

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 Programmers
- Wipro ↑↑

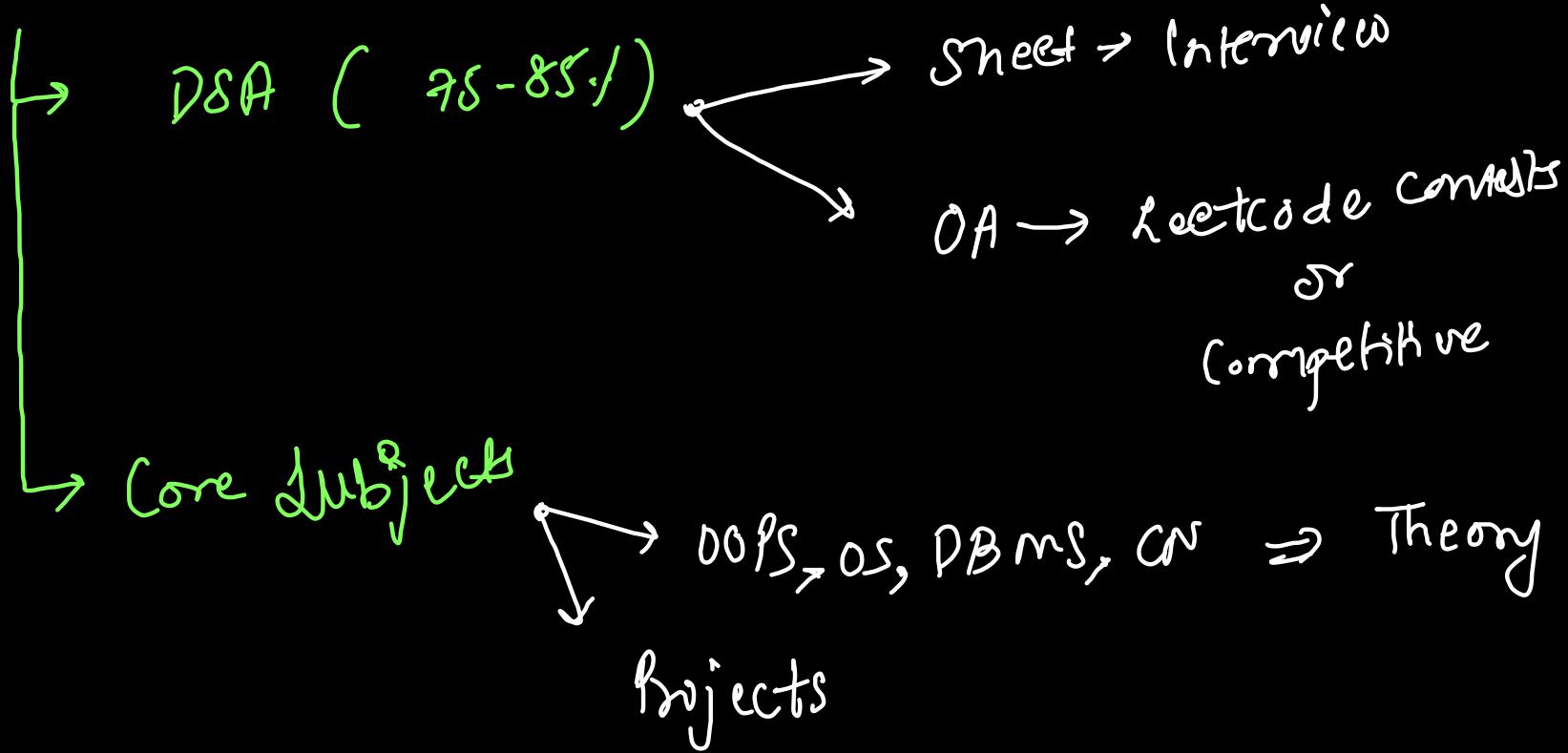
(i) Arrays & Strings

(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, -----)



(3.1) Some Product Based Companies

↳ Low level Design

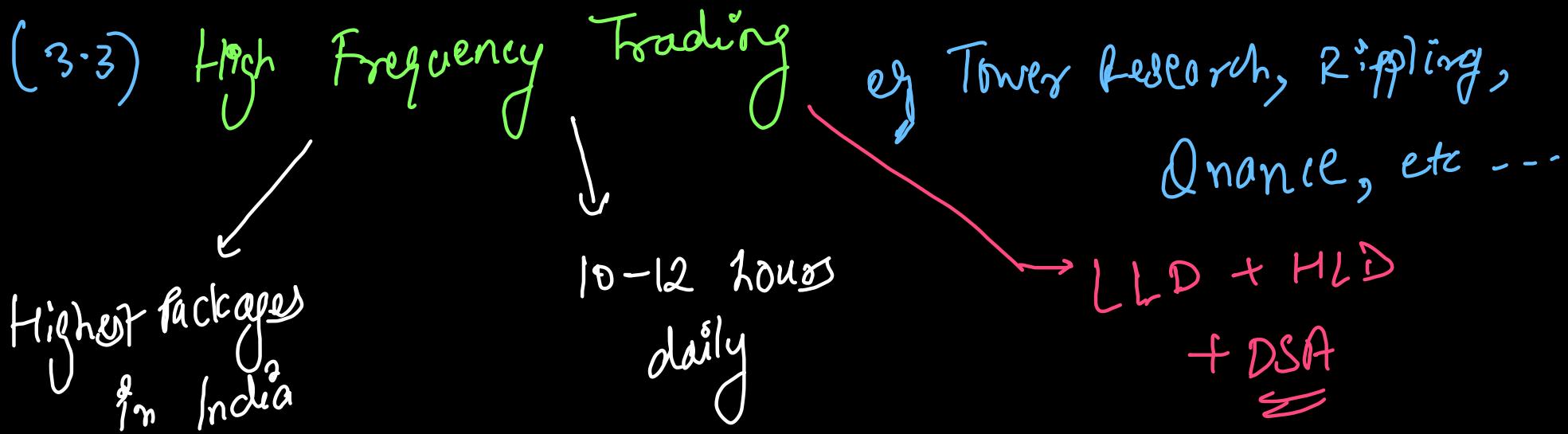
e.g. Sopnablog, Uber, Walmart, Flipkart, Phonepe, Ola,
Amazon (Bar raiser), HFT Companies,
Google (Googlyness), Salesforce

(3.2) Fin-tech Based Companies

↳ Goldman Sachs, Morgan Stanley, JP摩根,

↳ Agitude (Maths 11th-12thm)

geometry
Algebra
Calculus



Time & Space Complexity Analysis

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

Tradeoff → Time ↓ space ↑ ✓ if it is preferred
Time ↑ space ↓

(1) Runtime Analysis

Machine 1
(Windows 10)

Machine 2
(MacBook M2)

same code

Runtime ↑

↳ machine dependent ↳

same code

Runtime ↓

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

e.g.

Machine 1 (win 13)

Machine 2 (mac m2)

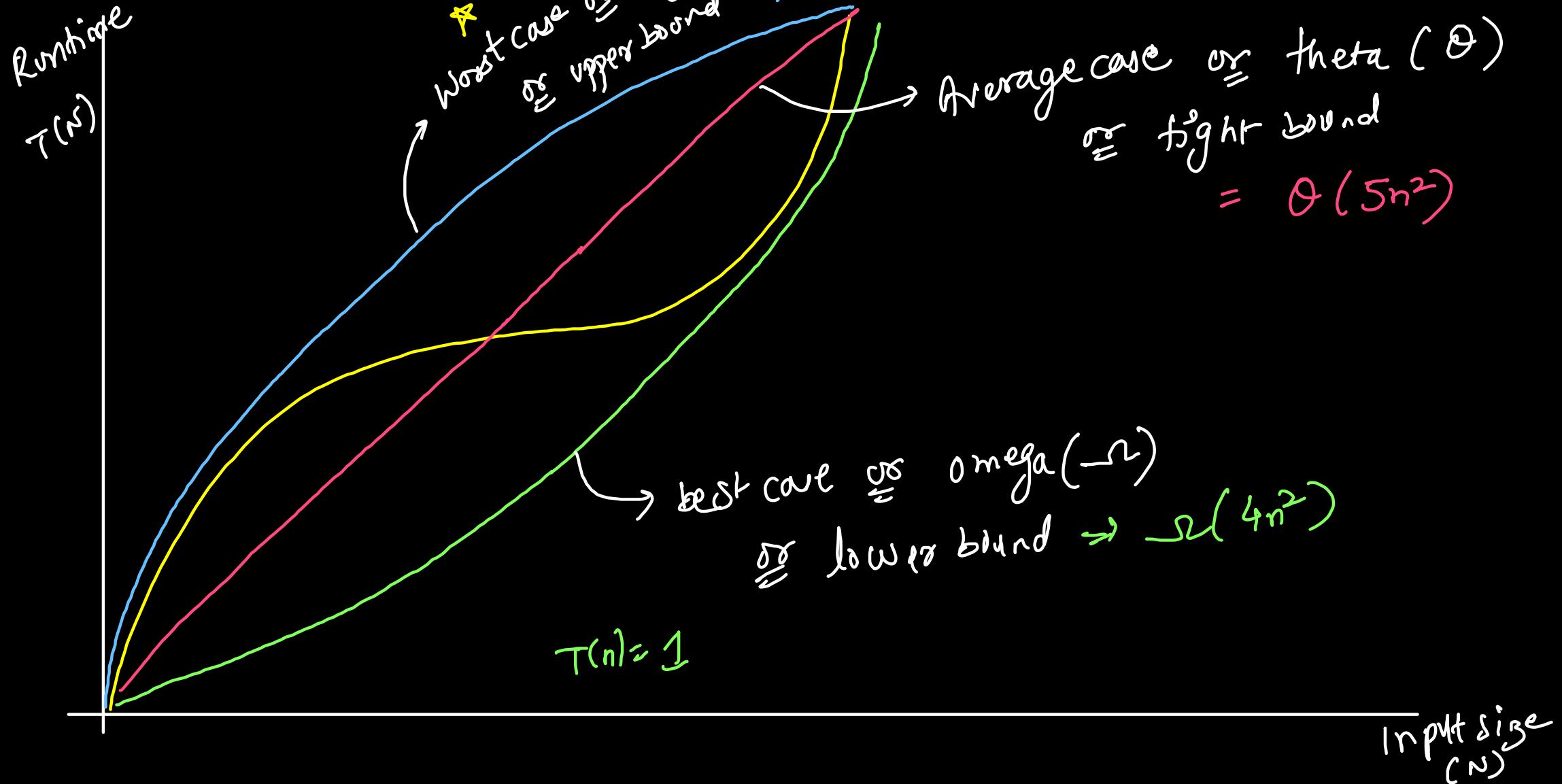
$$\begin{array}{ccc} \text{Input Size } (N) & & \\ N=10 & \xrightarrow{*10} & N=100 \\ & & \xrightarrow{*10} N=1000 \end{array}$$

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

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

Algorithm Runtime \propto Input size

Growth of Runtime



worst case

Actual runtime

$\underline{O(n^2)}$

$$6n^2$$

>

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

$n=1$
 $n=2$
 $n=3$
 $n=4$
 $n=5$

$$6 \cdot 1^2 = 6$$

↗

$$5 \cdot 1^2 + 3 \cdot 1 + 6 = 14$$

$$6 \cdot 2^2 = 24$$

↗

$$5 \cdot 2^2 + 3 \cdot 2 + 6 = 32$$

$$6 \cdot 3^2 = 54$$

↗

$$5 \cdot 3^2 + 3 \cdot 3 + 6 = 60$$

↗

↗

→ we ignore smaller
input size trend!

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

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

Computing

$$O(n) \underset{\sim}{\approx} O(3^n) < O(n^2)$$

10^0	100 ms	300 ms	10000 ms
$\sqrt{x}n$	$\downarrow \times 10$	$\downarrow \times 10$	$\downarrow \times 100$
10^{10}	1000 ms	3000 ms	100000 ms



We ignore constant values

$$O(3^n) \approx O(n)$$

Runtime

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

$$100n + 200n + 300n = \cancel{600n}$$

$$200 + 400 + 1000 = \cancel{1600n^0}$$

$$n^5 + n^4 * n^3 + n^2 * n^6 = \cancel{n^5} + \cancel{n^7} + n^8$$

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

Worst Case

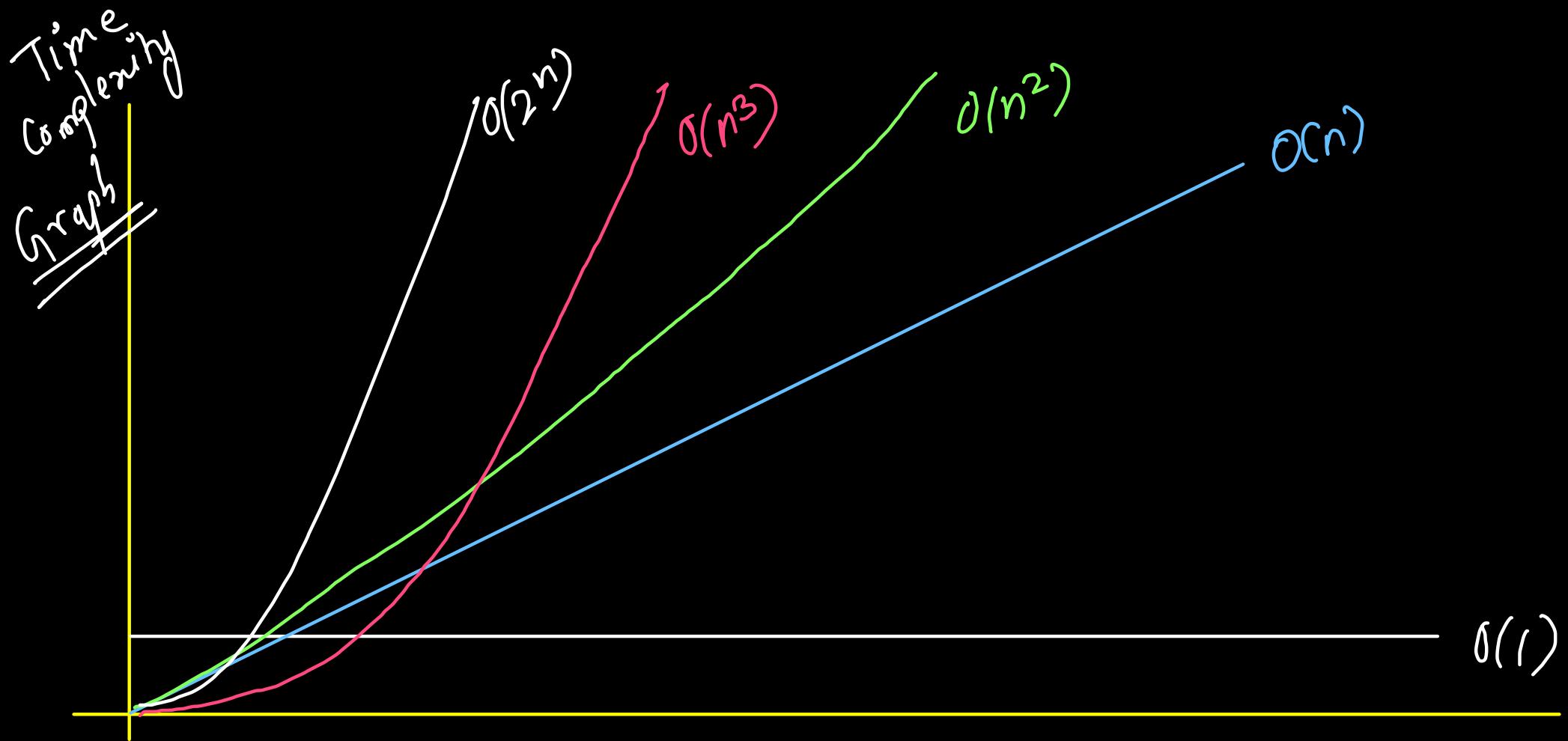
$O(n^3)$ cubic

$O(n)$ linear

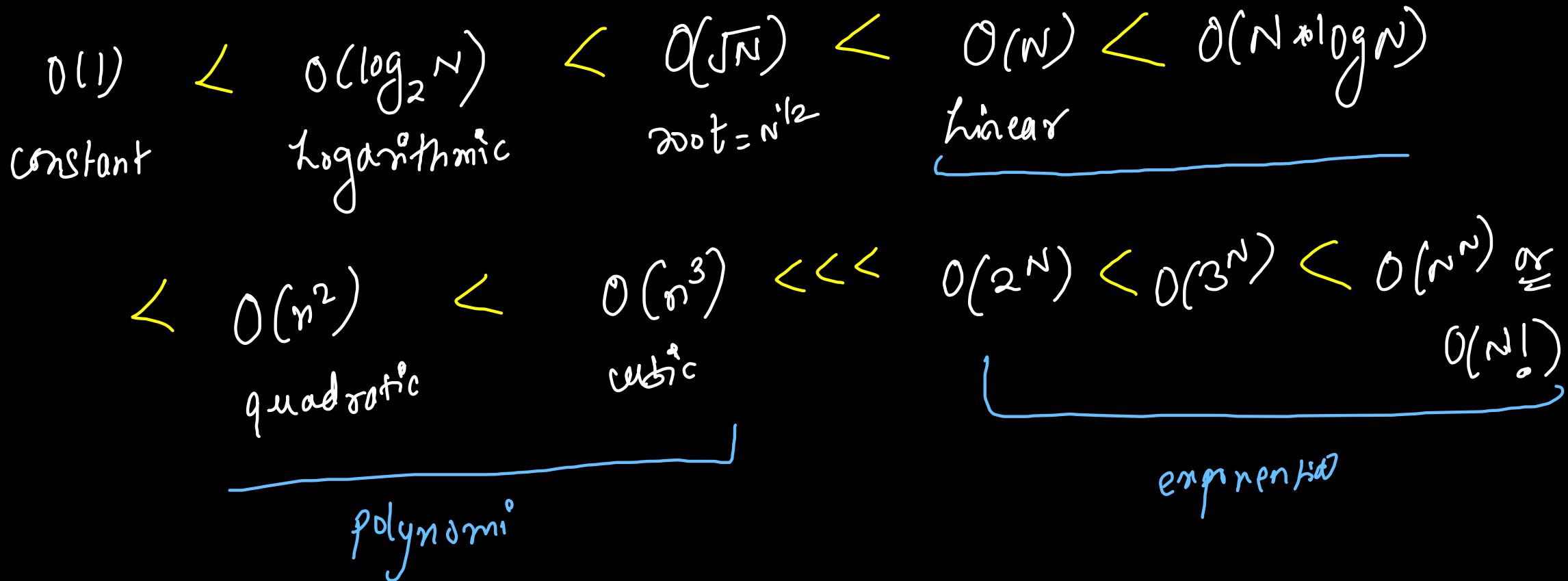
$O(n^0) = O(1)$ constant

$O(n^8)$

$O(2^n)$ exponential



Time Complexity Trend



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 (=) ↳ Ternary operator (()? :)
- (7) Function call & Return Statement ↳ Array/String indexing

Logarithmic Time Complexity

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

$$N=1 \quad 1 \quad 0 \quad 0 \quad 0$$

$$N=10 \quad 1 \quad 3 \quad 2 \quad 1$$

$$N=100 \\ = 10^2 \quad 1 \quad 6 \quad 4 \quad 2$$

$$N=10^3 \quad 1 \quad 12 \quad 8 \quad 4$$

$$N=10^{10} \\ (\text{int. max}) \quad 1 \quad 48 \quad 32 \quad 16$$

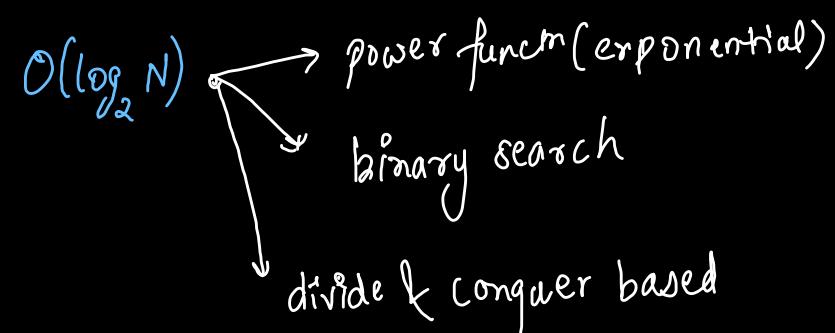
$$N=10^{18} \\ (\text{long max}) \quad 1 \quad 60 \quad 26 \quad 18$$

$$\rightarrow \log_a a = 1$$

$$\rightarrow \log_x 1 = 0$$

$$\rightarrow \log_x a^b \\ = b \log_x a$$

$$\rightarrow \log_n (a \times b) \\ = \log_n a + \log_n b$$



$O(\log_{10} x)$

Digit Traversal

```
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;  
}
```

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

$$10^{18} = (\text{9 digits})^1$$

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

$$10^2 = 100 \underbrace{\text{2 digits}}_{\text{1}}$$

$$10^5 = \underbrace{100000}_{\text{5 digits}}^1$$

For Loops Time Complexity

①

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

$n=5$

$idx = 1$	$k\ ms$	
$idx = 2$	$k\ ms$	$n \times O(1)$
$idx = 3$	$k\ ms$	$= O(n)$
$idx = 4$	$k\ ms$	
$idx = 5$	$k\ ms$	

$$\underbrace{5k\ ms}_{\Rightarrow O(n)} = n \times k\ ms$$

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 + " ");
}
```

$O(n)$

independent

$O(m)$

linear
 $O(n) + O(m)$

Time Complexity adds up



They are not nested

e.g. merge & sorted arrays

(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);
    }
}
```

$$n = 5, m = 3$$

$$i = 1$$

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

$$i = 2$$

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

$$i = 3$$

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

$$i = 4$$

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

$$i = 5$$

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

$$= m + m + m + \dots + m$$

$$= O(m \times n)$$

quadratic

⑤

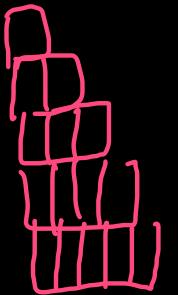
```
int n = scn.nextInt();

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

$n = 5$

$i = 1$

$j = 1 \Rightarrow 1$



$i = 2$

$j = 1, 2 \Rightarrow 2$

$i = 3$

$j = 1, 2, 3 \Rightarrow 3$

$i = 4$

$j = 1, 2, 3, 4 \Rightarrow 4$

$i = 5$

$j = 1, 2, 3, 4, 5 \Rightarrow 5$

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

$$k + 2k + 3k + \dots nk$$

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

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

6

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

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

$n=5$

$i=1$

$j=1, 2, 3, 4, 5$

$i=2$

$j=2, 3, 4, 5$

$i=3$

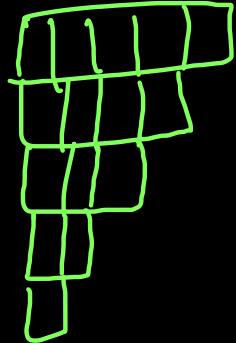
$j=3, 4, 5$

$i=4$

$j=4, 5$

$i=5$

$j=5$



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

$$= \overbrace{n(n+1)}^2 n^k$$

$= O(n^2)$

$n = 1^0$

⑦

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



$i = 1, 2, \dots, 9 \Rightarrow 9k \Rightarrow (n-1)k$
~~- nk - k~~
 $\therefore O(n)$

⑧

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



$i = 1, 2, 3, \dots, \lfloor \frac{n}{2} \rfloor \Rightarrow \frac{n}{2}k = \frac{n}{2}k$
 $\therefore O(n)$

⑨

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

⑩

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

n is +ve integer {> 1}

$n = 5$

$idx = 5$

$5 \leq 0$ false

$O(1)$ Constant

$n = 5$

$idx = 5$

$5 \geq 0$

Runtime Error
↳ Crash

$idx = 6$

$6 \geq 0$

$idx = 7$

$7 \geq 0$

8

$8 \geq 0$

⋮

$\infty \geq 0$

↳
infinite loop

$O(200fN) \Rightarrow$ check No is prime

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

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

11

06

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

$n = 16$

$idx = 1$

$|x| \leq 16$



16

$idx = 2$

$2 \times 2 \leq 16$



2

$idx = 3$

$3 \times 3 \leq 16$



4 times

$idx = 4$

$4 \times 4 \leq 16$



$idx \in \{$

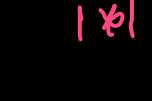
$5, 6\} \geq 16$



2

$n = 49$

$|x| \leq 49$



1

$idx = 2$

$2 \times 2 \leq 49$



2

$idx = 3$

$3 \times 3 \leq 49$



3

$idx = 4$

$4 \times 4 \leq 49$



4

$idx \in \{$

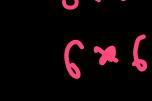
$5, 6\} \leq 49$



5

$idx = 6$

$6 \times 6 \leq 49$



6

$idx = 7$

$7 \times 7 \leq 49$



7

$$O(\sqrt{n}) = 10^{16}$$

$$O(n) = 10^8$$

$$O(r^n) = 10^4$$

int
 -2^{21} to 2^{21-1}
 10^9

10^{16}
 2^{63} to 2^{63-1}
 10^{19}

$O(\log_2 N)$: 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	$1 \leq 16$	✓	16 5 times
idx = 2	$2 \leq 16$	✓	
idx = 4	$4 \leq 16$	✓	
idx = 8	$8 \leq 16$	✓	
idx = 16	$16 \leq 16$	✓	

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

idx = 1	$1 \leq 49$	49 6 times
idx = 2	$2 \leq 49$	
idx = 4	$4 \leq 49$	
idx = 8	$8 \leq 49$	
idx = 16	$16 \leq 49$	
idx = 32	$32 \leq 49$	

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

idx = 1	$1 \leq 100$	100 7 times
idx = 2	$2 \leq 100$	
idx = 4	$4 \leq 100$	
idx = 8	$8 \leq 100$	
idx = 16	$16 \leq 100$	
idx = 32	$32 \leq 100$	
idx = 64	$64 \leq 100$	

$$\begin{aligned}
n &= 1024 \\
&= 2^{10} \\
&\downarrow \\
&10 \text{-time} \\
&10 = \log_2 1024
\end{aligned}$$

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

$$2^0 = 1$$

$$2^1 = 2$$

$$2^0 = 1$$

$$2^2 = 4$$

$$\log_2 2 \downarrow$$

$$2^3 = 8$$

$$N = 10^3$$

$$\log_2 N = 30$$

$$2^4 = 16$$

$$2^5 = 32$$

$$\log_2 1 = 0$$

$$\log_2 2 = 1$$

$$\log_2 4 = \log_2 (2^2) = 2 \times \underline{\log_2 2} = 2$$

$$\log_2 8 = \log_2 (2^3) = 3 \times \underline{\log_2 2} = 3$$

$$\log_2 16 = 4$$

$$\log_2 32 = 5$$

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
// 100
times

$n = 1000 \approx 10^3 = 2^{10}$

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);  $O(\sqrt{n})$ 
}
```

$i \in [0, \sqrt{n}, 2\sqrt{n}, \dots, \text{infinite}]$

✓ 15)

```
for(int idx = 0; idx <= n; idx = idx * 2){
    System.out.println(idx);  $\text{runtime error}$ 
}
```

✓ 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);
}
```

$n/2 = O(n)$

✓ 19)

```
for(int idx = 1; idx <= n; idx = idx + 3){
    System.out.println(idx);
}

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

✓ 20)

✓ 21)

$n = O(n)$

$n = \log_2 N$

$= O(n \log_2 N)$

$n = 10$
 $idx = \{3, 7, 3, 9\}$

$n = 20$
 $idx = \{3, 7, 3, 9, 11, 13, 15, 17, 19\}$

$n/2 \text{ times}$
 $\Rightarrow O(n)$



Weekend

Sat & Sun

9 AM - 12 PM

Weekdays

Monday, Wed, Thursday

10 PM - 12 AM

Schedule
for
DFA classes 2.0

```

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 + " ");
    }
}

```

$\} \quad O(2^n)$

$n=5$
 $1, 2, 3, \dots, 2^5 = 32$

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

$1 << 0 \quad 000001$
 $1 << 1 \quad 000010$
 $1 << 2 \quad 000100$
 $1 << 3 \quad 001000$
 $1 << 4 \quad 010000$

2^0
 2^1
 2^2
 $2^3 = 8$
 $2^4 = 16$

1	7	13
2	8	14
3	9	15
4	10	15
5	11	16K
6	12	

$16K = 2^4 K$
 $\Rightarrow O(2^n)$

```
//  $(1 << n) = 2^n$ 
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 + " ");
}
```

$O(2^n)$

$O(n)$

idx	max	✓
1	≤ 16	✓
2	≤ 16	✓
4	≤ 16	✓
8	≤ 16	✓
16	≤ 16	✓
32	> 16	

$n = 10$
 $max = 2^{10}$

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

11 times

$n = 4$
 $max = 2^4 = 16$

Leetcode

1) Problem Statement

2) Input & Output Format

3) Examples

4) Constraints

Online Judges

1

1 sec ↴

10^8 cycles/sec

\downarrow
10⁸ operations/sec

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

TIME

Max Input Size

Time Complexity

Examples

infinite (long_max)

$O(1)$

logarithmic
 $O(\log N)$

Binary Search, Exponentiation, D&C algorithms,
Digit Traversal, 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 / Quicksort / 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

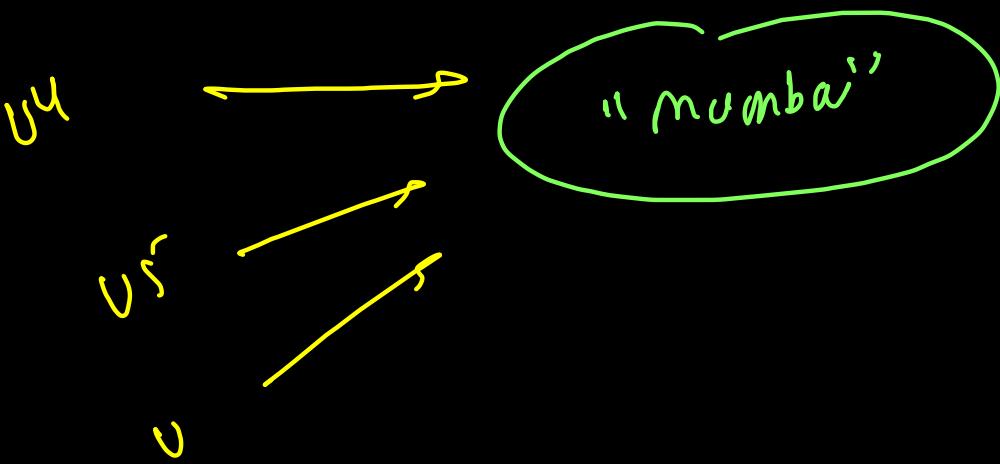
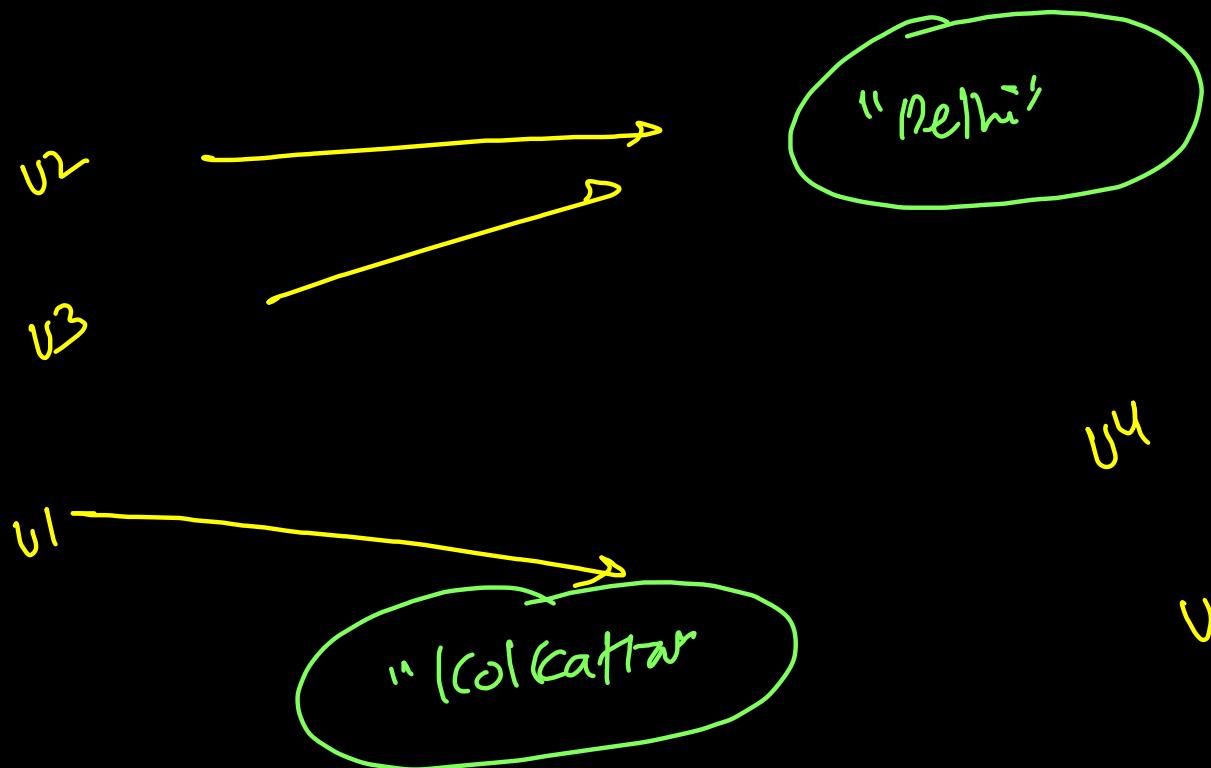
```
static long divisionMethod(long a, long b) {  
    while(a % b != 0) {  
        long rem = a%b;  
        a = b;  
        b = rem;  
    }  
  
    return b;  
}  
  
static long gcd(long a, long b) {  
  
    long originalA = a;  
  
    while(a >= 1) {  
        if(originalA % a == 0 && b % a == 0) return a;  
        a--;  
    }  
  
    return 1L;  
}
```

$\log \min(a, b)$

GCD
or
HCF

32 48
16 24
8 12
4 6
2 3
1

2^5



Arrays & Strings

Majority Element

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

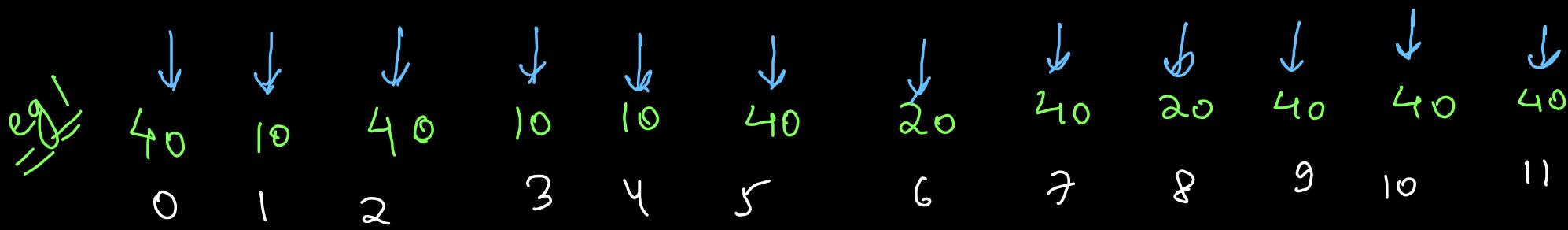
$n=12$
 $\sqrt{12} = 6$

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

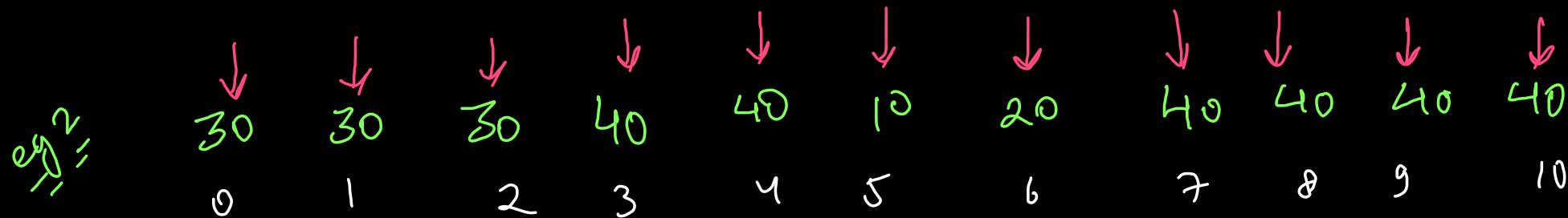
Approach ② Use Hashmap $O(n)$ time, $O(n)$ space

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

Approach ④ Boyer Moore Voting Algorithm



$$\text{majority} = \cancel{40} \cancel{10} \cancel{40} \cancel{10} \cancel{20} \quad (40) \text{ answer}$$



$$\text{majority} = \cancel{36} \cancel{20} (40) \text{ answer} \swarrow$$

head: ~~✓ ✓ b2 x0 z0 x23~~

111
111

```
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 \Rightarrow curr is new lead)

majority साझा ↓

\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow
30 30 30 10 20 40 50

$$\lfloor \frac{7}{2} \rfloor = 3 \uparrow$$

maj = 30 (50) \rightarrow neither majority nor highest frequency because majority element does not exist!

lead = 0 X 2 / 3 / X 0 1

```
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 - ||
freq > $\lfloor n/3 \rfloor$

① 0 majority

$$\downarrow \\ [10 \ 20 \ 30]$$

② only 1

$$\downarrow \\ [10 \ 10 \ 20]$$

③ atmost 2 majority

$$\downarrow \\ [10 \ 10 \ 10 \ 20 \ 20 \ 20 \ 30]$$

$$\begin{aligned} \lfloor n/3 \rfloor \\ = \lfloor 2/3 \rfloor = 1 \end{aligned}$$

	\downarrow											
ex	10	20	10	20	10	30	30	30	10	30	10	20
	0	1	2	3	4	5	6	7	8	9	10	11

maj1 ~~-~~¹⁰
 lead1 ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ ~~11~~ ~~12~~ ~~13~~ ~~14~~ ~~15~~ ~~16~~ ~~17~~ ~~18~~ ~~19~~ ~~20~~ ~~21~~ ~~22~~ ~~23~~ ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ ~~29~~ ~~30~~ ~~31~~ ~~32~~ ~~33~~ ~~34~~ ~~35~~ ~~36~~ ~~37~~ ~~38~~ ~~39~~ ~~40~~ ~~41~~ ~~42~~ ~~43~~ ~~44~~ ~~45~~ ~~46~~ ~~47~~ ~~48~~ ~~49~~ ~~50~~ ~~51~~ ~~52~~ ~~53~~ ~~54~~ ~~55~~ ~~56~~ ~~57~~ ~~58~~ ~~59~~ ~~60~~ ~~61~~ ~~62~~ ~~63~~ ~~64~~ ~~65~~ ~~66~~ ~~67~~ ~~68~~ ~~69~~ ~~70~~ ~~71~~ ~~72~~ ~~73~~ ~~74~~ ~~75~~ ~~76~~ ~~77~~ ~~78~~ ~~79~~ ~~80~~ ~~81~~ ~~82~~ ~~83~~ ~~84~~ ~~85~~ ~~86~~ ~~87~~ ~~88~~ ~~89~~ ~~90~~ ~~91~~ ~~92~~ ~~93~~ ~~94~~ ~~95~~ ~~96~~ ~~97~~ ~~98~~ ~~99~~ ~~100~~ ~~101~~ ~~102~~ ~~103~~ ~~104~~ ~~105~~ ~~106~~ ~~107~~ ~~108~~ ~~109~~ ~~110~~ ~~111~~ ~~112~~ ~~113~~ ~~114~~ ~~115~~ ~~116~~ ~~117~~ ~~118~~ ~~119~~ ~~120~~ ~~121~~ ~~122~~ ~~123~~ ~~124~~ ~~125~~ ~~126~~ ~~127~~ ~~128~~ ~~129~~ ~~130~~ ~~131~~ ~~132~~ ~~133~~ ~~134~~ ~~135~~ ~~136~~ ~~137~~ ~~138~~ ~~139~~ ~~140~~ ~~141~~ ~~142~~ ~~143~~ ~~144~~ ~~145~~ ~~146~~ ~~147~~ ~~148~~ ~~149~~ ~~150~~ ~~151~~ ~~152~~ ~~153~~ ~~154~~ ~~155~~ ~~156~~ ~~157~~ ~~158~~ ~~159~~ ~~160~~ ~~161~~ ~~162~~ ~~163~~ ~~164~~ ~~165~~ ~~166~~ ~~167~~ ~~168~~ ~~169~~ ~~170~~ ~~171~~ ~~172~~ ~~173~~ ~~174~~ ~~175~~ ~~176~~ ~~177~~ ~~178~~ ~~179~~ ~~180~~ ~~181~~ ~~182~~ ~~183~~ ~~184~~ ~~185~~ ~~186~~ ~~187~~ ~~188~~ ~~189~~ ~~190~~ ~~191~~ ~~192~~ ~~193~~ ~~194~~ ~~195~~ ~~196~~ ~~197~~ ~~198~~ ~~199~~ ~~200~~ ~~201~~ ~~202~~ ~~203~~ ~~204~~ ~~205~~ ~~206~~ ~~207~~ ~~208~~ ~~209~~ ~~210~~ ~~211~~ ~~212~~ ~~213~~ ~~214~~ ~~215~~ ~~216~~ ~~217~~ ~~218~~ ~~219~~ ~~220~~ ~~221~~ ~~222~~ ~~223~~ ~~224~~ ~~225~~ ~~226~~ ~~227~~ ~~228~~ ~~229~~ ~~230~~ ~~231~~ ~~232~~ ~~233~~ ~~234~~ ~~235~~ ~~236~~ ~~237~~ ~~238~~ ~~239~~ ~~240~~ ~~241~~ ~~242~~ ~~243~~ ~~244~~ ~~245~~ ~~246~~ ~~247~~ ~~248~~ ~~249~~ ~~250~~ ~~251~~ ~~252~~ ~~253~~ ~~254~~ ~~255~~ ~~256~~ ~~257~~ ~~258~~ ~~259~~ ~~260~~ ~~261~~ ~~262~~ ~~263~~ ~~264~~ ~~265~~ ~~266~~ ~~267~~ ~~268~~ ~~269~~ ~~270~~ ~~271~~ ~~272~~ ~~273~~ ~~274~~ ~~275~~ ~~276~~ ~~277~~ ~~278~~ ~~279~~ ~~280~~ ~~281~~ ~~282~~ ~~283~~ ~~284~~ ~~285~~ ~~286~~ ~~287~~ ~~288~~ ~~289~~ ~~290~~ ~~291~~ ~~292~~ ~~293~~ ~~294~~ ~~295~~ ~~296~~ ~~297~~ ~~298~~ ~~299~~ ~~300~~ ~~301~~ ~~302~~ ~~303~~ ~~304~~ ~~305~~ ~~306~~ ~~307~~ ~~308~~ ~~309~~ ~~310~~ ~~311~~ ~~312~~ ~~313~~ ~~314~~ ~~315~~ ~~316~~ ~~317~~ ~~318~~ ~~319~~ ~~320~~ ~~321~~ ~~322~~ ~~323~~ ~~324~~ ~~325~~ ~~326~~ ~~327~~ ~~328~~ ~~329~~ ~~330~~ ~~331~~ ~~332~~ ~~333~~ ~~334~~ ~~335~~ ~~336~~ ~~337~~ ~~338~~ ~~339~~ ~~340~~ ~~341~~ ~~342~~ ~~343~~ ~~344~~ ~~345~~ ~~346~~ ~~347~~ ~~348~~ ~~349~~ ~~350~~ ~~351~~ ~~352~~ ~~353~~ ~~354~~ ~~355~~ ~~356~~ ~~357~~ ~~358~~ ~~359~~ ~~360~~ ~~361~~ ~~362~~ ~~363~~ ~~364~~ ~~365~~ ~~366~~ ~~367~~ ~~368~~ ~~369~~ ~~370~~ ~~371~~ ~~372~~ ~~373~~ ~~374~~ ~~375~~ ~~376~~ ~~377~~ ~~378~~ ~~379~~ ~~380~~ ~~381~~ ~~382~~ ~~383~~ ~~384~~ ~~385~~ ~~386~~ ~~387~~ ~~388~~ ~~389~~ ~~390~~ ~~391~~ ~~392~~ ~~393~~ ~~394~~ ~~395~~ ~~396~~ ~~397~~ ~~398~~ ~~399~~ ~~400~~ ~~401~~ ~~402~~ ~~403~~ ~~404~~ ~~405~~ ~~406~~ ~~407~~ ~~408~~ ~~409~~ ~~410~~ ~~411~~ ~~412~~ ~~413~~ ~~414~~ ~~415~~ ~~416~~ ~~417~~ ~~418~~ ~~419~~ ~~420~~ ~~421~~ ~~422~~ ~~423~~ ~~424~~ ~~425~~ ~~426~~ ~~427~~ ~~428~~ ~~429~~ ~~430~~ ~~431~~ ~~432~~ ~~433~~ ~~434~~ ~~435~~ ~~436~~ ~~437~~ ~~438~~ ~~439~~ ~~440~~ ~~441~~ ~~442~~ ~~443~~ ~~444~~ ~~445~~ ~~446~~ ~~447~~ ~~448~~ ~~449~~ ~~450~~ ~~451~~ ~~452~~ ~~453~~ ~~454~~ ~~455~~ ~~456~~ ~~457~~ ~~458~~ ~~459~~ ~~460~~ ~~461~~ ~~462~~ ~~463~~ ~~464~~ ~~465~~ ~~466~~ ~~467~~ ~~468~~ ~~469~~ ~~470~~ ~~471~~ ~~472~~ ~~473~~ ~~474~~ ~~475~~ ~~476~~ ~~477~~ ~~478~~ ~~479~~ ~~480~~ ~~481~~ ~~482~~ ~~483~~ ~~484~~ ~~485~~ ~~486~~ ~~487~~ ~~488~~ ~~489~~ ~~490~~ ~~491~~ ~~492~~ ~~493~~ ~~494~~ ~~495~~ ~~496~~ ~~497~~ ~~498~~ ~~499~~ ~~500~~ ~~501~~ ~~502~~ ~~503~~ ~~504~~ ~~505~~ ~~506~~ ~~507~~ ~~508~~ ~~509~~ ~~510~~ ~~511~~ ~~512~~ ~~513~~ ~~514~~ ~~515~~ ~~516~~ ~~517~~ ~~518~~ ~~519~~ ~~520~~ ~~521~~ ~~522~~ ~~523~~ ~~524~~ ~~525~~ ~~526~~ ~~527~~ ~~528~~ ~~529~~ ~~530~~ ~~531~~ ~~532~~ ~~533~~ ~~534~~ ~~535~~ ~~536~~ ~~537~~ ~~538~~ ~~539~~ ~~540~~ ~~541~~ ~~542~~ ~~543~~ ~~544~~ ~~545~~ ~~546~~ ~~547~~ ~~548~~ ~~549~~ ~~550~~ ~~551~~ ~~552~~ ~~553~~ ~~554~~ ~~555~~ ~~556~~ ~~557~~ ~~558~~ ~~559~~ ~~560~~ ~~561~~ ~~562~~ ~~563~~ ~~564~~ ~~565~~ ~~566~~ ~~567~~ ~~568~~ ~~569~~ ~~570~~ ~~571~~ ~~572~~ ~~573~~ ~~574~~ ~~575~~ ~~576~~ ~~577~~ ~~578~~ ~~579~~ ~~580~~ ~~581~~ ~~582~~ ~~583~~ ~~584~~ ~~585~~ ~~586~~ ~~587~~ ~~588~~ ~~589~~ ~~590~~ ~~591~~ ~~592~~ ~~593~~ ~~594~~ ~~595~~ ~~596~~ ~~597~~ ~~598~~ ~~599~~ ~~600~~ ~~601~~ ~~602~~ ~~603~~ ~~604~~ ~~605~~ ~~606~~ ~~607~~ ~~608~~ ~~609~~ ~~610~~ ~~611~~ ~~612~~ ~~613~~ ~~614~~ ~~615~~ ~~616~~ ~~617~~ ~~618~~ ~~619~~ ~~620~~ ~~621~~ ~~622~~ ~~623~~ ~~624~~ ~~625~~ ~~626~~ ~~627~~ ~~628~~ ~~629~~ ~~630~~ ~~631~~ ~~632~~ ~~633~~ ~~634~~ ~~635~~ ~~636~~ ~~637~~ ~~638~~ ~~639~~ ~~640~~ ~~641~~ ~~642~~ ~~643~~ ~~644~~ ~~645~~ ~~646~~ ~~647~~ ~~648~~ ~~649~~ ~~650~~ ~~651~~ ~~652~~ ~~653~~ ~~654~~ ~~655~~ ~~656~~ ~~657~~ ~~658~~ ~~659~~ ~~660~~ ~~661~~ ~~662~~ ~~663~~ ~~664~~ ~~665~~ ~~666~~ ~~667~~ ~~668~~ ~~669~~ ~~670~~ ~~671~~ ~~672~~ ~~673~~ ~~674~~ ~~675~~ ~~676~~ ~~677~~ ~~678~~ ~~679~~ ~~680~~ ~~681~~ ~~682~~ ~~683~~ ~~684~~ ~~685~~ ~~686~~ ~~687~~ ~~688~~ ~~689~~ ~~690~~ ~~691~~ ~~692~~ ~~693~~ ~~694~~ ~~695~~ ~~696~~ ~~697~~ ~~698~~ ~~699~~ ~~700~~ ~~701~~ ~~702~~ ~~703~~ ~~704~~ ~~705~~ ~~706~~ ~~707~~ ~~708~~ ~~709~~ ~~710~~ ~~711~~ ~~712~~ ~~713~~ ~~714~~ ~~715~~ ~~716~~ ~~717~~ ~~718~~ ~~719~~ ~~720~~ ~~721~~ ~~722~~ ~~723~~ ~~724~~ ~~725~~ ~~726~~ ~~727~~ ~~728~~ ~~729~~ ~~730~~ ~~731~~ ~~732~~ ~~733~~ ~~734~~ ~~735~~ ~~736~~ ~~737~~ ~~738~~ ~~739~~ ~~740~~ ~~741~~ ~~742~~ ~~743~~ ~~744~~ ~~745~~ ~~746~~ ~~747~~ ~~748~~ ~~749~~ ~~750~~ ~~751~~ ~~752~~ ~~753~~ ~~754~~ ~~755~~ ~~756~~ ~~757~~ ~~758~~ ~~759~~ ~~760~~ ~~761~~ ~~762~~ ~~763~~ ~~764~~ ~~765~~ ~~766~~ ~~767~~ ~~768~~ ~~769~~ ~~770~~ ~~771~~ ~~772~~ ~~773~~ ~~774~~ ~~775~~ ~~776~~ ~~777~~ ~~778~~ ~~779~~ ~~780~~ ~~781~~ ~~782~~ ~~783~~ ~~784~~ ~~785~~ ~~786~~ ~~787~~ ~~788~~ ~~789~~ ~~790~~ ~~791~~ ~~792~~ ~~793~~ ~~794~~ ~~795~~ ~~796~~ ~~797~~ ~~798~~ ~~799~~ ~~800~~ ~~801~~ ~~802~~ ~~803~~ ~~804~~ ~~805~~ ~~806~~ ~~807~~ ~~808~~ ~~809~~ ~~810~~ ~~811~~ ~~812~~ ~~813~~ ~~814~~ ~~815~~ ~~816~~ ~~817~~ ~~818~~ ~~819~~ ~~820~~ ~~821~~ ~~822~~ ~~823~~ ~~824~~ ~~825~~ ~~826~~ ~~827~~ ~~828~~ ~~829~~ ~~830~~ ~~831~~ ~~832~~ ~~833~~ ~~834~~ ~~835~~ ~~836~~ ~~837~~ ~~838~~ ~~839~~ ~~840~~ ~~841~~ ~~842~~ ~~843~~ ~~844~~ ~~845~~ ~~846~~ ~~847~~ ~~848~~ ~~849~~ ~~850~~ ~~851~~ ~~852~~ ~~853~~ ~~854~~ ~~855~~ ~~856~~ ~~857~~ ~~858~~ ~~859~~ ~~860~~ ~~861~~ ~~862~~ ~~863~~ ~~864~~ ~~865~~ ~~866~~ ~~867~~ ~~868~~ ~~869~~ ~~870~~ ~~871~~ ~~872~~ ~~873~~ ~~874~~ ~~875~~ ~~876~~ ~~877~~ ~~878~~ ~~879~~ ~~880~~ ~~881~~ ~~882~~ ~~883~~ ~~884~~ ~~885~~ ~~886~~ ~~887~~ ~~888~~ ~~889~~ ~~890~~ ~~891~~ ~~892~~ ~~893~~ ~~894~~ ~~895~~ ~~896~~ ~~897~~ ~~898~~ ~~899~~ ~~900~~ ~~901~~ ~~902~~ ~~903~~ ~~904~~ ~~905~~ ~~906~~ ~~907~~ ~~908~~ ~~909~~ ~~910~~ ~~911~~ ~~912~~ ~~913~~ ~~914~~ ~~915~~ ~~916~~ ~~917~~ ~~918~~ ~~919~~ ~~920~~ ~~921~~ ~~922~~ ~~923~~ ~~924~~ ~~925~~ ~~926~~ ~~927~~ ~~928~~ ~~929~~ ~~930~~ ~~931~~ ~~932~~ ~~933~~ ~~934~~ ~~935~~ ~~936~~ ~~937~~ ~~938~~ ~~939~~ ~~940~~ ~~941~~ ~~942~~ ~~943~~ ~~944~~ ~~945~~ ~~946~~ ~~947~~ ~~948~~ ~~949~~ ~~950~~ ~~951~~ ~~952~~ ~~953~~ ~~954~~ ~~955~~ ~~956~~ ~~957~~ ~~958~~ ~~959~~ ~~960~~ ~~961~~ ~~962~~ ~~963~~ ~~964~~ ~~965~~ ~~966~~ ~~967~~ ~~968~~ ~~969~~ ~~970~~ ~~971~~ ~~972~~ ~~973~~ ~~974~~ ~~975~~ ~~976~~ ~~977~~ ~~978~~ ~~979~~ ~~980~~ ~~981~~ ~~982~~ ~~983~~ ~~984~~ ~~985~~ ~~986~~ ~~987~~ ~~988~~ ~~989~~ ~~990~~ ~~991~~ ~~992~~ ~~993~~ ~~994~~ ~~995~~ ~~996~~ ~~997~~ ~~998~~ ~~999~~ ~~1000~~

maj2 ~~-~~²⁰
 lead2 ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ ~~11~~ ~~12~~ ~~13~~ ~~14~~ ~~15~~ ~~16~~ ~~17~~ ~~18~~ ~~19~~ ~~20~~ ~~21~~ ~~22~~ ~~23~~ ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ ~~29~~ ~~30~~ ~~31~~ ~~32~~ ~~33~~ ~~34~~ ~~35~~ ~~36~~ ~~37~~ ~~38~~ ~~39~~ ~~40~~ ~~41~~ ~~42~~ ~~43~~ ~~44~~ ~~45~~ ~~46~~ ~~47~~ ~~48~~ ~~49~~ ~~50~~ ~~51~~ ~~52~~ ~~53~~ ~~54~~ ~~55~~ ~~56~~ ~~57~~ ~~58~~ ~~59~~ ~~60~~ ~~61~~ ~~62~~ ~~63~~ ~~64~~ ~~65~~ ~~66~~ ~~67~~ ~~68~~ ~~69~~ ~~70~~ ~~71~~ ~~72~~ ~~73~~ ~~74~~ ~~75~~ ~~76~~ ~~77~~ ~~78~~ ~~79~~ ~~80~~ ~~81~~ ~~82~~ ~~83~~ ~~84~~ ~~85~~ ~~86~~ ~~87~~ ~~88~~ ~~89~~ ~~90~~ ~~91~~ ~~92~~ ~~93~~ ~~94~~ ~~95~~ ~~96~~ ~~97~~ ~~98~~ ~~99~~ ~~100~~

152

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

$\mathcal{O}(n)$

```
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;  
    }
```

$\mathcal{O}(n)$

$\rightarrow \mathcal{O}(n)$ linear

$\mathcal{O}(1)$

① Searching & Sorting

From Saturday

② Recursion & Backtracking

From Scratch

Basic subsets → 1 class
Maze Path onwards