# Data Structure & Algorithms

(Problem Solving Skill)

## Data Structure

(Logic)

→ efficiently Store the data

Python

Linear

- Arrays
- Linked List
- Stack & Queue
- HashMap (Dictionary)

Non-Linear

- Tree
- Graphs
- Heap

Leetcode

Recursion & Backtracking

- Divide & conquer
- Greedy Algorithm
- Dynamic Programming

## Algorithm

Logical Building

→ Set of instructions to Solve a specific Problem

→ Independent of any programming Language

Data Science (Projects)

DSA

Coding

Technical Interview

Easy - Intermediate

↳ Sum of n natural numbers

$1+2+3+4+5+6 = 21$

$n = 6$

Hire

S1

sum = 0

for i = 1 to 6

sum += i

n

return sum

O(n)

S2

$\frac{6 * 7}{2} = 21$

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

return sum

O(1)

approximation

Constant

## Time Complexity

Amount of time to execute a program

→ Less time complexity

(Perfection)

Ideal situation → ↓↓ Time complexity + ↓↓ Space Complexity

## Space complexity

Extra space to execute a program

→ Less space complexity

## Tradeoff

Situation 1

time ↑↑ + Space ↓↓

Situation 2

time ↓↓ + Space ↑↑

Module 1

# Analysis Module

→ optimized code

↓

Great- Developer → time ↓, & space ↓↓

↓

## Apriori Analysis

↳ O(n)

↳ Approximations & Logic

↓

Not dependent on hardware

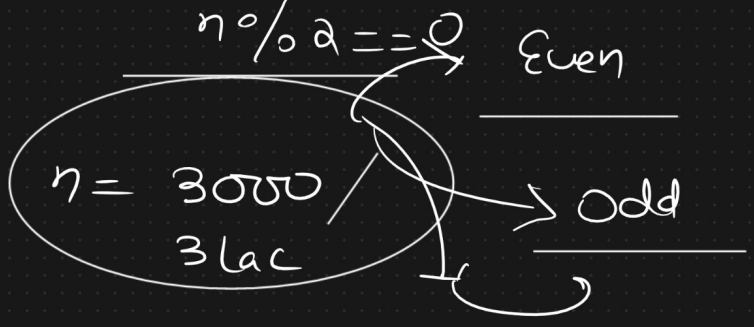
## Research

## Apostiary Analysis

ns

↳ Exact- results & type of hardware

↳ supers computers



Apriori Analysis

(Standard Books)

Cormen

↓

Time & Space complexity

↑

Time complexity → Order of magnitude of the statement

↓

# times any statement is running

n → very very high

Example 1

```
y = 7
z = 8
x = y + 2   → O(1)
```

Notation (Worst case)

Constant time complexity

Example 2

i = 0 to n-1

```
for i in range(n):
    x = y + 2   → n-1 times
```

O(n) → time complexity

| i=0 | i=1 | i=2 | i=n-1 |
|---|---|---|---|
| x=y+2 | x=y+2 | x=y+2 | x=y+2 |

Example 3

x=y+2 ——— c

```
for i in range(n):
    x=y+2 ——— n
```

c + n + n²

⇒ O(n²)

```
for i in range(n):
    for j in range(n):  } — n²
        x=y+2
```

n = 3 — (3)² = 9

| i=0 | | | i=1 | | | i=3 | | |
|---|---|---|---|---|---|---|---|---|
| j=0 | j=1 | j=2 | j=0 | j=1 | j=2 | j=0 | j=1 | j=2 |
| x=y+2 | | | | | | | | |

① Loop (for, while) → time complexity

② Bigger loop → Time complexity

↳ $n + n^2 + n^3 = O(n^3)$

③ No loop → Constant time complexity

$O(1)$

Conditional Statement (If elif else)

↙

$O(1)$

Example 4

```
i = n ——— (1)
while (i > 1):
        i = i - 1 ——— (2)
```

True

n = 5

| (1) | (2) | (3) | (4) |
|---|---|---|---|
| i = 5 | i = 4 | i = 3 | i = 2 |
| i = 4 | i = 3 | i = 2 | i = 1 |

n = 5 ——— 4 times

n = 10 ——— 9 times

Apriori analysis

n ——— (n-1) times

O(n)

O(n/2) = O(n)

## Examples

$i = n$

$O(n)$

```
while (i >= 1):
    i = i - 2   (1)
    i = i - 3   (2)
```

$i = i - 5$

$n/5$ times

$n = 10$

$n/2$ times

$i = 10$

| (1) | (2) | (3) | (4) |
|---|---|---|---|
| $i = 8$ | $i = 6$ | $i = 4$ | $i = 2 \geq 1$ |
| $i = 6$ | $i = 4$ | $i = 2$ | $i = 0$ |

Linear Complexity

$25/5 = n/5$ → 5 times $= O(n)$

$n = 25$

| | | | | |
|---|---|---|---|---|
| $i = 25$ | $i = 20$ | $i = 15$ | $i = 10$ | $i = 5$ |
| $i = 20$ | $i = 15$ | $i = 10$ | $i = 5$ | $i = 0$ |

## Example 6

```
i = n
while (i > 2):
    i = i^(1/2)
```

false

Stop

$n = 16$

| $i = 16$ | $i = 4$ |
|---|---|
| $i = (16)^{1/2}$ | $i = (4)^{1/2}$ |
| $= (2^4)^{1/2}$ | $= (2^2)^{1/2}$ |
| $= 2^2 = 4$ | $= 2$ |

$i = n$

1st $\longrightarrow n^{1/2}$

2nd $\longrightarrow (n^{1/2})^{1/2} = (n)^{1/2^2}$

3rd $\longrightarrow (n^{1/4})^{1/2} = n^{1/2^3}$

k times

| LHS | RHS |
|---|---|
| $n^{1/2^k}$ | $= 2$ |

breaking point

$\log_2$ on both sides

$$\log_2 n^{1/2^k} = \log_2 2^1$$

(1) $\log_a n^b = b \log_a n$

(2) $\log_a a = 1$

$$\frac{1}{2^k} \log_2 n = 1$$

$$\log_2 n = 2^k$$

$$\frac{1}{2^k} \log_2 n = 1$$

$$\log_2 n = 2^k$$

$$\log_2(\log_2 n) = k \log_2 2^1$$

$$k = \log_2(\log_2 n)$$

external data structure

Space complexity

↓ arr

↓ Dictionary

↓ O(n)

↓ to store the results

↓ MergeSort

time complexity

Binary search → O(log n)

# Complexity Classes

(Increasing order)

## Standard (Daily basis)

1. Constant complexity → $O(1)$
2. Logarithmic complexity → $O(\log_2 n)$ ← Binary search
3. Linear complexity → $O(n)$

n log n

4. Quadratic complexity → $O(n^2)$
5. Cubic complexity → $O(n^3)$
6. Polynomial complexity → $O(n^c)$

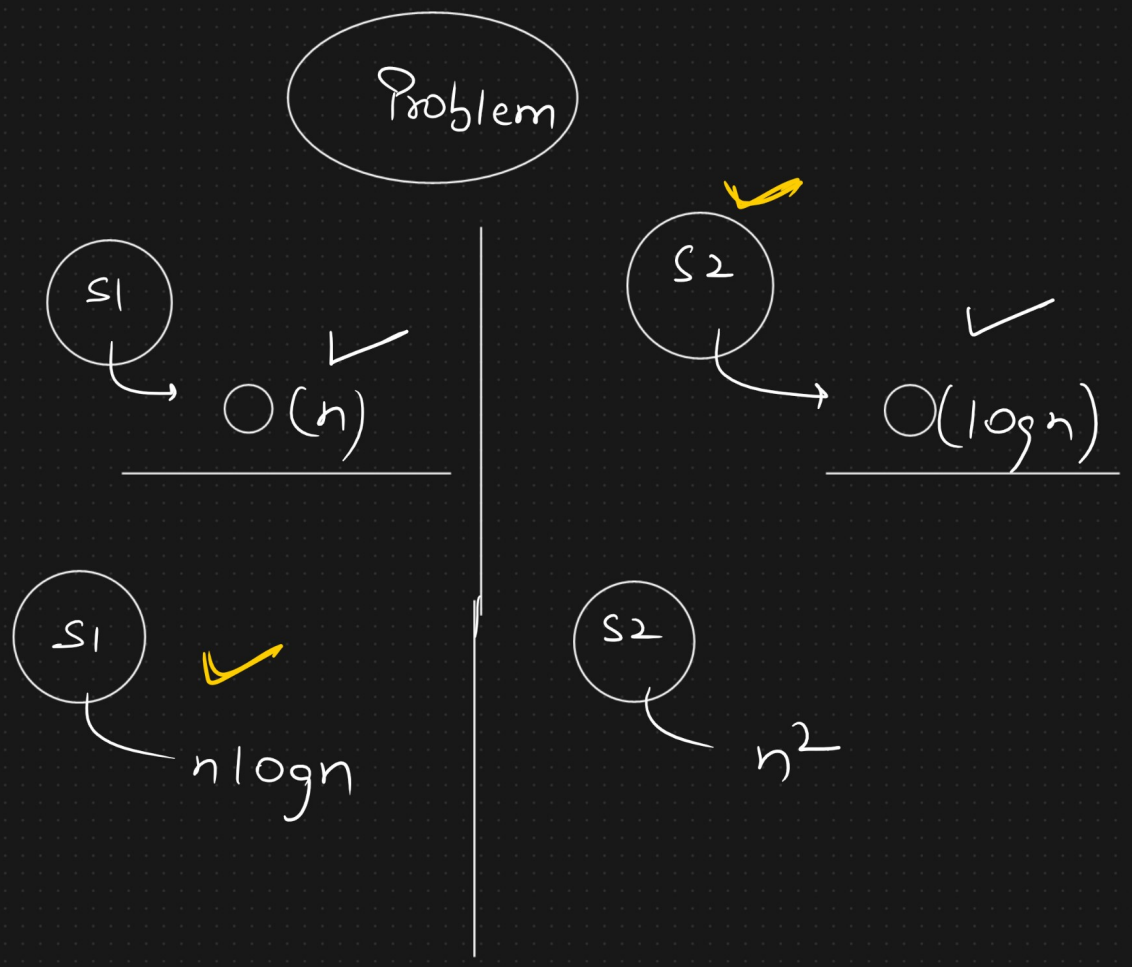
$c > 0$

Recursion

Dynamic Programming

7. Exponential complexity → $O(2^n)$

$O(c^n)$ ← $c > 1$




$\log_2 n$ $\log_3 n$

$\log_2^{10}$ > $\log_3^{10}$

$\log_2 2^3 \approx 3.$

$\log_3 3^2 \approx 2.$

Higher the base, lower the value

DSA + Data Science &

Maths (Expertise)

↳ Number

↳ Number System

↳ Logarithmic

↳ AP & GP Series (formula)

↳ Calculus

↳ Probability & Statistics

↳ Matrix Computation

| S1 | Problem | S2 |
|---|---|---|
| $\log n$ | $<$ | $n$ |
| $(\log n)^2$ | $<$ | $n^2$ |
| $(\log n)^3$ | $<$ | $n^3$ |
| $(\log n)^{\log n}$ | | $n^{\log n}$ |
| $\log n (\log(\log n))$ | $<$ | $\log n (\log n)$ |

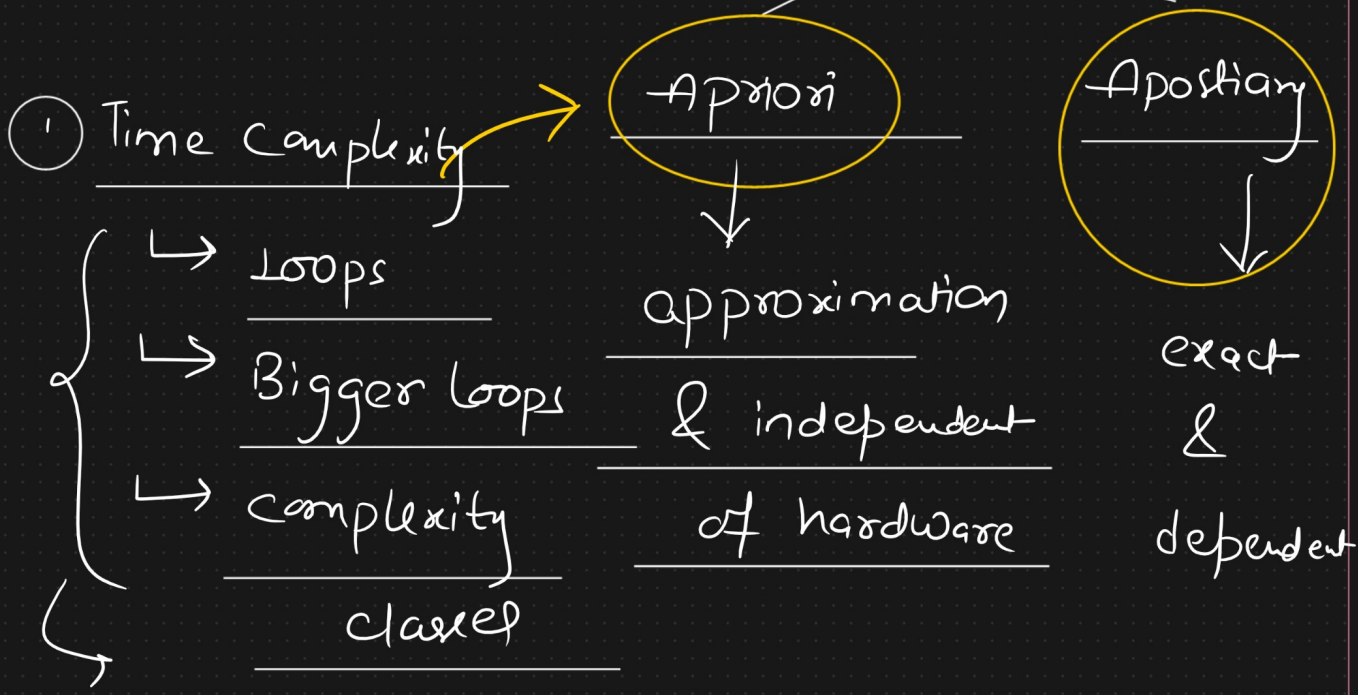
$\log_2(\log_2 10) = 1.73$ $\quad$ $\log(\log_2 10) = 3.32$

$\log_{40} n \quad < \quad \log_{32} n$

$2^n \quad < \quad 3^n$

$\frac{2^n}{2^n} \quad < \quad \frac{(2*1.5)^n}{2^n * (1.5)^n}$

1. Data Structure & Algorithm
2. Linear & Non-Linear
3. Syllabus
4. Module 1 → Analysis




2. Space complexity

# Asymptotic Notation