1
125 / 1
62 / 1
31 / 1
15 / 1
7 / 1

3 / 1
1 / 1
0 / 1

10 times

14)
```
for(int idx = 0; idx * idx <= n; idx++){
    System.out.println(idx);
}
```
$O(\sqrt{N})$

15)
```
for(int idx = 0; idx <= n; idx = idx * 2){
    System.out.println(idx);
}
```
idx = 0, 0*2, 0*2, ... infinite
runtime error

16)
```
for(int idx = 1; idx <= n; idx = idx * 3){
    System.out.println(idx);
}
```
$O(\log_3 N)$

17)
```
for(int idx = 1; idx <= n; idx = idx * 10){
    System.out.println(idx);
}
```
$\log_{10} N$

18)
```
for(int idx = 1; idx <= n; idx = idx + 2){
    System.out.println(idx);
}
```
$\frac{n}{2} = O(n/2)$

19)
```
for(int idx = 1; idx <= n; idx = idx + 3){
    System.out.println(idx);
}
```
$\frac{n}{3} = O(n)$

20)
```
for(int idx = 1; idx <= n; idx++){
    for(int j = 0; j < 5; j++){
        System.out.println(idx);
    }
}
```
$\rightarrow O(5) = O(1)$

$\rightarrow n * O(1) = O(n)$

21)
```
for(int idx = 1; idx <= n; idx++){
    for(int j = 1; j <= n; j = j * 2){

    }
}
```
$n \rightarrow \log_2 N$

$= O(n * \log_2 N)$

$n = 10$
idx = 1, 3, 5, 7, 9

$n = 20$
idx = 1, 3, 5, 7, 9, 11, 13, 15, 17, 19

$n/2$ times
$\Rightarrow O(n)$

Weekend

Sat & Sun

9 AM - 12 PM

Weekdays

Monday, Wed, Thursday

10 PM - 12 AM

Schedule for DSA classes 2.0

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();

    // (1 << n) = 2 ^ n
    for (int idx = 1; idx <= (1 << n); idx++) {
        System.out.print(idx + " ");
    }
}
```

$O(2^n)$

$n = 5$

$1, 2, 3 \ldots\ldots 2^5 = 32$

$n = 4$

$1 << 4 = 2^4 = 16$

$1 << 0$ 0 0 0 0 1

$1 << 1$ 0 0 0 0 1 0

$1 << 2$ 0 0 0 1 0 0

$1 << 3$ 0 0 1 0 0 0

$1 << 4$ 0 1 0 0 0 0

2^0

2^1

2^2

$2^3 = 8$

$2^4 = 16$

1 2 3 4 5 6

7 8 9 10 11 12

13 14 15 16K

$16K = 2^{4K}$

$\Rightarrow O(2^n)$

```java
// (1 << n) = 2 ^ n
int max = (1 << n);

for (int idx = 1; idx <= max; idx++) {
    System.out.print(idx + " ");
}

System.out.println();
for (int idx = 1; idx <= max; idx = idx * 2) {
    System.out.print(idx + " ");
}
```

$O(2^n)$

$O(n)$

$n = 4$

$max = 2^4 = 16$

idx max

$1 \leq 16$ ✓

$2 \leq 16$ ✓

$4 \leq 16$ ✓

$8 \leq 16$ ✓

$16 \leq 16$ ✓

$32 > 16$

$n = 10$

$max = 2^{10}$

$\rightarrow 2^1 \rightarrow 2^2 \cdots \rightarrow 2^{10}$

11 times

Leetcode

1) Problem Statement

2) Input & Output Format

3) Examples

4) Constraints

Online Judges

↓

1 sec ↴

10^8 cycles/sec

↓

10^8 operations/sec

$N = 10^8$ ↓ *10

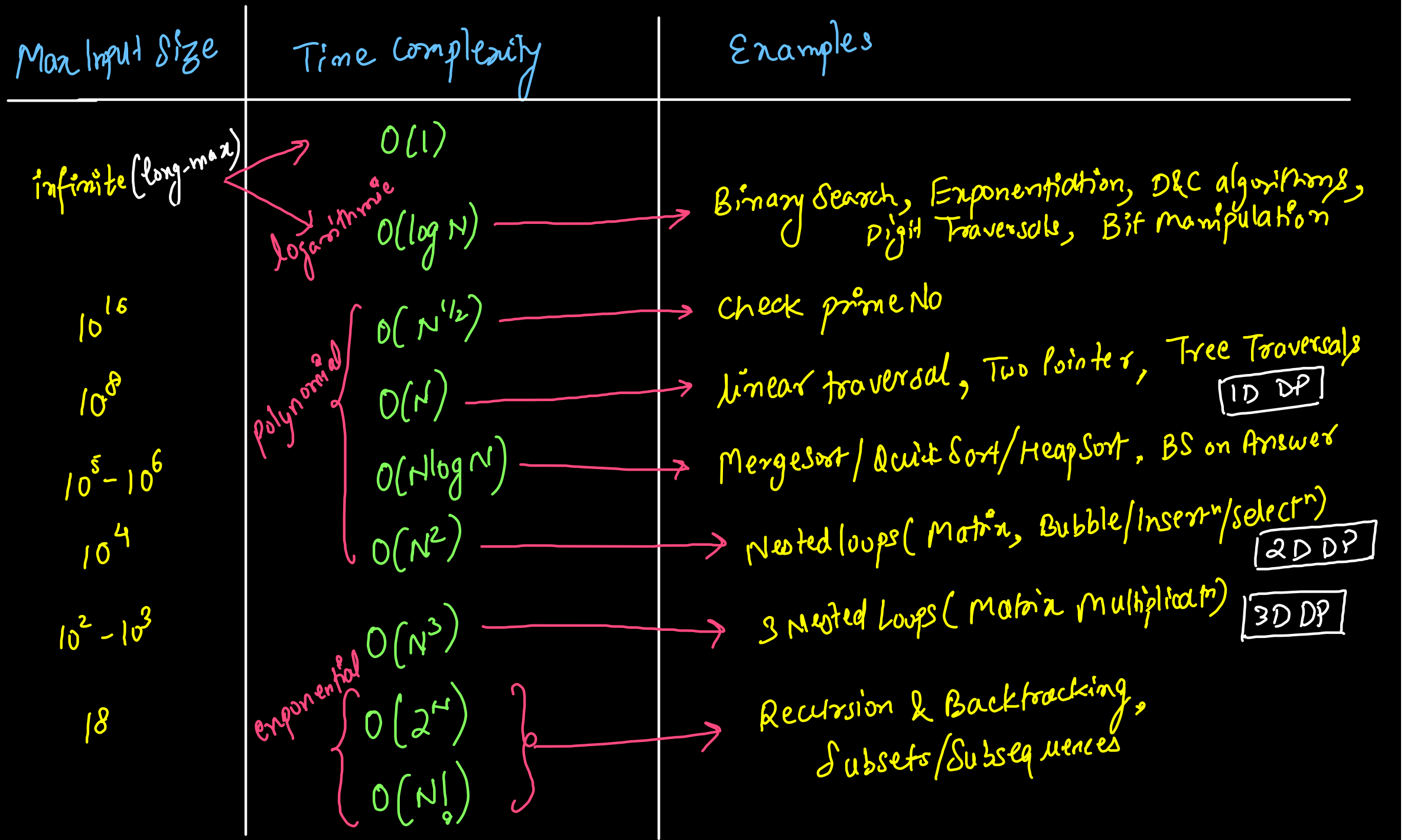
$N = 10^9$

$O(N) = 1$ sec — Accepted

*10 →

$O(N) = \frac{10^9 \text{ operations}}{10^8} = 10$ sec

TLE


Max Input Size
Time Complexity
Examples
infinite (long-max)
O(1)
Logarithmic
O(log N)
Binary Search, Exponentiation, D&C algorithms, Digit Traversals, Bit manipulation
10^{16}
Polynomial
$O(N^{1/2})$
Check prime No
10^8
O(N)
linear traversal, Two Pointer, Tree Traversals
1D DP
$10^5 - 10^6$
O(N log N)
Mergesort / Quick Sort / HeapSort, BS on Answer
10^4
$O(N^2)$
Nested loops (Matrix, Bubble/Insertn/selectn)
2D DP
$10^2 - 10^3$
$O(N^3)$
3 Nested Loops (Matrix Multiplication)
3D DP
18
Exponential
$O(2^N)$
$O(N!)$
Recursion & Backtracking, Subsets/Subsequences

Euclid's Algorithm

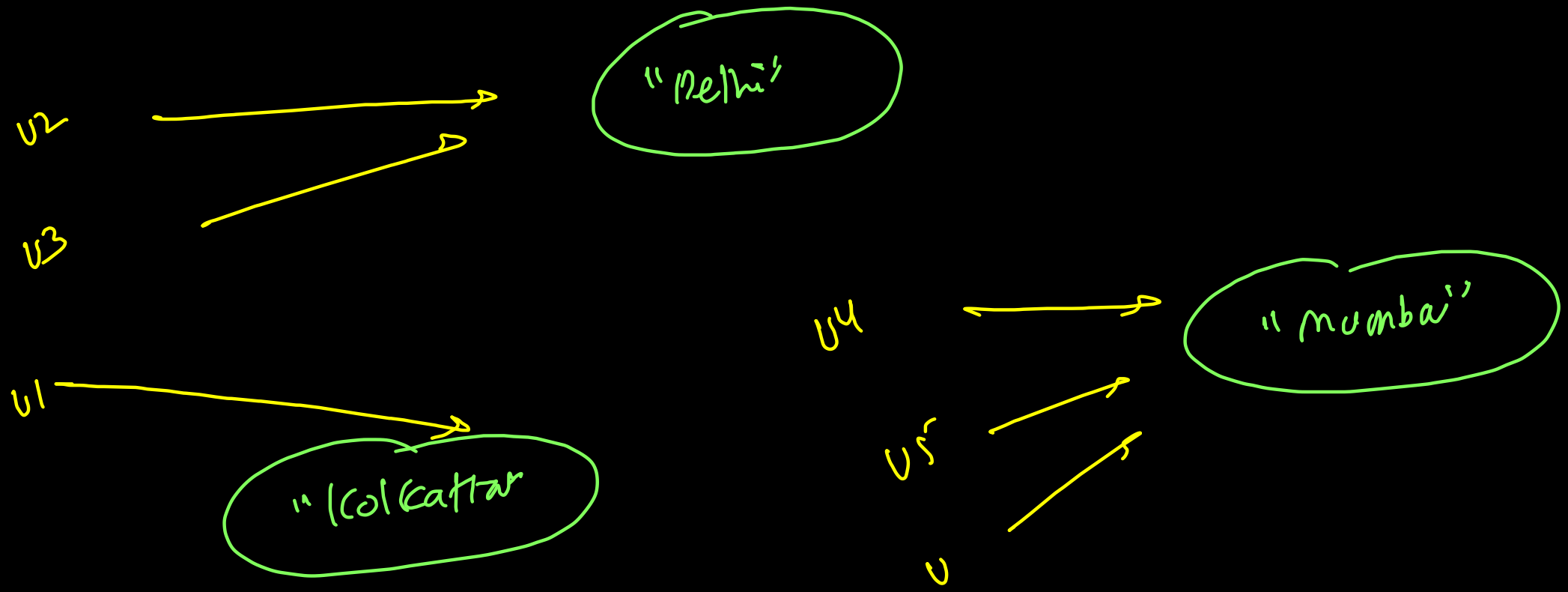
```java
static long divisionMethod(long a, long b) {
    while(a % b != 0) {
        long rem = a%b;
        a = b;
        b = rem;
    }

    return b;
}

static long gcd(long a,long b) {

    long originalA = a;

    while(a >= 1) {
        if(originalA % a == 0 && b % a == 0) return a;
        a--;
    }

    return 1L;
}
```

$\log_2 \min(a, b)$

GCD or HCF

32	48
16	24
8	12
4	6
2	3
1	1

2^5


U2
U3
"Delhi"
U1
"Kolkattar
U4
"mumbai"
U5
U

Arrays & Strings

Majority Element

eg!

40	10	40	10	10	40	20	40	20	40	40	40
0	1	2	3	4	5	6	7	8	9	10	11

n = 12

$\lfloor \frac{12}{2} \rfloor = 6$

Approach (1) Nested loop : For every element, count the frequency
$O(n^2)$ time, $O(1)$ space

Approach (2) Use Hashmap $O(n)$ time, $O(n)$ space

Approach (3) Use Sorting + Two Pointer $O(n \log n)$ time, $O(1)$ space

Approach (4) Boyer Moore Voting Algorithm

eg1:

40	10	40	10	10	40	20	40	20	40	40	40
0	1	2	3	4	5	6	7	8	9	10	11

majority = ~~40~~ ~~10~~ ~~40~~ ~~10~~ ~~20~~ (40) answer

frequency = ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ ~~1~~ (2)

eg2:

30	30	30	40	40	10	20	40	40	40	40
0	1	2	3	4	5	6	7	8	9	10

majority = ~~30~~ ~~20~~ (40) answer

lead = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~2~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ 1 2 3

$\lfloor \frac{n}{2} \rfloor : \frac{11}{2}$

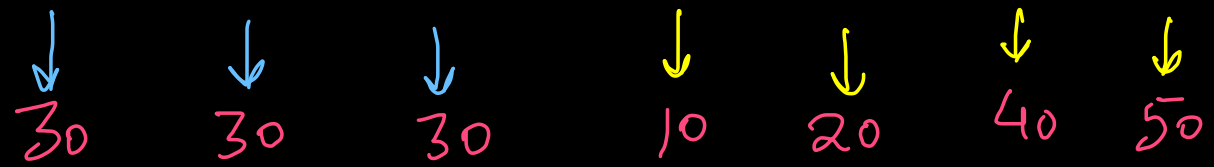
$= 5$

```java
public int majorityElement(int[] nums) {
    int majority = nums[0], lead = 0;

    // Time Complexity: O(N), Space Complexity: O(1) No Extra Space
    for(int val: nums){
        if(val == majority){
            lead++;
        } else if(lead == 0){
            majority = val;
            lead = 1;
        }
        else {
            lead--;
        }
    }

    return majority;
}
```

curr ही majority

lead = 0 ⇒ curr is new lead

majority सामने ↓



$\lfloor 7/2 \rfloor = 3$

maj = ~~30~~ 50 → neither majority nor highest frequency because majority element does not exist!

lead = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~2~~ ~~1~~ ~~0~~ 1

```java
public int majorityElement(int[] nums) {
    int majority = nums[0], lead = 0;

    // Time Complexity: O(N), Space Complexity: O(1) No Extra Space
    for(int val: nums){
        if(val == majority){
            lead++;
        } else if(lead == 0){
            majority = val;
            lead = 1;
        }
        else {
            lead--;
        }
    }

    return majority;
}
```

Majority Element - II

freq > $\lfloor n/3 \rfloor$

① 0 majority

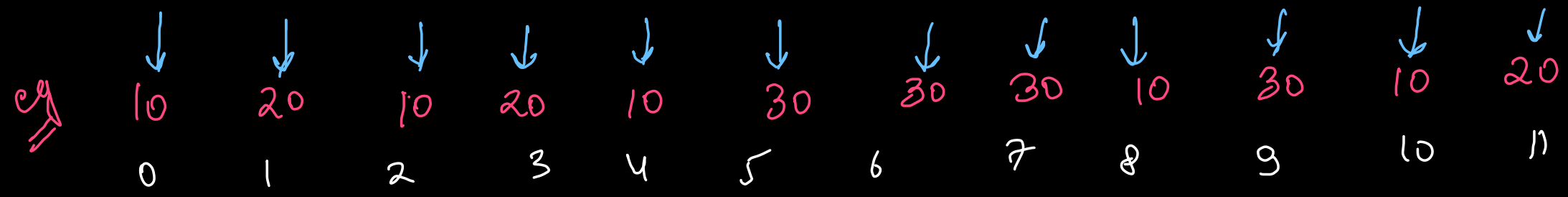
[10 20 30]

② only 1

[10 10 20]

③ atmost 2 majority

$\lfloor n/3 \rfloor = \lfloor 4/3 \rfloor = 1$



maj1 $-\infty$ 10

lead1 0 1 2 3 2 1 2 3 2

Yes ✓

maj2 $-\infty$ 20 30

lead2 0 1 2 1 0 1 2 1

2

```java
public int freq(int[] nums, int target){
    int count = 0;
    for(int val: nums){
        if(val == target) count++;
    }
    return count;
}
```

O(n)

```java
public List<Integer> majorityElement(int[] nums) {
    int majority1 = Integer.MIN_VALUE;
    int lead1 = 0;

    int majority2 = Integer.MIN_VALUE;
    int lead2 = 0;

    for(int val: nums){
        if(val == majority1){
            lead1++;
        } else if(val == majority2){
            lead2++;
        } else if(lead1 == 0){
            majority1 = val;
            lead1 = 1;
        } else if(lead2 == 0){
            majority2 = val;
            lead2 = 1;
        } else {
            lead1--;
            lead2--;
        }
    }

    List<Integer> res = new ArrayList<>();
    if(freq(nums, majority1) > nums.length / 3) res.add(majority1);
    if(freq(nums, majority2) > nums.length / 3) res.add(majority2);
    return res;
}
```

O(1)

O(n) linear

O(1)

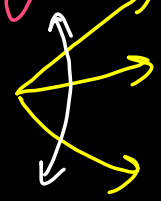
O(n)

1. Searching & Sorting

From Saturday

2. Recursion & Backtracking

From Scratch



Basic → 1 class

Subsets → 1 class

Maze path onwards