CS & IT ENGINEERING

Algorithm

Analysis of Algorithms

Lecture No.- 03

By- Aditya sir

Recap of Previous Lecture

- **Topic** Need for analysis (How, why, what)
- **Topic** Methodology of Analysis
- **Topic** Aposteriori analysis (platform Dependent)
- **Topic** Apriori Analysis

Platform Independent

Topics to be Covered

Topic Types of Analysis

Topic Intro to Best, Worst and Average Case

Topic : Analysis of Algorithms

Approach : $\boxed{\text{Step-Count Method}}$

Algorithm Test

{

Constant

2

1 $\boxed{x = y + z}$

2 for $i : 1 \longrightarrow n$.

$\boxed{x = y + z}$

$\longrightarrow 1 + (n+1) + n + n + n \longrightarrow (4n+2)$

3 for $i : 1 \longrightarrow n \longrightarrow 1 + (n+1) + n$

for $j : 1 \longrightarrow n \longrightarrow n(1 + (n+1) + n)$

$\boxed{x = y + z} \longrightarrow (n+n)n$

$= (2n+2) + n(2n+2) + n(2n)$

$(2n+2) + 2n^2 + 2n + 2n^2$

$= 4n^2 + 4n + 2$

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

$= \boxed{4n^2 + 8n + 6}$

Approach 2: Order of Magnitude

→ To determine the time of Algorithm under $\boxed{\text{Apriori}}$ Analysis.

→ Order of Magnitude of a statement/step of the Algorithm refers to the frequency/count of the fundamental operation in that step.

Algorithm Test

{

1. $x = y + z$ → 1

2. for $i: 1 \longrightarrow n$

$x = y + z$ → (n)

3. for $i: 1 \longrightarrow n$

for $j: 1 \longrightarrow n$

$x = y + z$ → n^2

$4n^2 + 8n + 6$ → Step count Method

$O(n^2)$

Total Time By Order of Magnitude approach

$= (1 + n + n^2)$

$= (n^2 + n + 1)$

$O(n^2)$

eg2 :-

```
Algo Sum(a, b, c)
{
    int a, b, c
    read (a, b)        → 1
  if  a < b:           → 1
    { c = a + b        → 1
    }
  else
    { c = a * 1        → 1
    }
    Print(c)           → 1
}
```

$1 + 1 = 2$

$$\text{Time} = 1 + 2 + 1$$
$$= 4$$
$$= O(1) \rightarrow \text{Constant}$$

Topic : Analysis of Algorithms

```
Algo Sum (A, n):
{
    int n, A[n];          → 1
    int sum = 0, i;       → 1
    for i: 1 → n
    {
        sum = sum + A[i]  → n
    }
    print(sum)            → 1
}
```

$\text{Time} = \underbrace{1+1}_{} + n + 1$

$= n+3$

$\boxed{O(n)}$

→ The basic objective of Apriori Analysis is to represent/(obtain) the Time Complexity (Running Time) of the Algorithm by means of a mathematical function w.r.t the input size. (usually 'n')

$n \longrightarrow$ input size

1) $\underbrace{T(n)}$: $\underline{n^2+n+1}$ $\qquad \underline{(4n^2+8n+6)}$

2) $\underbrace{T(n)} = 4 \longrightarrow$ Constant

3) $\underbrace{T(n)} = n+3$

$n+3$

$n+5$

The rate of growth of time w.r.t 'n'

Time $\propto$ funtion (w.r.t input size 'n')

Polynomial

eg:- $n, n^2, n^3, n^4 \ldots\ldots$

- n → linear
- n^2 → Quadratic
- n^3 → Cubic

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

logarithmic

eg:- $\log(n)$

$\log(\log(n))$

Exponential

→ a^n $(a > 1)$

eg:- $2^n, 3^n, n^n \ldots n!$

	(T1)	(T2)
	A1	A2
n	n^2	2^n
2	4	4
3	9	8
4	16	16
5	25	32
6	36	64
7	49	128

units

Note :-

① Exponential functions have a higher rate of growth (takes more time)

② Polynomial functions have a lower rate of growth (takes less time)

The objective is always to develop Algorithms having Polynomial time Complexity (more efficient)

→ To Solve a Problem

100M

↓

200M

↓

1B

Apriori Analysis

Types of Analysis

① To determine the running time w.r.t the increasing input size (n)

② To observe the behaviour of the Algorithm for a fixed input of size 'n'.

(diff input class)

Algo LinearSearch(A, n, x):

{

int i

for i : 1 → n

{

if (A[i] == x)

{

Print(i)

exit

}

}

Print ("elem not present")

}

Time ∝ Comparison

① Best Case $n=5$

$\Rightarrow O(1)$

Const

② Worst Case $n=5$

$O(n)$

$n=5$

$A = [a_1, a_2, a_3, a_4, a_5]$

eg1: I1: Increasing:

$n=5$

$x : 3$

$A = [3, 4, 7, 9, 15]$

$x=3$

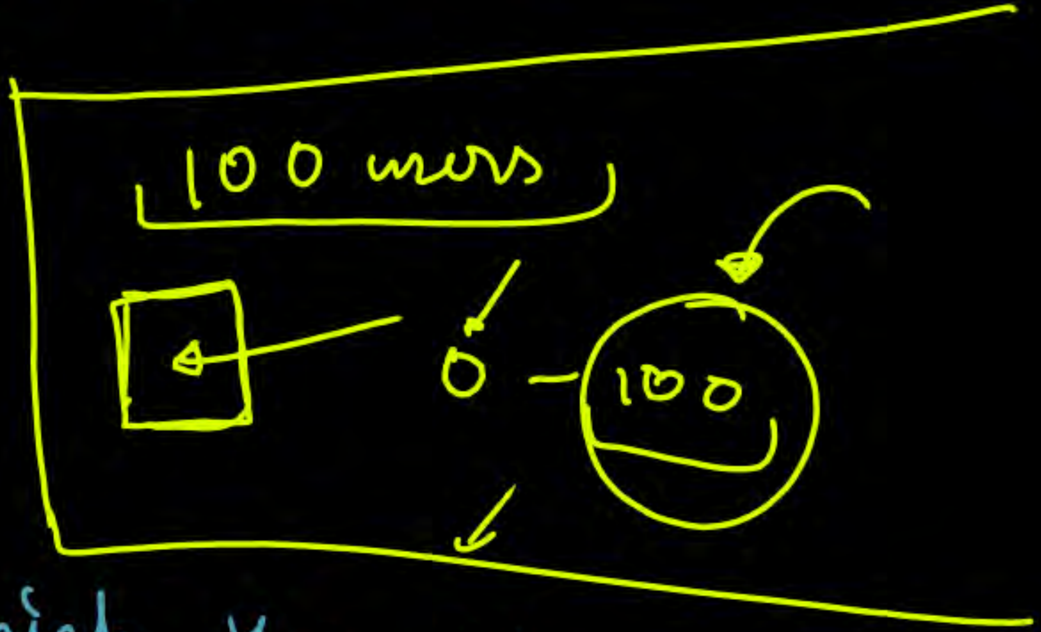
eg2: I2: Decreasing

$A = [15, 9, 7, 4, 3]$

I3: Random

$A = [4, 9, 7, 3, 15]$

① Best Case: The input class for which the Algo takes min amount of time is called the Best Case Time.



② Worst Case: The input class for which the Algo , the max amount of time is called the Worst Case time.

interested in this

③ Average Case : it is determined in 3 steps:

a) Enumerate all ip's ($ip_1 \ldots ip_k$)

b) Determine the time for each of these ip's ($t_1 \to ip_1$, $t_2 \to ip_2$, $t_3 \to ip_3$ ....)

c) Associate the Prob P_i with each i/p class.

$$\boxed{A(n) = \sum_{i=1}^{k} P_i * t_i}$$

eg:- Linear Search :

1. Best Case: $1 \longrightarrow O(1)$
2. Worst Case: $n \longrightarrow O(n)$

3. Avg Case (for a successful linear search)

No of Comparisons: $1+2+3+\dots+n = \dfrac{\cancel{n}(n+1)}{\dfrac{2}{\cancel{n}}} = \dfrac{(n+1)}{2} = O(n)$

(1) Best Case time : $B(n)$
(2) Worst Case time : $W(n)$
(3) Avg Case time : $A(n)$

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

(1) $B(n) = A(n) = W(n)$ $\longrightarrow$ (1) Merge Sort, (2) Selection Sort

(2) $B(n) < (A(n) = W(n))$ $\longrightarrow$ (1) Linear Search

(3) $(B(n) = A(n)) < W(n)$ $\longrightarrow$ (1) Quick Sort

(4) $\boxed{B(n) < A(n) < W(n)}$

Topic : Analysis of Algorithms

Linear search A(n)

1. Best Case : 1 : O(1)
2. Worst case : n : O(n)
3. Average case: (for a successful linear search)

Topic : Analysis of Algorithms

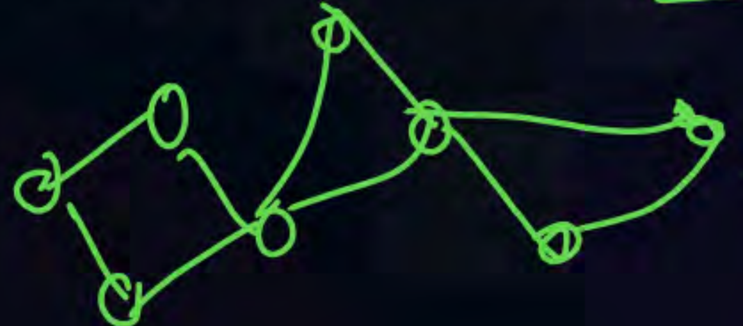
Analyzing algorithms involves thinking about how their resource requirements-the amount of time and space they use-will scale with increasing input size.

1) **Proposed Definition of Efficiency (1):** An algorithm is efficient if, when implemented, it runs quickly on real input instances. (Platform Dependent)

2) **Proposed Definition of Efficiency (2):** An algorithm is efficient if it achieves qualitatively better worst-case performance, at an analytical level, than brute-force search.

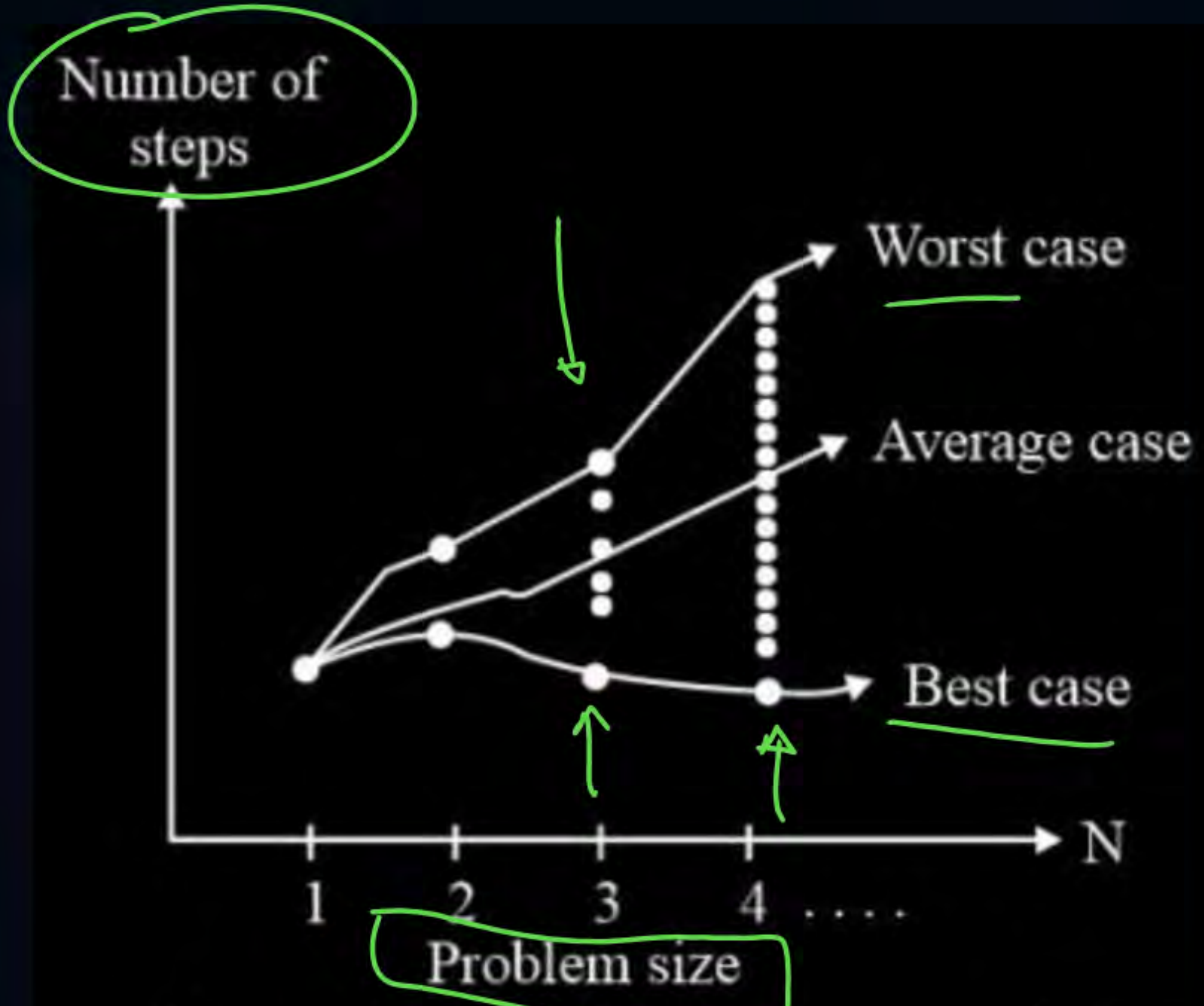
3) **Proposed Definition of Efficiency (3):** An algorithm is efficient if it has a polynomial running time.

(Brute force) {



}

Topic : Analysis of Algorithms



Best, worst, and average-case complexity

Topic : Analysis of Algorithms

- The Worst-case Complexity of the algorithm is the function defined by the maximum number of step taken in any instance of size n. This represents the curve passing through the highest point in each column.

- The Best-case complexity of the algorithm is the function defined by the minimum number of steps taken in any instance of size n. This represents the curve passing through the lowest point of each column.

- The average-case complexity of the algorithm, which is the function defined by the average number of steps over all instances of size n.

THANK - YOU