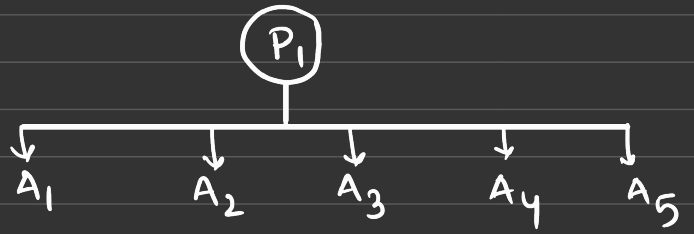
Your Ultimate Guide To Landing Top AI roles

2.3.1

Time and space complexity

→ For solving any problem P_1, there will be many solutions

→ Before implementing any algorithm as a Program we must have to find out which algorithm is good in terms of time and memory.



→ After analysis, we will choose the best algorithm in terms of Time and Space

→ How to calculate time and space complexity?

↳ Asymptotic notation

↳ **Big-O**

Asymptotic Notation

→ Asymptotic notation describes the performance of algorithms in terms of input size(n), particularly for large inputs.

time and space complexity.

→ It gives a mathematical way to express the growth rate of an algorithm's resource (Time and Space) Consumption

→ Types of Asymptotic Notation

① Big-O(O) → Worst-Case time and Space

② Omega(Ω) → Best-Case time and Space

③ Theta (Θ) → Average-Case time and Space

Q. Given a list of numbers of size n. Find the best, average and worst case time complexity to search for number x.

NOTE - List is not sorted.

Time Complexity = T(n)

→ Worst case time complexity ⇒ $O(n)$ ← last/donot exists

Average case time complexity ⇒ $\Theta(n)$ ← target at middle

Best case time complexity ⇒ $\Omega(1)$ ← first position

↳ what about space complexity?

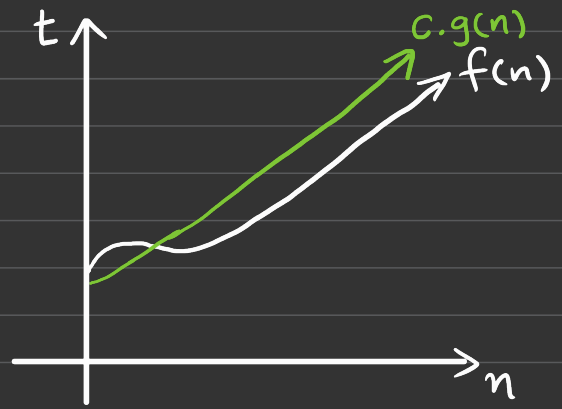
Big-O Notation (O)

→ Big-O defines the maximum time/space an algorithm can take

→ Focus on worst-case scenario.

→ Analogy : Going to Airport

→ we will find another function for a given function such that after some input n_0, the value of $c \cdot g(n)$ is always greater than $f(n)$



$$f(n) \leq c \cdot g(n)$$

for $n \geq n_0$

$c > 0$, $n_0 \geq 1$

$$f(n) = O(g(n))$$

→ Big-O is an upper bound, not necessarily a tighter upper bound.

Example : $f(n) = 3n + 2$

→ $g(n) = n$

$f(n) \leq c \cdot g(n)$, $c > 0$

$n_0 \geq 1$

$3n + 2 \leq c \cdot n$

Put $c = 4$

$3n + 2 \leq 4 \cdot n$

$2 \leq n \Rightarrow n \geq 2$

$\therefore \quad f(n) = O(g(n)) = O(n)$

Shortcut

① keep the highest order term

→ Drop all the lower-order terms — they become insignificant for large input size ($n \to \infty$)

② Ignore Constants

→ Coefficients don't affect asymptotic growth

$T(n) = 100n \longrightarrow O(n)$

Comparison of Functions → To find highest order term

→ when we compare two functions, we are interested in values at $n \to \infty$ (i.e. when n grows very large, which one wins)

Examples

① n ☐ n^5

② 3^n ☐ 2^n

③ 2^n ☐ n^2

④ n^2 ☐ $n \cdot \log n$

⑤ n ☐ $(\log n)^{100}$

⑥ $n^{\log n}$ ☐ $n \cdot \log n$

⑦ $\sqrt{\log n}$ ☐ $\log \log n$

TRICK

① Take larger n value

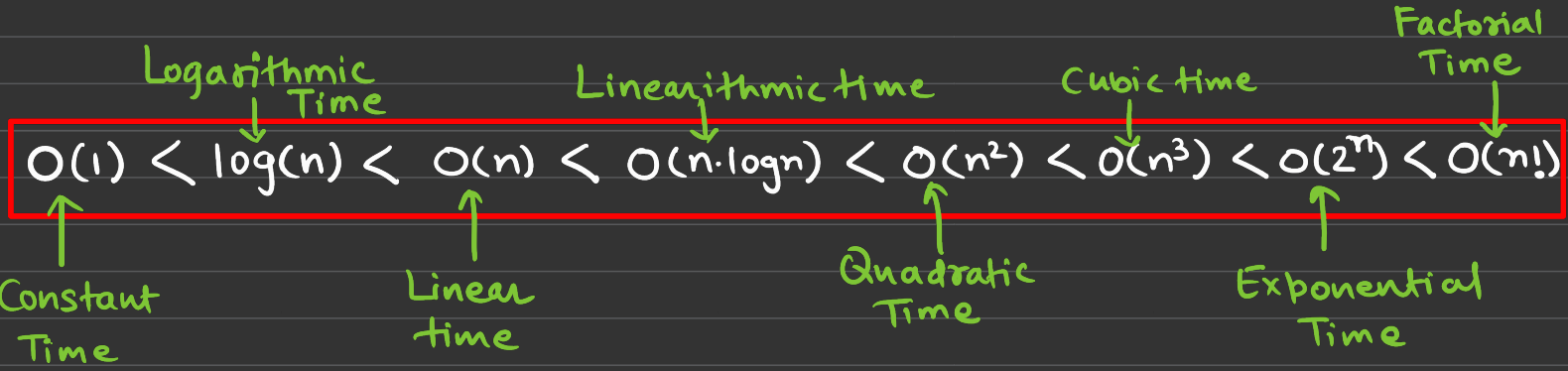
② Apply log and then ①

⑧ $n^{\sqrt{n}} \square n^{\log n}$

⑨ $f(n) = \begin{cases} n^3 & 0 < n < 10000 \\ n^2 & n \geq 10000 \end{cases}$

$g(n) = \begin{cases} n & 0 < n < 100 \\ n^3 & n \geq 100 \end{cases}$

Function Growth (Fastest to Slowest)




DECODE
AiML
Like
Subscribe