# CS & IT ENGINEERING


## Algorithm

### Analysis of Algorithms

Lecture No.- 02


By- Aditya sir

# Recap of Previous Lecture

**Topic** Introduction to course

**Topic** Algorithm Concept

**Topic** Algorithm Lifecycle Steps

why
What
How
Analyse ?

# Topics to be covered

- **Topic** Need to Analysis
- **Topic** Methodology of Analysis
- **Topic** Types of Analysis

(How to analyse?) Methodology of Analysis

Aposteriori

After implementation

Apriori

Before implementation

(2) Apriori Analysis

↳ carried out before the actual implementation of Algorithm.

## Advantages

1) Platform independent.
2) Can be carried out without actual implementation
3) Helps us to compare the relative performance of 2 or more algorithms.

## Disadvantages

1) It gives approximate values
(Time Complexity)
(Space Complexity)

eg:-

Maggie

A1 | A2

$x^2 > x$

A1 $\rightarrow$ $x$ units

A2 $\rightarrow$ $x^2$ units

$x$ units $\rightarrow$ better for large values of $x$.

(comparison can be made with Approximate values) ✓

* Steps to carry out **Apriori Analysis**:-

→ Pseudo Code of the Algorithm is required  ($x \to y+z$)

→ Need a metric to Compare two or more Algo's running time.

→ Need some notation/symbol to represent the running time

Asymptotic notations

* Apriori Analysis can be carried out in 2 ways:

1. Step-count method → gives exact no. of operations (not much used)
2. Order of Magnitude method → gives approximate no. of operations ✓ (preferred) ✓

Assumption :- Every fundamental operation takes 1 unit of time

## ① Step-count Method :

```
Algo AJSir(n)
{
  1.  p = q + r                          ──→ 2 units
  2.  x = y * z                          ──→ 2 units
  3.  for (i=1; i<=n; i++)
      {  a = b + c
      }                                  ──→ (4n+2) units
  4.  for (j=1; j<=n; j++)
      {
        for (k=1; k<=n; k++)
        {
              x = y + z
        }
      }                                  ──→ (4n^2+4n+2) units
}
```

eg:

```
for (i=1; i<=n; i++)  ⟶ (2n+2)
{
    a=b+c  ⟶ (2*n)
}
```

Total operations

$\Rightarrow (2n+2) + 2*n$

$= \boxed{4n+2}$

① initialization $i=1$

② Condition $i<=n$

③ Updation $i=i+1$

for (i=1; i<=4; i++)

$\boxed{i=1 \longrightarrow 1}$

$i<=4 \longrightarrow$ True

$i++ \longrightarrow 2$

$1 \longrightarrow 2$

$2 \longrightarrow 2$

$3 \longrightarrow 2$

$4 \longrightarrow 2$

$5 \times \longrightarrow 1$

for (i=1; i<=n; i++)

$1 \longrightarrow$ initializatn

$(n+1) \longrightarrow$ comparison

$n \longrightarrow$ updation

$1+(n+1)+n$

$=(2n+2)$

runs n times ← for(j=1; j<=n; j++) ⟶ (2n+2)

{

run n times ← for(k=1; k<=n; k++) ⟶ (2n+2)

{

x = y+z ⟶ 2

}

}

Total operations

$(2n+2)$

$+$

$n * (2n+2)$

$+ n^2 * 2$

$\Rightarrow 2n^2+2n+2n+2+2n^2$

$\Rightarrow (4n^2+4n+2)$

Algo AJSir(n)
{

}

Total no. of operations

$$= 2+2+(4n+2)+(4n^2+4n+2)$$

$$= \underline{\underline{4n^2+8n+8}} \text{ units of time}$$

Approach 2 : Order of Magnitude ✓

(independent of n) → we just need to consider the

$\overset{\rightarrow 1}{\text{Constant}}$ ← $\boxed{\text{order}}$ of the number of

linear → n

Quadratic → $n^2$

⋮

fundamental operations in the

fundamental statement.

Algo AJSir2(n)

```
{
1. p = q + r          ⟶ 1 or 2 ⟶ Constant ⟶ 1
2. x = y * z          ⟶ 1 or 2 ⟶ Constant ⟶ 1
3. for (i=1; i<=n; i++)
   {
     a = b + c        ⟶ n units
   }
4. For (j=1; j<=n; j++)
   {
     for (k=1; k<=n; k++)
     {
       x = y + z      ⟶ n^2 units
     }
   }
}
```

Total units as per order of Magnitude approach

$\Rightarrow \quad 1 + n + n^2$

$\Rightarrow \quad \underline{\underline{(n^2 + n + 1)}}$ ✓

Notations

For Same Algo, The no. of units by Step-count mtd $\Longrightarrow$ $\underline{(4n^2 + 8n + 8)}$ ✓

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

Note :

↳ Aim of Apriori Analysis is to get / represent the running time of an algo as a mathematical function of input size 'n'.

eg:- $T(n) = 2n^2 + 2$

$T(n) = 4n^2 + 8n + 8$

$T(n) = 5n^3 + 2$

→ Order of magnitude refers to the rate of growth of time w.r.t 'n'.

* Rate of growth of Time as a <u>mathematical function</u>

**Logarithmic**

Form: $\log(n)$

eg: $\log_5(n)$, $\log_2(n)$

$\log(\log(n))$

**Polynomial**

$n^x$

$n^0$, $n^1$, $n^{0.5} \Rightarrow \sqrt{n}$, $n^2$

$n^3$ . . . . . .

Constant ($n^0$)

linear ($n^1$)

Sq root ($\sqrt{n}$)

quadratic ($n^2$)

cubic ($n^3$)

$\boxed{n * \log(n)}$

**Exponential**

$x^n$

eg: $3^n$, $2^n$, $7^n$ - - -

$n^n$, $n!$ . . .

**Note:** Rate of growth Comparison [in general]

Usually: [ Decreasing Func < Constant < logarithmic < Polynomial < Exponential ]


eg: $\left[ \left(\frac{1}{n}\right) < 10 < \log_2(n) < \sqrt{n} < n < n^2 \ldots < 2^n < 3^n \ldots n^n \right]$

* How to Compare the rate of growth of 2 functions?

→ larger input size (n)

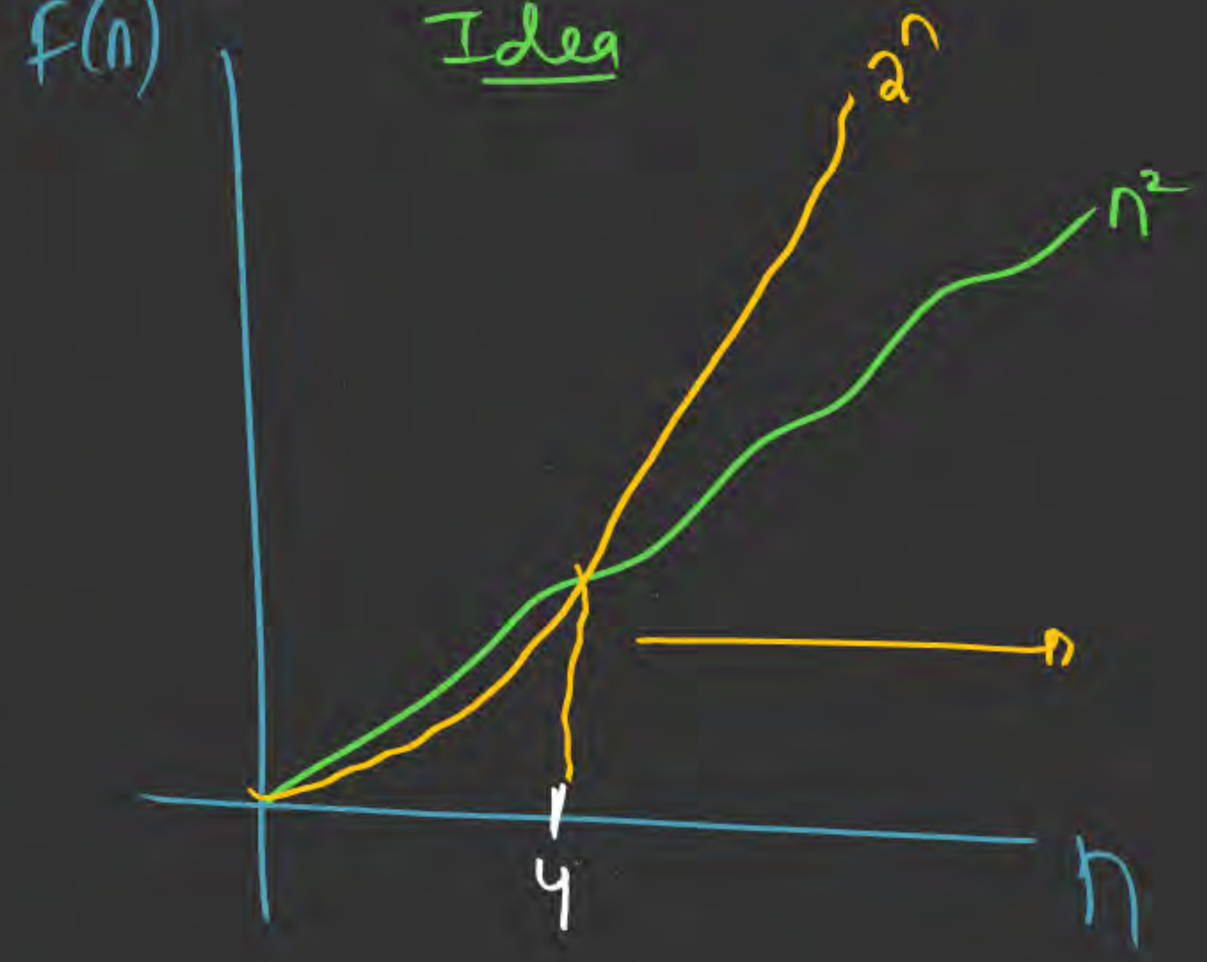
(Poly vs Expo)

eg:

| (Linear) $n$ | (expo) $2^n$ | $n^2$ (quadratic) |
|---|---|---|
| 1 | 2 ✓ | 1 |
| 2 | 4 = | 4 |
| 3 | 8 | 9 ✓ |
| 4 | 16 | 16 |
| 5 | 32 | 25 |
| 6 | 64 | 36 |
| 7 | 128 | 49 |

($2^n > n^2$ For larger values of $n$)

Idea



Algo1 ⟶ $n^2$ ⟶ lower Rate of growth ⟶ takes <u>less</u> time ✓ ⟶ <u>better</u>

Algo2 ⟶ $2^n$ ⟶ higher rate of growth ⟶ takes more time

<u>Note</u>: [Algorithms that takes Polynomial unit of time are <u>more</u> efficient than those with exponential time]

* Apriori Analysis :-

(input size)

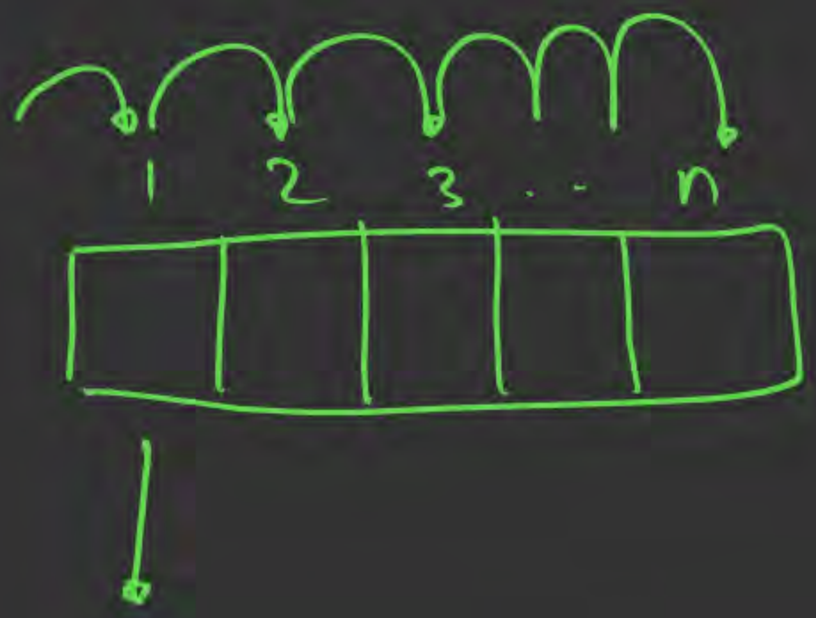
$(n^2+n+1)$ ← ① Time Complexity / Running Time as a function of 'n'.

Best Case (BC)
Avg Case (AC)
Worst Case (WC)

* ② To understand the change in nature of running time for a given algo and a fixed input size 'n', but for different input Classes. → (test cases)

## Example to understand point (2) :-

```
Algo LinearSearch( A, n, key)
{
    for (i=1; i<=n; i++)
    {
        if (A[i]== key)
            return i
    }
    print("Element Not Found")
}
```



eg: $n = 5$ (fixed)

Case 1: input class 1 → Key present at 1st positn (Best Case i/p)

A:

| 2 | 5 | 10 | 7 | 9 |
|---|---|---|---|---|

, Key = 2

→ Loop runs how many times?

⇒ 1 time → Constant

Best Case TC ⇒ TC ⇒ $O(1)$

Case 2: input class 2 → Key is not present or is present at last positn. (worst case i/p)

A:

| 3 | 9 | 15 | 7 | 5 |
|---|---|---|---|---|

, key = 4

→ Loop runs how many times?

⇒ 5 times ($n$ times) → linear Rate of growth.

worst case TC of linear Search ⇒ TC = $O(n)$

* Different Cases for Time Complexity for a fixed input size.
(Behaviors)

① **Best Case** → The i/p for which the algo runs $\boxed{\text{min}}$ no. of times
(does min work)

→ The Time Complexity for such i/p ⇒ Best Case TC.

② **Worst Case** → The i/p for which the algo runs $\boxed{\text{max}}$ no. of times
(does max work)

→ The time Complexity for such i/p ⇒ Worst Case TC

③ **Average Case**

→ deals with probability

Ex: Linear Search

BC TC → O(1)

WC TC → O(n)

AC TC → O(n) → Bonus

★ For any given Algo, in general:

Always:

$$\boxed{B(n) \leq A(n) \leq W(n)}$$

Best Case TC → B(n)
Worst Case TC → W(n)
Avg Case TC → A(n)

Few Examples:-

eg:-

① $B(n) < [A(n) = W(n)]$ ⇒ Linear Search, Binary Search.

② $[B(n) = A(n)] < W(n)$ ⇒ Quick Sort

③ $[B(n) = A(n) = W(n)]$ ⇒ Merge Sort, Heap Sort, Selection Sort.

Next lecture → [Asymptotic Notations] → V.V. Imp

HW → Revise both lectures ≈ (1-2hrs)

## Summary:

Apriori Analysis

- Step-count - mtd
- order of magnitude

→ BC, AC, WC

→ Types of functions

# THANK - YOU