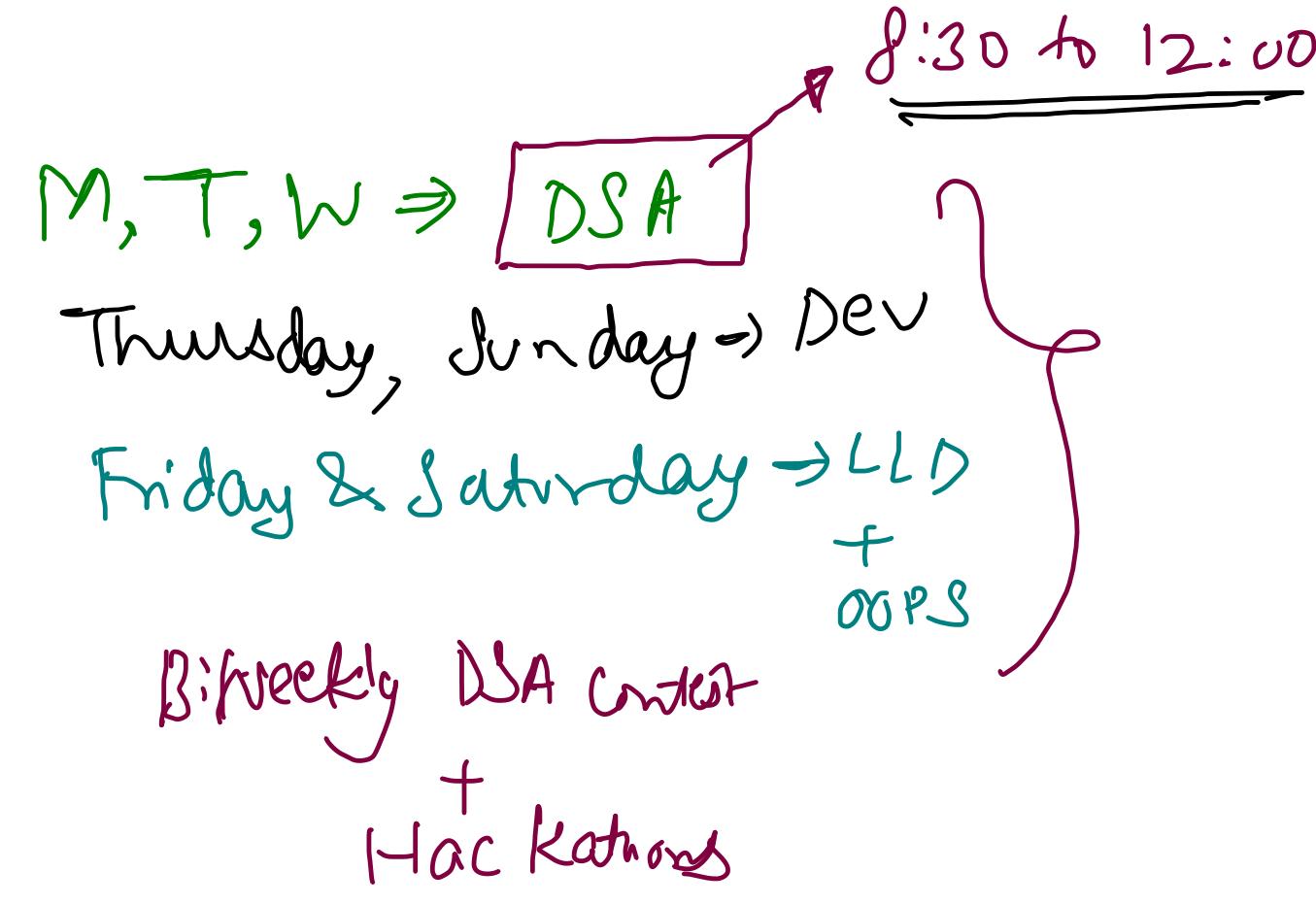


Tentative Schedule (FJP-2)

{ DSA - level ① } Approximately 35 lectures remaining

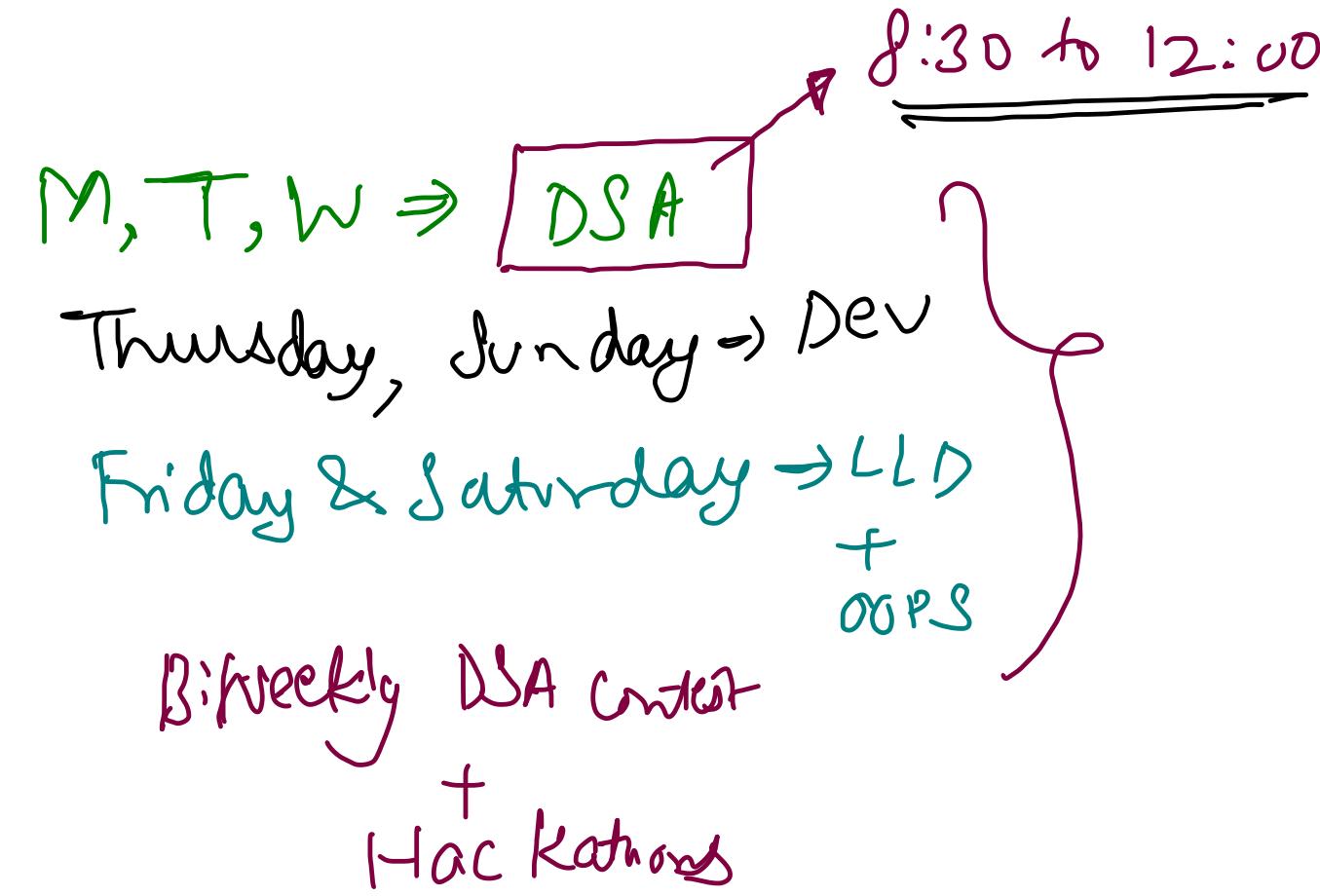
- ① Conceptual Building Blocks → Complete
- ② Recursion & Backtracking → Complete
- ③ Searching & Sorting
Time & Space Complexity] → 3 lectures
- ④ Dynamic Programming → 7 lectures
- ⑤ Linked List → 4 lectures
- ⑥ Stack & Queue, OOPS → 4 lectures
- ⑦ Generic Tree → 3 lectures
- ⑧ Binary Tree & BST → 4 lectures
- ⑨ Hashmap & Heap → 4 lectures
- ⑩ Graphs → 5 lectures



Tentative Schedule (FJP-2)

{ DSA - level ① } Approximately 35 lectures remaining

- ① Conceptual Building Blocks → Complete
- ② Recursion & Backtracking → Complete
- ③ Searching & Sorting
Time & Space Complexity] → 3 lectures
- ④ Dynamic Programming → 7 lectures
- ⑤ Linked List → 4 lectures
- ⑥ Stack & Queue, OOPS → 4 lectures
- ⑦ Generic Tree → 3 lectures
- ⑧ Binary Tree & BST → 4 lectures
- ⑨ Hashmap & Heap → 4 lectures
- ⑩ Graphs → 5 lectures



~~Time & Space Complexity~~

TIME COMPLEXITY

→ theta (Θ or tight bound)

→ Best Case, Average Case, Worst Case

↳ omega (Ω or lower bound)

↳ Big O (O or upper bound)

→ Asymptotic Analysis

→ Graph (Time vs Input)

→ Run-time vs Complete Time

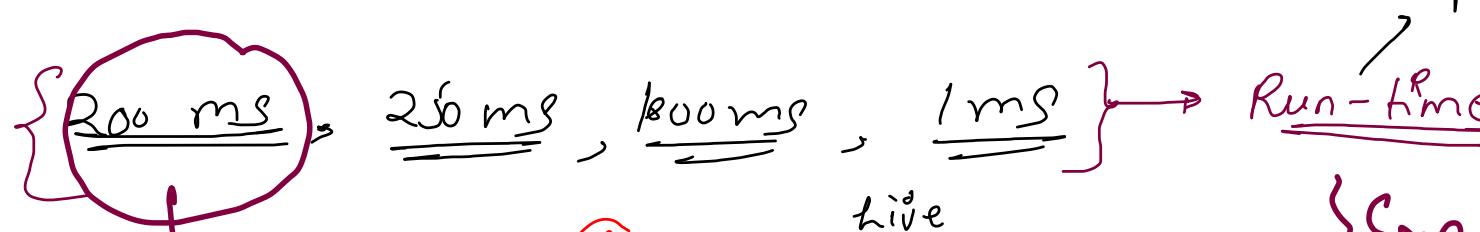
vs Asymptotic Time vs Amortized Time

Architecture Neutrality

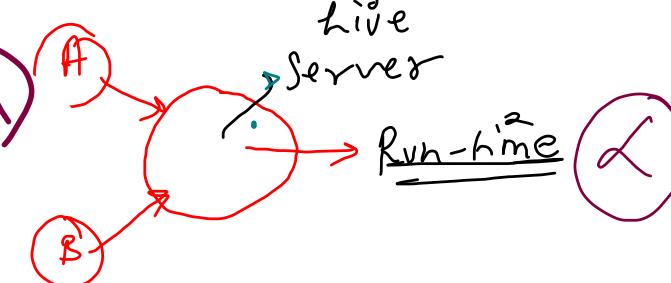
Ignoring constant/smaller terms.
Ignoring small input trend;
{ Always took for large input }
eg { $O(n^3)$ vs $O(2^n)$ }

Vector/Arraylist
design

Time Complexity



Program & process (PAM)



Asymptotic Analysis

Runtime
 $T(N)$

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



Algorithm \propto input size

ignore rest of the factors

machine independent

growth of Runtime w.r.t input size

$N=100$ ele \Rightarrow Runtime

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

$$A \rightarrow 10 \text{ ms} \xrightarrow{*10} 100 \text{ ms}$$

$$B \rightarrow 20 \text{ ms} \xrightarrow{*10} 200 \text{ ms}$$

Algorithm is linearly dependent on input size

Architecture

Machine dependent

32bit 64bit

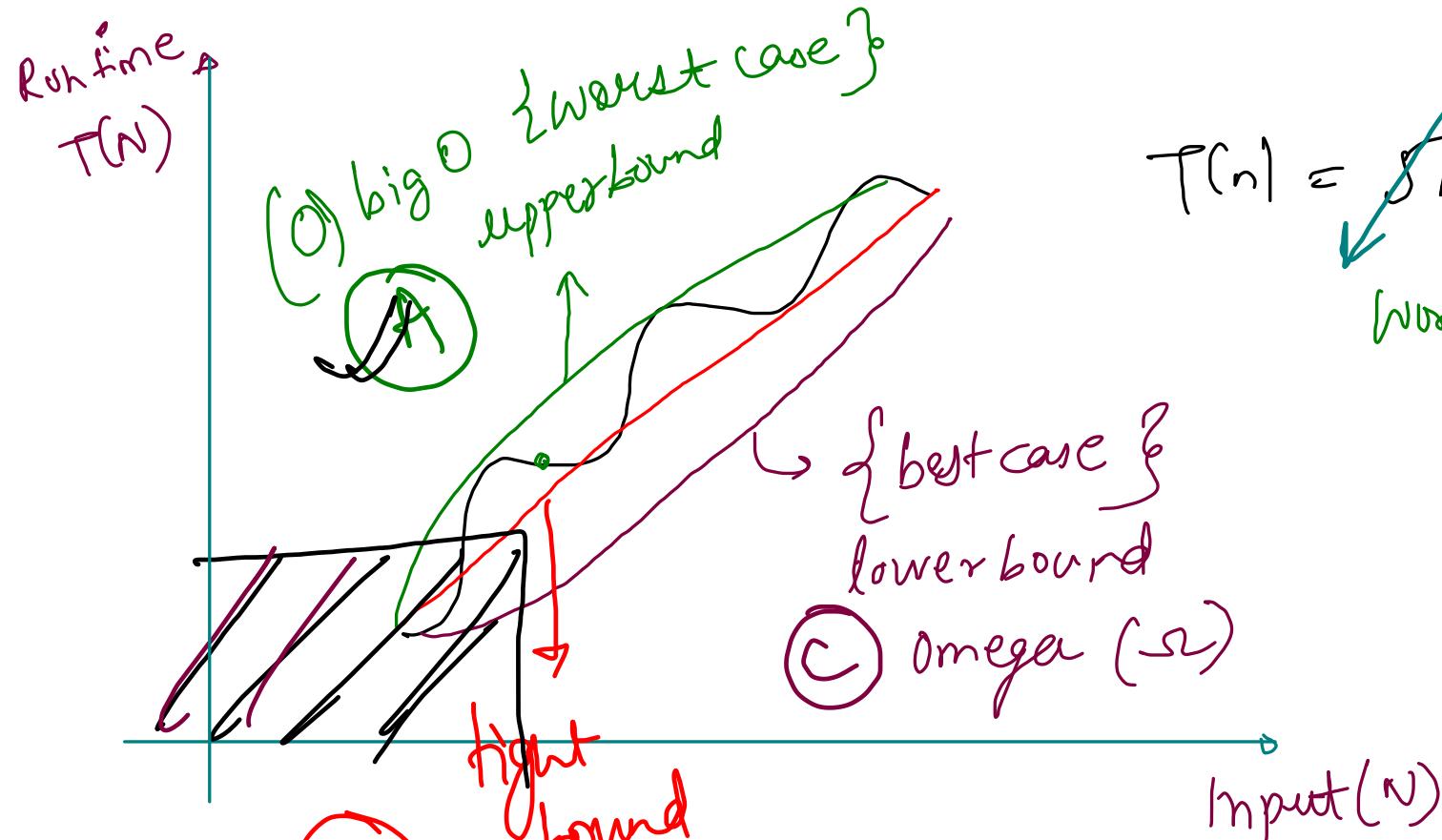
Compilation

+

linking → Packages + Head files

ignore

loading



tight bound $\Rightarrow \Theta(n^2)$

$\Theta(n^2)$

$$T(n) = \Theta(n^2 + 3n + 6)$$

worst case $\Rightarrow \Omega(n^2)$

$T(n) = \Theta(n^2)$ machine

$15n^2 \geq n^2 + 3n + 6$

$10n^2 \geq 3n + 6$

$n > 0$

$\Theta(n^2) = n^2$

$\Theta(n^2) = n^2$

$\Theta(n^2) = n^2$

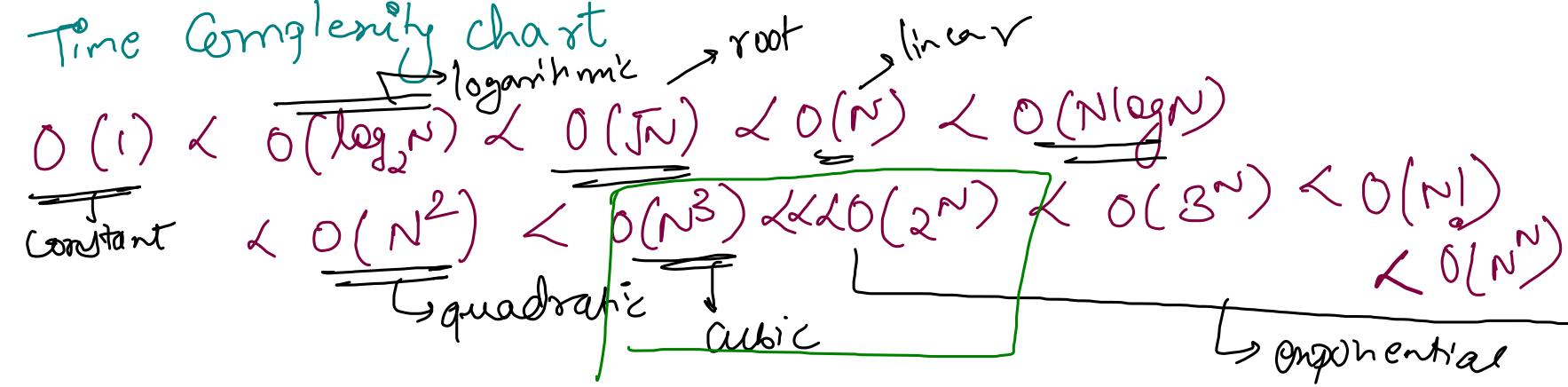
Abdul Basit

BigO \neq worstcase

Omega \neq BestCase

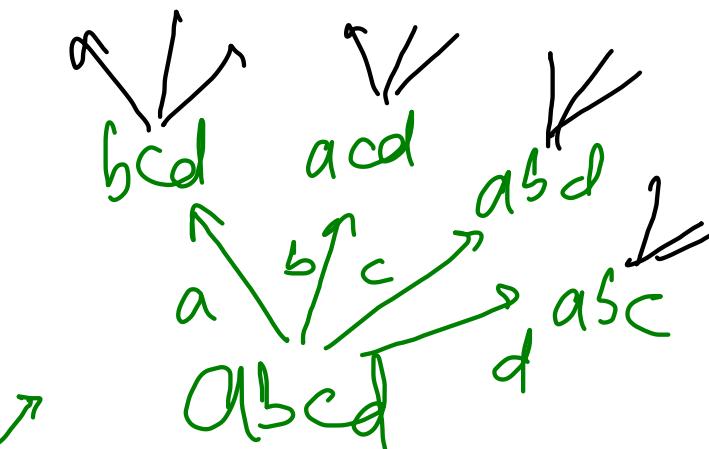
best case : $T(n) = 4n^2$
 $\sim \Omega(n^2)$

Time Complexity chart

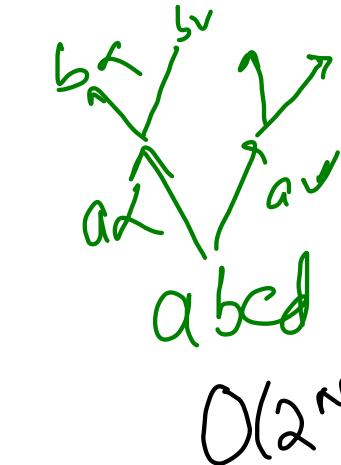


Constant < Logarithmic < Polynomial < Exponential

$$\left\{ \begin{array}{l} O(n^3) < O(2^n) \\ n=2 \quad O(2^3) = O(8) \quad \cancel{O(2^2) = O(4)} \\ n=3 \quad O(3^3) = O(27) \quad \cancel{O(2^3) = O(8)} \\ n=4 \quad O(4^3) = O(64) \quad \cancel{O(2^4) = O(16)} \end{array} \right.$$

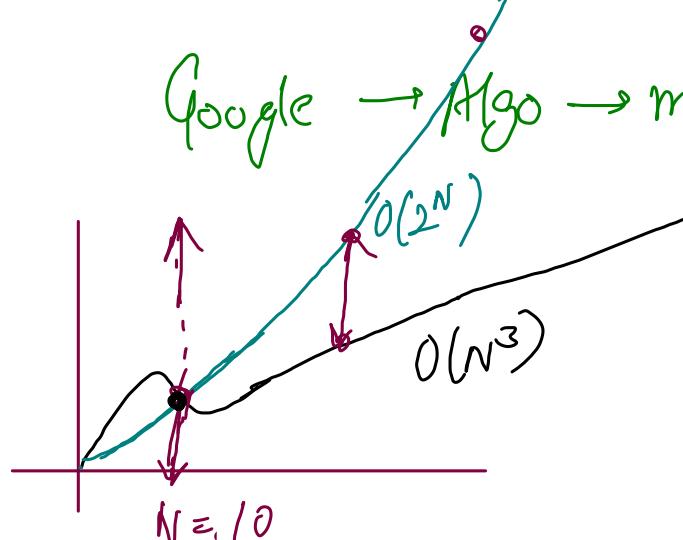


$O(n!)$



$O(2^n)$

Google \rightarrow Algo \rightarrow millions of users ($n \uparrow$)



$n >$ Intersection pt

Large Input
base case ignore

$$T(n) = 4n^3 + 3n^2 \log n + 5n + 10$$

$O(n^3)$

Upperbound(n) = $C * n^3 \geq T(n)$

Runtim^r
 ~~$T(n) = 8n^2 + 12n^2 + 20n^2 + 10n + 30n$~~

worst case $O(n^2)$

For loops Time complexity { Code → Time Complexity }

1

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

$n \times k$
 $\frac{nk}{k}$
 $O(n)$

4

```
for (int i = 0; i < n; i++) {  
    for (int j = 0; j <= i; j++) {  
    }  
}
```

2

```
for (int i = 0; i < n; i++) {  
    for (int j = 0; j < n; j++) {  
        System.out.println(i + " " + j);  
    }  
}
```

5

```
for (int i = 1; i < n; i *= 2) {  
}
```

3

```
for (int i = 0; i < n; i++) {  
    for (int j = n - 1; j >= 0; j--) {  
    }  
}
```

6

```
for (int i = 0; i * i < n; i++) {  
}
```

$$n^3 * O(n \log n) = O(n^4 \log n)$$

```
(2)
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        System.out.println(i + " " + j); → kms
    }
}
```

$$n = 5$$

$$\begin{aligned} & \left(f(n) \right) O(g(n)) \\ &= O(f(n) * g(n)) \end{aligned}$$

$i = 0$	$j = 0, 1, 2, 3, 4$	$(k+k+k+k+k) = O(n)$	$O(n)$	$n \times O(n)$
$i = 1$	$j = 0, 1, 2, 3, 4$	$(k+k+k+k+k) = O(n)$	$+ O(n)$	$= O(n^2)$
$i = 2$	$j = 0, 1, 2, 3, 4$	$(5k) = O(n)$	$+ O(n)$	
$i = 3$	$j = 0, 1, 2, 3, 4$	$(5k) = O(n)$	$+ O(n)$	
$i = 4$	$j = 0, 1, 2, 3, 4$	$(5k) = O(n)$	$n \times O(n)$	

\vdots

$= O(n \times n)$

$= O(n^2)$

3

```
for (int i = 0; i < n; i++) {  
    for (int j = n - 1; j >= 0; j--) {  
        sysk(i+j)  
    }  
}
```

1-20

$$j \in \{3, 2, 1, 0\}$$

12

Σ 4, 3, 2, 1, 0

112

$\int \in 4, 3, 2, 1, 0$

3

$$j \in \{4, 3, 2, 1, 0\}$$

14

$$j \in \{4, 3, 2, 1, 0\}$$

A hand-drawn diagram in green ink. On the left, there is a capital letter 'N' written vertically. A horizontal line segment extends from 'N' towards the right. This line segment meets a circle at its midpoint. The circle is drawn with a thick green outline. Inside the circle, near the point of intersection, there is a small, roughly triangular mark.

A hand-drawn diagram illustrating a computational complexity analysis. On the left, a green circle contains the letter 'N'. A curved arrow points from this circle to the text 'R * 25 \rightarrow O(n^2)'.

$$k + k + k + k =$$

k f t f k f k f k

$$k \cdot \cancel{k} \cdot \cancel{\int} \quad \in \mathcal{O}(n)$$

$$b^*j' = d_2$$

$\text{rank } \mathcal{G} \leq O(\sqrt{n})$

$O(n)$

+

nime

$$= O(n^2)$$

④

```
for (int i = 0; i < n; i++) {  
    for (int j = 0; j <= i; j++) {  
        Sys(i+j)  
    }  
}
```

$$N = \sum_{i=0}^n i$$

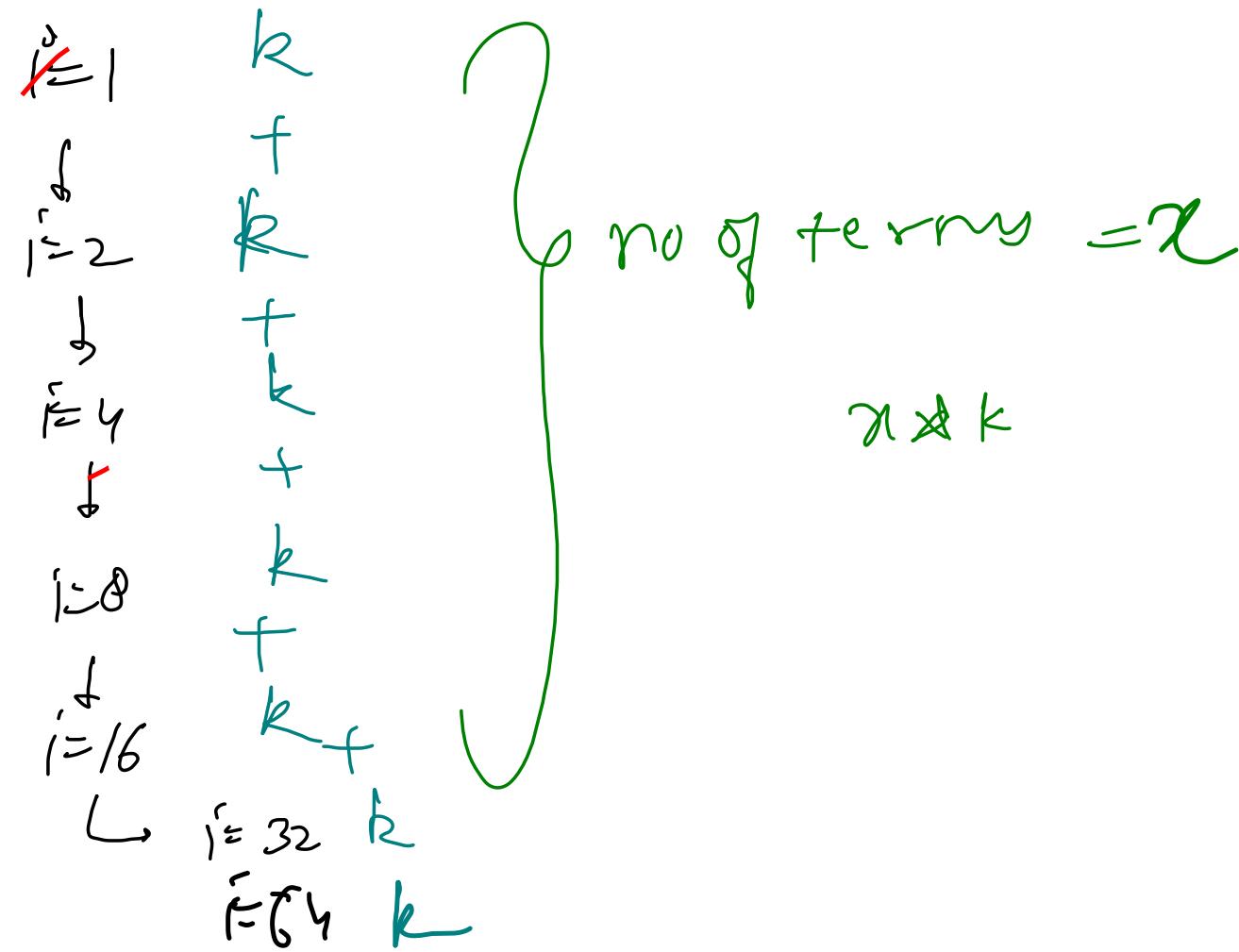
for loop for
is not
always

$i=0$	$j=0$	$k = 1 * k$	$1k + 2k + 3k + 4k + 5k$	$= \frac{n(n+1)}{2} * k$
$i=1$	$j=0, 1$	$k+k = 2 * k$	$= (1+2+3+\dots+n) * k$	
$i=2$	$j=0, 1, 2$	$k+k+k = 3 * k$	$= \frac{n(n+1)}{2} * k$	
$i=3$	$j=0, 1, 2, 3$	$k+k+k+k = 4 * k$	$= \frac{n(n+1)}{2} * k$	
$i=4$	$j=0, 1, 2, 3, 4$	$k+k+k+k+k = 5 * k$	$= \frac{n(n+1)}{2} * k$	

$$= O(n^2)$$

⑤

```
for (int i = 1; i < n; i *= 2) {
    Sys0(i)
}
```



$$N = \text{input size}$$

$$N = \cancel{64}$$

$$\log_2 N \leq \sum_{i=0}^{\lfloor \log_2 N \rfloor} 2^i$$

$$N = \cancel{64} 2^{30}$$

$1, 2, 4, 8, \dots n$

$$a=1, r=2$$

$$a r^{n-1} = n$$

$$1 \cdot (2)^{n-1} = n$$

$$\log_2 (2)^{n-1} = \log_2 n$$

$$\Rightarrow n-1 = \log_2 n$$

$$\Rightarrow n = \log_2 n + 1$$

$$\chi \propto k \Rightarrow O(\log_2 n)$$

⑥

```
for (int i = 0; i * i < n; i++) {  
}
```

- 64

$i=0 \quad k \quad 0*0 < 64$

$i=1 \quad k \quad 1*1 < 64$

$i=2 \quad k \quad 2*2 < 64$

$i=3 \quad k \quad 3*3 < 64$

$i=4 \quad k \quad 4*4 < 64$

$i=5 \quad k \quad 5*5 < 64$

$i=6 \quad k \quad 6*6 < 64$

$i=7 \quad k \quad 7*7 < 64$

no of term = 8

$\hookrightarrow O(\sqrt{n})$

$O(n+m)$

7

```
for (int i = 0; i < n; i++) { }  $O(n)$ 
}  $O(m)$ 
for (int j = 0; j < m; j++) { }  $O(n+m)$ 
}
```

g

for (int i = 0; i < n; i++) {
 for (int j = 0; j < 4; j++) {
 \downarrow non-terminating
 }
}

$$= 4n$$

$$= O(n)$$

8

```
for (int i = 0; i < n; i += 2) { }
```

$$N = 20$$

10

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j *= 2) {
         $\downarrow$  infinite
    }
}
```

$j = 0 \times 2$
 $= 0 \times 2$ $N = 5$
 $= 0 \times 2$
 $= 0$

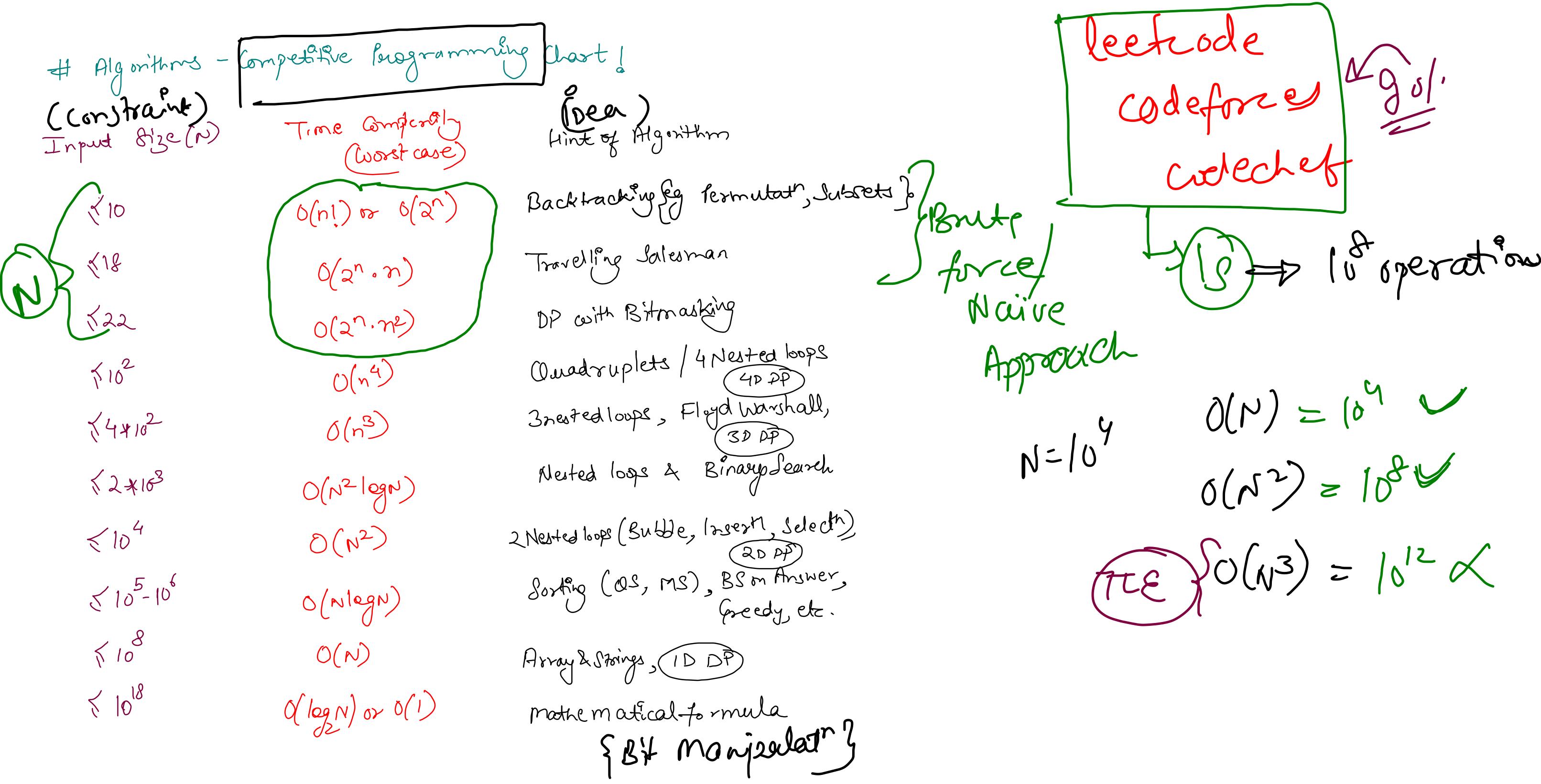
$O(\text{inf})$

non-terminating

$\{0, 2, 4, 6, 8, 10, 12, 14, 16, 18\}$

\downarrow non-terminating

$$O(N/2) \approx O(0.5n) = O(n)$$



Recursion Generic Formula

99%

$$\text{TC} = (\text{No of calls}) \xrightarrow{\text{height}} + (\text{preorder} + \text{postorder}) * \xrightarrow{\text{height}}$$

if else
|| &

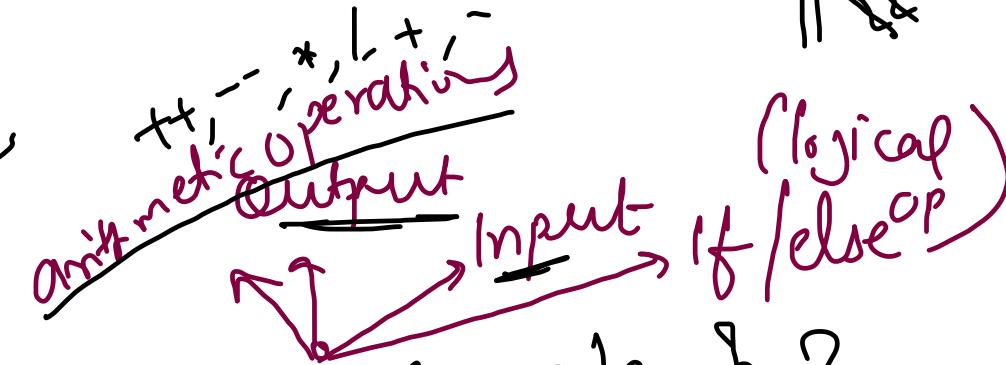
→ Recursion Basics (Factorial, First & Last node, max, Print I/O/ID)

→ Tower of Hanoi

→ Power - I, II, III

→ Get & Print → maze Paths
 ↓ Subsets → Stair Paths
 Keypad Encoding
 Permutations

→ Backtracking → Flood fill
 N Queen
 Knight's Tour



Constant Time Complexity?

Refm → Base Case
 # NP-Hard & NP-Complete Problems.

eg Subset Sum
 Hamiltonian Path & Cycle
 Travelling Salesman
 Graph coloring

```

public static void main(String[] args) throws Exception {
    // write your code here
    Scanner scn = new Scanner(System.in); → k₁
    int n = scn.nextInt(); → k₂
    int[] arr = new int[n]; → k₃
    for(int i=0; i<n; i++)
        arr[i] = scn.nextInt(); → O(n)

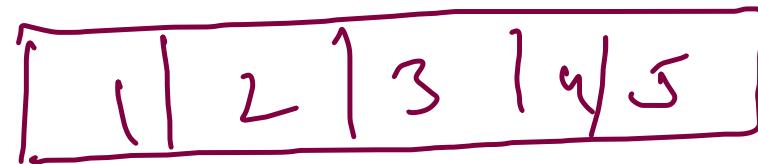
    displayArr(arr, 0);
}

public static void displayArr(int[] arr, int idx){ } ← O(n)
    // Base Case
    if(idx == arr.length) return;

    // 1. Print arr of idx
    System.out.println(arr[idx]);

    // 2. Faith
    displayArr(arr, idx + 1);
}

```



Base Case $i_{dn} = 5$

$i_{dn} = 4$

$i_{dn} = 3$

$i_{dn} = 2$

$i_{dn} = 1$

$i_{dn} = 0$

TC

$$(1)^N + \{k + C\} * N$$

$O(n)$

$$= 1 + kN$$

Recurrence Relations

→ without code logic
dry run
Recursion tree

$$T(n) = T(n-1) + O(1)$$

- ① linear search, Power - linear
- ② Binary Search (Divide & Conquer), Power - logarithmic
(Tower of Hanoi, Print zig zag)
- ③ Power - III $T(n) = 2T(n/2) + O(1)$

```

public static int power(int x, int n){
    // Base Case: x^0 = 1
    if(n == 0) return 1;

    // 1. Calculate x ^ n-1 using Faith
    int xpn1 = power(x, n-1);

    // 2. Calculate x ^ n using x ^ n-1
    int xpn = xpn1 * x;

    // 3. Calculate x ^ n
    return xpn;
}

```

```

public static int power2(int x, int n){
    // Number of calls in power2 : Homework
    // Base Case: x^0 = 1
    if(n == 0) return 1;

    // 1. Faith : x ^ n/2
    int xpn2 = power2(x, n/2);

    // 2. x^n = x^n/2 * x^n/2
    int xpn = xpn2 * xpn2;

    // 3. If n is odd
    if(n % 2 == 1) xpn = xpn * x;

    // 4. Return
    return xpn;
}

```

$$T(n) = T(n-1) + O(1)$$

Expectant = ~~faith~~ + meeting expectant
 (x^n)

$$T(n) = T(n-1) + k$$

$$T(n-1) = T(n-2) + k$$

$$T(n-2) = T(n-3) + k$$

$$T(n-3) = T(n-4) + k$$

$$\vdots$$

$$T(1) = T(0) + k$$

base case

$$T(n) = \underbrace{T(0)}_{O(1)} + n * k$$

$$T(n) = k_2 + n * k_1$$

$$O(n)$$

$$T(n) = T(n/2) + O(1)$$

x^n
Exp
 \downarrow
 $x^{n/2}$
Faith
 \uparrow
constant
 \uparrow
P.M.

$$T(n) = T(n/2) + k$$

$$T(n/2) = T(n/4) + k$$

$$T(n/4) = T(n/8) + k$$

$$T(n/8) = T(n/16) + k$$

$$\vdots$$

$$T(1) = T(0) + k$$

$$T(n) = T(0) + x * k$$

$$= T(0) + \log_2 n * k$$

$$= O(\log_2 n)$$

$$2^0 T(n) = \cancel{2^1 T(n/2)} + k$$

$$\cancel{2^1 T(n/2)} = 2^2 T(n/4) + 2^1 k$$

$$\cancel{2^2 T(n/4)} = 2^3 T(n/8) + 2^2 k$$

$$2^3 T(n/8) = 2^4 T(n/16) + 2^3 k$$

$$2^x T(1) = 2^{x+1} T(0) + 2^x k$$

$$T(n) = 2^{x+1} T(0) + \{k + 2^1 k + 2^2 k + 2^3 k + \dots + 2^x k\}$$

$$\{x = \text{No of terms} = \log_2 n\}$$

$$= 2^{x+1} T(0) + k \{ 2^0 + 2^1 + 2^2 + \dots + 2^x \}$$

$$= 2^{x+1} T(0) + k * (2^x)$$

$$\sum GP = \frac{a(r^x - 1)}{r - 1}$$

$$= 2^{\log_2 n} k' + k * 2^{\log_2 n}$$

$$= n k' + k * n = O(n)$$

$$4 \rightarrow O(n^2)$$

$$T(n) = T(n-1) + O(n)$$

$$6 \rightarrow O(2^n)$$

$$T(n) = 2T(n-1) + O(1)$$

$$8 \rightarrow O(\sqrt{n})$$

$$5 \rightarrow O(n \log n)$$

$$T(n) = T(n-1) + O(\log_2 n)$$

$$7 \rightarrow O(n \log n)$$

$$T(n) = T(n/2) + O(n)$$

$$T(n) = T(\sqrt{n}) + O(1) \{ \text{Root-function} \}$$

Recurrence Relation ?

$$T(n) = T(\sqrt{n}) + O(1)$$

$$T(n^{\frac{1}{2}}) = T(n^{\frac{1}{4}}) + O(1)$$

$$T(n^{\frac{1}{4}}) = T(n^{\frac{1}{8}}) + O(1)$$

$$T(n^{\frac{1}{8}}) = T(n^{\frac{1}{16}}) + O(1)$$

$$T(n^{\frac{1}{16}}) = T(n^{\frac{1}{32}}) + O(1)$$

$$T(1) = T(0) + O(1)$$

no of calls

$$T(n) = T(\cancel{\delta}) + k * \cancel{x}$$

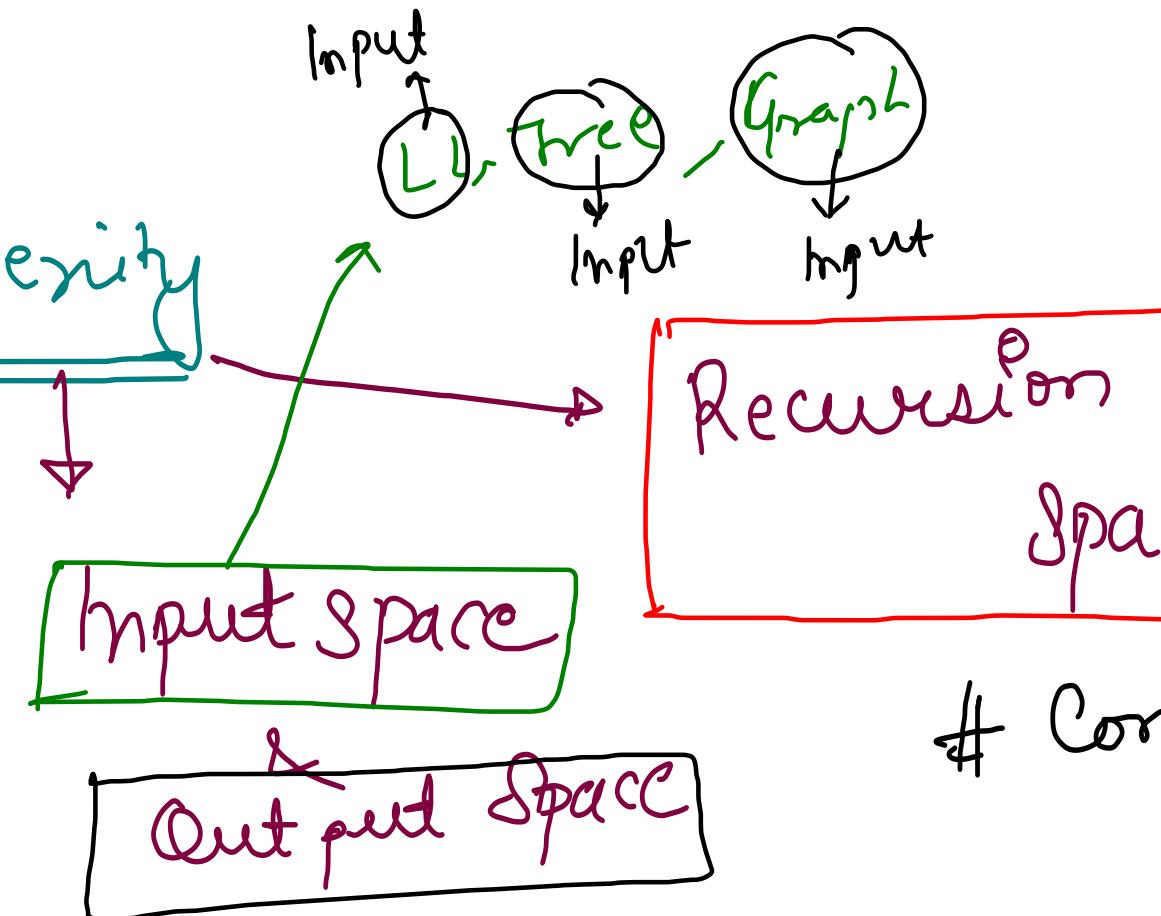
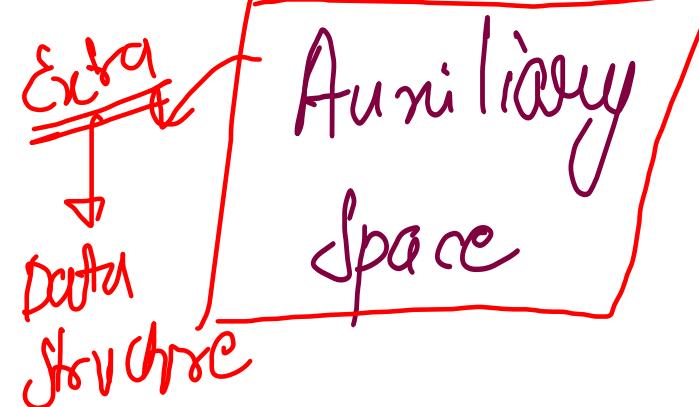
$$n, n^{1/2}, n^{1/4}, n^{1/8}, \dots 1$$

$256, 16, 4, 2, 1$

$$2^8, 2^4, 2^2, 2^1, 2^0$$

No of fty & Input size

Space Complexity

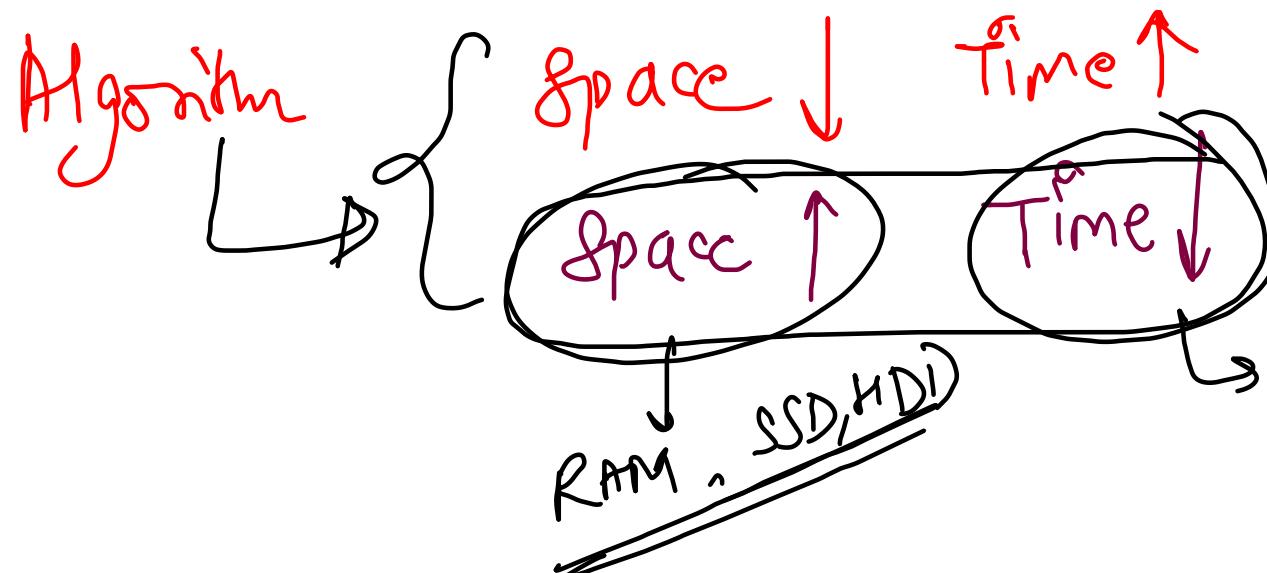


Constant Extra space ?

int(4) $\xrightarrow{4 \times 26}$ int(?) $\xrightarrow{?}$ double

"Primitive data types"

Time vs Space Tradeoff!



Master Theorem

Decreasing
Functions

Recurrence Relation

Master theorem for Decreasing Functions

$$T(n) = aT(n-b) + f(n)$$

$a > 0$ $b > 0$ and $f(n) = O(n^k)$ where $k \geq 0$

Case

1. if $a < 1$

$$\begin{cases} O(n^k) \\ O(f(n)) \end{cases}$$

$$T(n) = T(n-1) + O(n)$$

2. if $a = 1$

$$\begin{cases} O(n^{k+1}) \\ O(n * f(n)) \end{cases}$$

$$T(n) = T(n-1) + O(n \log n)$$

3. if $a > 1$

$$O(n^k a^{n/b})$$

$$T(n) = 2T(n-2) + 1 \rightarrow O(2^n)$$

$$T(n) = 3T(n-1) + 1 \rightarrow O(3^n)$$

$$T(n) = 2T(n-1) + n \rightarrow O(n2^n)$$

Master Theorem

Dividing
Function

Master Theorem for Dividing Functions

$$\textcircled{1} \log_b^a$$

$$\textcircled{2} K$$

$$T(n) = aT(n/b) + f(n)$$

$$\begin{array}{l} a \geq 1 \\ b > 1 \end{array} \quad f(n) = \Theta(n^k \log^p n)$$

Case 1: if $\log_b^a > k$ then $\Theta(n^{\log_b^a})$

Case 2: if $\log_b^a = k$

$$\begin{array}{ll} \text{if } p > -1 & \Theta(n^k \log^{p+1} n) \end{array}$$

$$\begin{array}{ll} \text{if } p = -1 & \Theta(n^k \log \log n) \end{array}$$

$$\begin{array}{ll} \text{if } p < -1 & \Theta(n^k) \end{array}$$

Case 3: if $\log_b^a < k$

if $p \geq 0$	$\Theta(n^k \log^p n)$
if $p < 0$	$\Theta(n^k)$