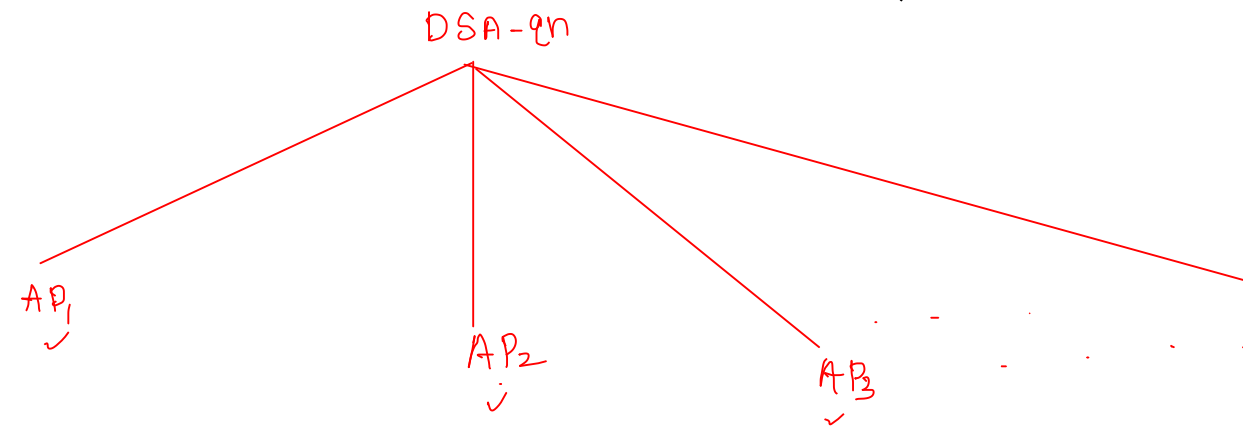


Time and Space Complexity-1

* Why to study Time Complexity?

best?
↳ efficient (\checkmark T, \checkmark space)



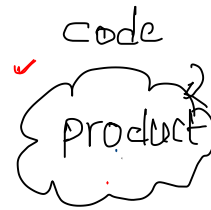
store u_1 { $AP_1 \rightarrow TLE$
 $AP_2 \rightarrow x$

✓ Analysis of algorithms is important for two reasons

1. To estimate the efficiency of the given algorithm
2. To find a framework for comparing the algorithms or solutions for the given
problem.

① ^{cc} Apriori analysis V/S ^{ss} Posteriori analysis

Test 1:



System

MAC ✓	win/len ✓	linux/HP
✓ ✓ ↓	8GB i3	16GB i7
1sec ✓	10sec ✓ ✓	5sec ✓ ✓

PF - Indep

pre = before
post = after

PF-Dep

⇒ release ✓

post

Test 2:-

Run x

Without running, your code you should say whether it is efficient or not?

How?

① **Apriori Analysis (Mathematical Analysis):** Apriori analysis is conducted before the algorithm is translated into a program. It involves estimating the running time by counting the number of executions of the dominant operations within the algorithm. This analysis helps in understanding the efficiency of the algorithm in terms of time complexity without the need to run it on actual data.

② **Posteriori Analysis:** Posteriori analysis, on the other hand, is performed after the algorithm has been implemented as a program. This involves executing the program using standard datasets to empirically estimate its running time and space requirements. Posteriori analysis provides practical insights into the performance of the program in real-world scenarios.

Asymptotic Notations

adjective. of or referring to an asymptote. (of a function, series, formula, etc) approaching a given value or condition, as a variable or an expression containing a variable approaches a limit, usually infinity.

a given value or condition, as a variable or an expression containing a variable approaches a limit, usually infinity.

$$f(x) = \frac{1}{x}$$

$$\lim_{x \rightarrow \infty} f(x) = ?$$

approaching a given value or condition, as a variable or an expression containing a variable approaches a limit, usually infinity.

$$f(x) = \frac{1}{x}$$

$$\frac{1}{0} = \infty$$

$$\lim_{x \rightarrow \infty} f(x) =$$

$$\lim_{x \rightarrow \infty} \frac{1}{x}$$

$$= \frac{1}{\infty} = \frac{1}{\frac{1}{0}} = \frac{0}{1} = 0 \checkmark$$

when x approaches to ∞

DSA:-

Very Very small

✓ Asymptotic Analysis:-

Means, we should consider input as infinite

as possible

Size as large

practically \leftarrow
infinite cannot exist

✓ size of the input :-

Constraints:-

① $1 \leq n \leq 10^9$ ✓ \leftarrow refer 'n' value

② $1 \leq \text{arr}[i] \leq 10^5$
ele's value

it means we have to analyse by taking larger input
in a problem?

Finding T.C of code :-

code

① cannot run & check the time

② for that we do "Asymptotic analysis"

↳ i.e our i/p should be as large as possible.

* Goal : is to Simplify analysis of running time by getting rid of "details" ✓

* A technique that focuses analysis on the " significant term "

✓ *Capturing the essence : how the running time of an algorithm increase with the size of input

$$f(n) = 2n^2 + 3n + 5 \Rightarrow 2n^2$$

-ve value?

$$f(n) = \underbrace{2n} + \overbrace{\log_2 n} \Rightarrow 2n.$$

30-mins

1) There are some symbols there, which are used to represent the time complexity of a program

2) $\Delta, \delta, \tau, \phi, \varphi, \theta, \beta, \alpha, \lambda, \mu, \dots$

✓ 3) O [Big-Oh]

* Definition we will see it in next class

* Big-Oh Notation [$O(\)$] : read it Order of

→ $O(n)$: order of n (or) Big-oh of n

$O(n^2)$: order of n^2 (or) Big-oh of n^2

$$f(n) = \underline{n^2} + \underline{n} + 5 \rightarrow \overset{\text{sig. term}}{\text{cloud}} \{n^2\} \rightarrow O(n^2)$$

Big-oh gives or tells upper bound ✓

↳ Highest n^{th} degree term / Dominated term

$$U_1 \cdot$$
$$* \quad \frac{1 + 2 + 3 + \dots + n}{2} = \frac{n(n+1)}{2} = \frac{n^2 + n}{2} \Rightarrow O(n^2)$$

$$* \quad 1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6} \Rightarrow O(n^3)$$

$$f(n) = n^2 + n \cdot \log n + 2n \Rightarrow O(\checkmark)$$

- ✓ 1. Algorithms T.C is very much related to functions in maths
- ✓ 2. The following functions are commonly used in Algorithms

99%

Sno	Function Name	Function Expression
1	Constant	$1 \rightarrow O(1)$
2	Logarithmic	$\log(n)$ ✓
3	Square root	\sqrt{n} ✓
4	Linear	n ✓
5	Linearithmic	$n \cdot \log(n)$ ✓
6	Quadratic	n^2 ✓
7	Cubic	n^3 ✓
8	Exponential	2^n ✓
9	Factorial	$n!$ ✓

best

worst

One thing that is sure :-

Time complexity of any program cannot be in particular units

like : sec, nano sec, msec, etc....

Why? because we are not running the program to know the time

but still we should say which program is efficient?

by using Asymptotic analysis

Summary:-

By doing AS-Analysis
1. C₁ → (How)
 <Next class>

Tc₁ : $O(n)$

✓ C₂ →

Tc₂ : $O(n^2)$

✓ C₃ →

Tc₃ : $O(\sqrt{n})$

✓ C₄ →

Tc₄ : $O(\log_2^n)$

* *
compare and tell
which is best?

$O(1)$, $O(\sqrt{n})$
 $O(n)$
 $O(n \log n)$, $O(\log^n)$
 $O(n^2)$

Comparison among various functions :

$$f(n) = \sqrt{n} \checkmark$$

$$g(n) = \underline{n} \checkmark$$

-> see if any common terms are there, if yes cancel them



-> ^{*} substitute very large values of n

$f(n)$	$g(n)$
\sqrt{n}	$\sqrt{n} \cdot \sqrt{n}$
$\frac{1}{\sqrt{n}}$	$\sqrt{n} \checkmark$

$\therefore g(n)$ bigger

Hence : $g(n)$ is worst

Q1)

$f(n) = n^2$	$g(n) = n^3$	<u>n: size of input</u> ↳ -ve
n^2	$n^3 * n$	
		

$\therefore g(n)$ is big, Hence it is worst.

Q2)

$f(n) = n \log_2 n$	$g(n) = n^2$
$n \cdot \log_2 n$	$n \cdot n$
$\underbrace{\log_2 n}_{+}$	\underbrace{n}_{+}

Remember:

n power very very small value

v/s logn power very very very large value

$n^{0.00001}$
bigger.

>

$(\log_2^n)^{10000000000}$
best.

Q) \sqrt{n} v/s \log_2^n
 \downarrow
 $n^{0.5}$ > \log_2^L
best.

Qn) $f(n) = \sqrt{n}$, $g(n) = \log_2^n$, $h(n) = n$

Arrange above f's best to worst.

a) f, g, h

✓ b) g, f, h

c) h, f, g

d) h, f, g

$$\log_2^n < \sqrt{n} < n$$

best \rightarrow worst

Q)

$$f_1(n) = \sqrt{n}$$

$$f_2(n) = \log_2 n$$

$$f_3(n) = n \cdot \log_2 n$$

Always s best

$$\log_2 n < \underbrace{\sqrt{n}} < n \cdot \log_2 n$$

$$\sqrt{n}$$

$$\sqrt{n}$$

~~$n \cdot \log_2 n$~~
 $\underbrace{\quad}_{\text{better}} \quad * \log_2 n$

Whenever logn is there, that does not mean that it is always BEST

best
 $\underbrace{\sqrt{n} \quad \log_2 n}$

Consider the following three functions.

$$\sqrt{n} \cdot \sqrt{n}$$

$$\log_{10}^n \log_{10}^n$$

$$f_1 = 10^n \quad f_2 = n^{\log n} \quad f_3 = n^{\sqrt{n}}$$

Which one of the following options arranges the functions in the increasing order of asymptotic growth rate?

☒ A. f_3, f_2, f_1

☒ B. f_2, f_1, f_3

☒ C. f_1, f_2, f_3

☒ D. f_2, f_3, f_1

↳ best to worst

$$\underline{f_2} < \underline{f_1}$$

f_1

f_2

f_3

$$10^n$$

$$n^{\log_{10} n}$$

$$n^{\sqrt{n}}$$

$$f_3 < f_1$$

$$f_2 < f_3$$

f_1
 $\cancel{5n \cdot 5n}$
 $5n$

f_3
 $\sqrt{n} \cdot \log_{10}^n$
 \log_{10}^n

$\underbrace{10^n}$
 $\log_{10}^{10^n}$
 $n \cdot \log_{10}^{10} \rightarrow 1$
 \underbrace{n}_{f_1}

$\underbrace{n^{\log_{10} n}}$
 $\log_{10}^n \log_{10}^n$
 $\log_{10}^n \cdot \log_{10}^n$
 $f_2 \cdot (\log_{10}^n)^2$

$\log_{10}^{n^{\sqrt{n}}}$
 $\sqrt{n} \cdot \log_{10}^n$
 $\sqrt{n} \cdot \log_{10}^n$

$$\sqrt{n}$$

$$\log n \star \log n$$

$$\underline{\underline{n^{0.5}}} > \frac{(\log n)^{\underline{\underline{2}}}}{\text{best}}$$

#12

best \rightarrow worst-

Consider the following functions:

- $f(n) = 2^n$
- $g(n) = n!$
- $h(n) = n^{\log n}$