

# Runtime and Space Complexity

Course on C-Programming & Data Structures: GATE - 2024 & 2025

# Data Structure: Introduction

By: Vishvadeep Gothi

# Data Structure

Chapter Number	Chapter Name
1	Data Structure Basics
2	Arrays
3	Linked List
4	Queue & Stack
5	Tree
6	Graph
7	Hashing

# Data Structure

- ◆ Mathematical and logical model of organizing the interrelated data.

# Types of Data Structure

- ◆ **Types:**

- 1. Linear:** Elements are arranged in linear (sequential) fashion.
- 2. Non-Linear:** Elements are arranged in non-linear fashion.

# Classification of Data Structure



# Operations on Data Structures

# Operations on Data Structures

# Operations on Data Structures

# Algorithm

is a step-by-step procedure

for solving a problem

or performing a task

in a finite number of steps

using a well-defined set of rules

and operations

that can be repeated

until a desired result is obtained

or a specific condition is met

or a limit is reached

or a certain number of iterations have occurred

or a certain amount of time has passed

or a certain amount of memory has been used

or a certain level of accuracy has been achieved

or a certain number of errors have occurred

or a certain number of warnings have been issued

or a certain number of messages have been sent

or a certain number of notifications have been delivered

or a certain number of tasks have been completed

or a certain number of units have been processed

or a certain number of items have been handled

or a certain number of requests have been fulfilled

or a certain number of users have been served

or a certain number of transactions have been processed

or a certain number of files have been processed

or a certain number of lines of code have been executed

# Analysis of Algorithm

Algorithm analysis is the process of determining the computational complexity of an algorithm. It involves analyzing the time and space requirements of the algorithm as a function of the input size. The goal is to understand how the performance of the algorithm scales with the size of the input and to identify efficient ways to implement the algorithm.

There are two main types of analysis:

- Asymptotic Analysis:** This type of analysis focuses on the behavior of the algorithm as the input size grows to infinity. It uses Big O notation to describe the upper bound of the running time or space requirements. For example, if an algorithm has a time complexity of  $O(n^2)$ , it means that the running time increases quadratically with the input size  $n$ .
- Empirical Analysis:** This type of analysis involves running the algorithm on various inputs and measuring its actual performance. It can be used to verify the results of asymptotic analysis and to identify any hidden constants or lower bounds in the algorithm's performance.

Common techniques used in algorithm analysis include:

- Recurrence Relations:** These are equations that describe the running time of an algorithm in terms of smaller subproblems. They are often solved using methods like substitution, recursion trees, or the master theorem.
- Loop Invariants:** These are properties that remain true throughout the execution of a loop. They are used to prove the correctness of the algorithm and to analyze its time complexity.
- Divide-and-Conquer:** This is a general problem-solving technique that involves breaking a problem into smaller subproblems, solving them independently, and then combining their solutions to get the final result. It is often used in algorithms like merge sort and quicksort.

Efficiency is a key consideration in algorithm analysis. An efficient algorithm is one that performs well even for large input sizes. It is often compared against other algorithms for the same problem to determine which is more efficient. Efficiency is typically measured in terms of time complexity, but space complexity is also important in many applications.

# Analysis of Algorithm

- ◆ **Space Complexity**
- ◆ **Run-Time Complexity**

→ no. of spaces (no. of element storage)  
needed by algo.

Time needed to  
run an algo

$$\text{Efficiency} := \frac{A_1}{t_1} \quad \frac{A_2}{t_2}$$

$t_1 < t_2$   
 $A_1$  is efficient

## How to calculate time

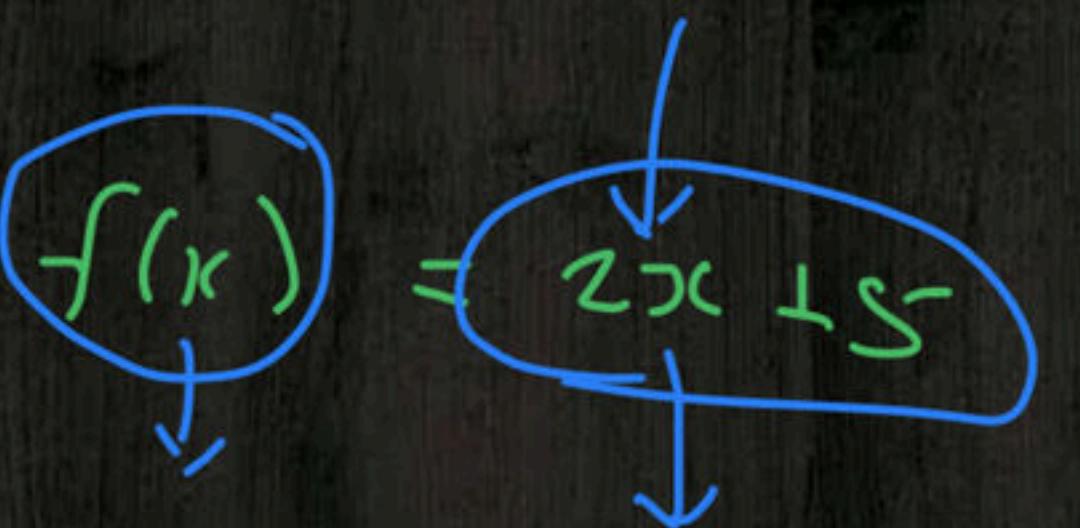
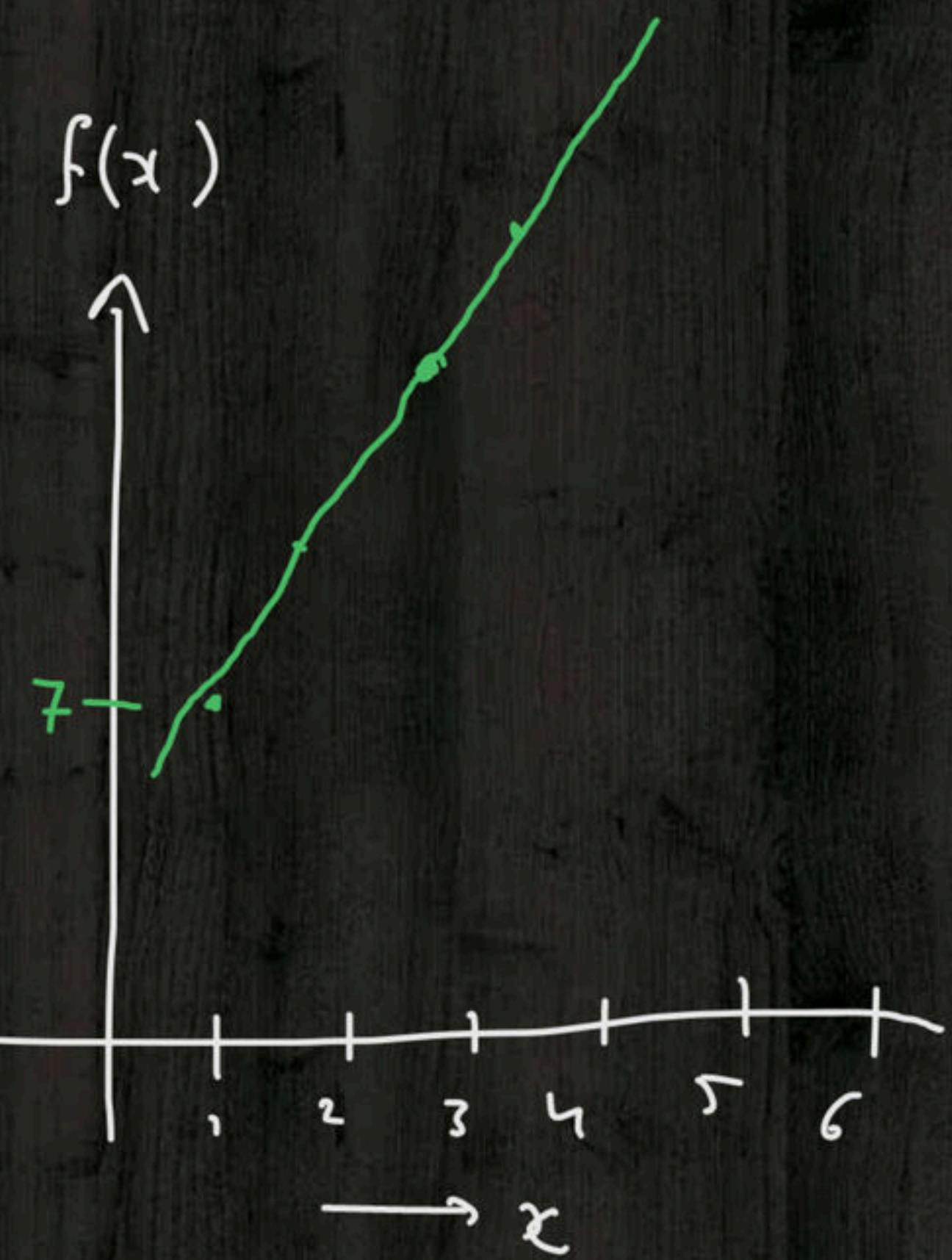
- on wall clock
- ↓
- ① implement algo in a prog. language
- ② Run program on a system & measure time to run.

↓

Time is dependent on so many external factors  $\Rightarrow$  no any consistency in result

- ↙ based on no. of operations or steps performed
- ↓
- based on rate of growth

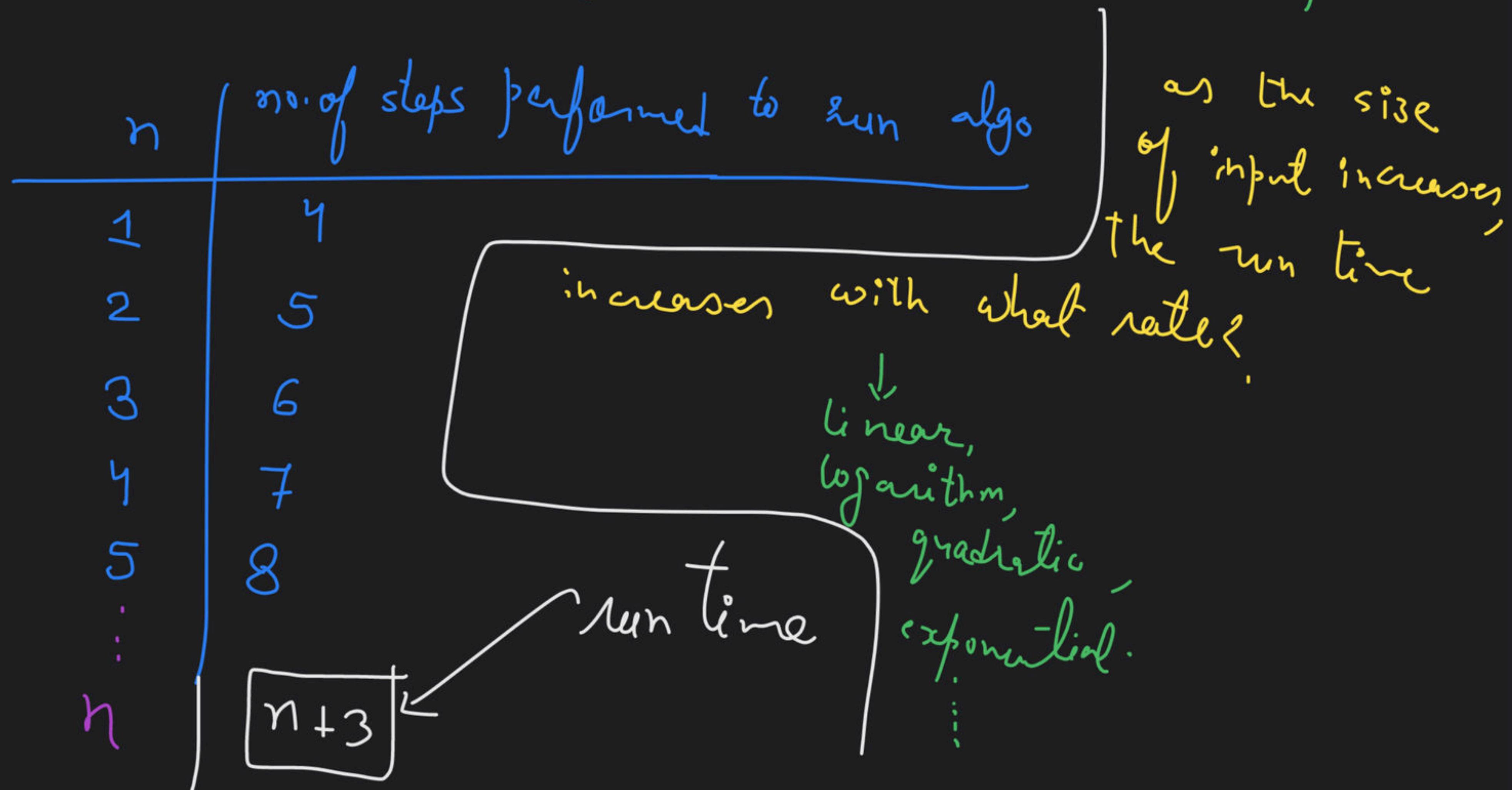
# Analysis of Algorithm



$x = 1$	,	$f(x) = 7$
$x = 2$	,	$f(x) = 9$
$x = 3$	,	$f(x) = 11$
$x = 4$	,	$f(x) = 13$
$x = 5$	,	$f(x) = 15$

Rate of growth :-

An algo takes  $n$  numbers as input



If rate of growth

complexity

constant

1

logarithmic

$\log n$

linear

$n$

linear logarithmic

$n \log n$

quadratic

$n^2$

quadratic log.

$n^2 \log n$

cubic

$n^3$

;

Exponential

$2^n$

example:-

Array of  $n$  elements

Algo:-

Total no. of steps

$$= 1 + n + 1 + n$$

$$= 2n + 2$$

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

① take a variable sum , with initial value zero → ①

② Run a loop for  $n$  times with  $i$  ranging from 0 to  $n-1$   
for( $i=0; i < n; i++$ )  
{

    sum = sum + array element at index  $i$

$$\text{sum} = \text{sum} + A[i]$$

    }  $\underbrace{n}$

Ques:-

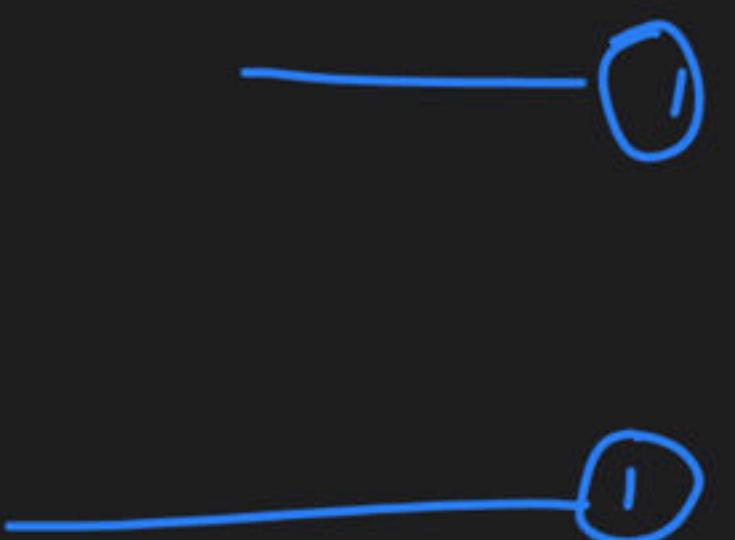
Input  $\Rightarrow$  array of size  $n$   
 $A[ ]$ , starting index  $\Rightarrow 0$   
last index  $\Rightarrow n-1$

algo:-

$$\text{result} = (A[0] + A[n-1]) / 2$$

return result

Run time = 2



$n$	$T(n)$
5	2
6	2
100	2
⋮	⋮
100000	2

Assume:-

There are 2 algos for same problem

A<sub>1</sub>



A<sub>2</sub>

Run Time  $\Rightarrow n \log n$        $n^2$

A<sub>1</sub> is efficient

# Example: 1

for(  $i = 1$ ;  $i \leq n$ ;  $i++$ )

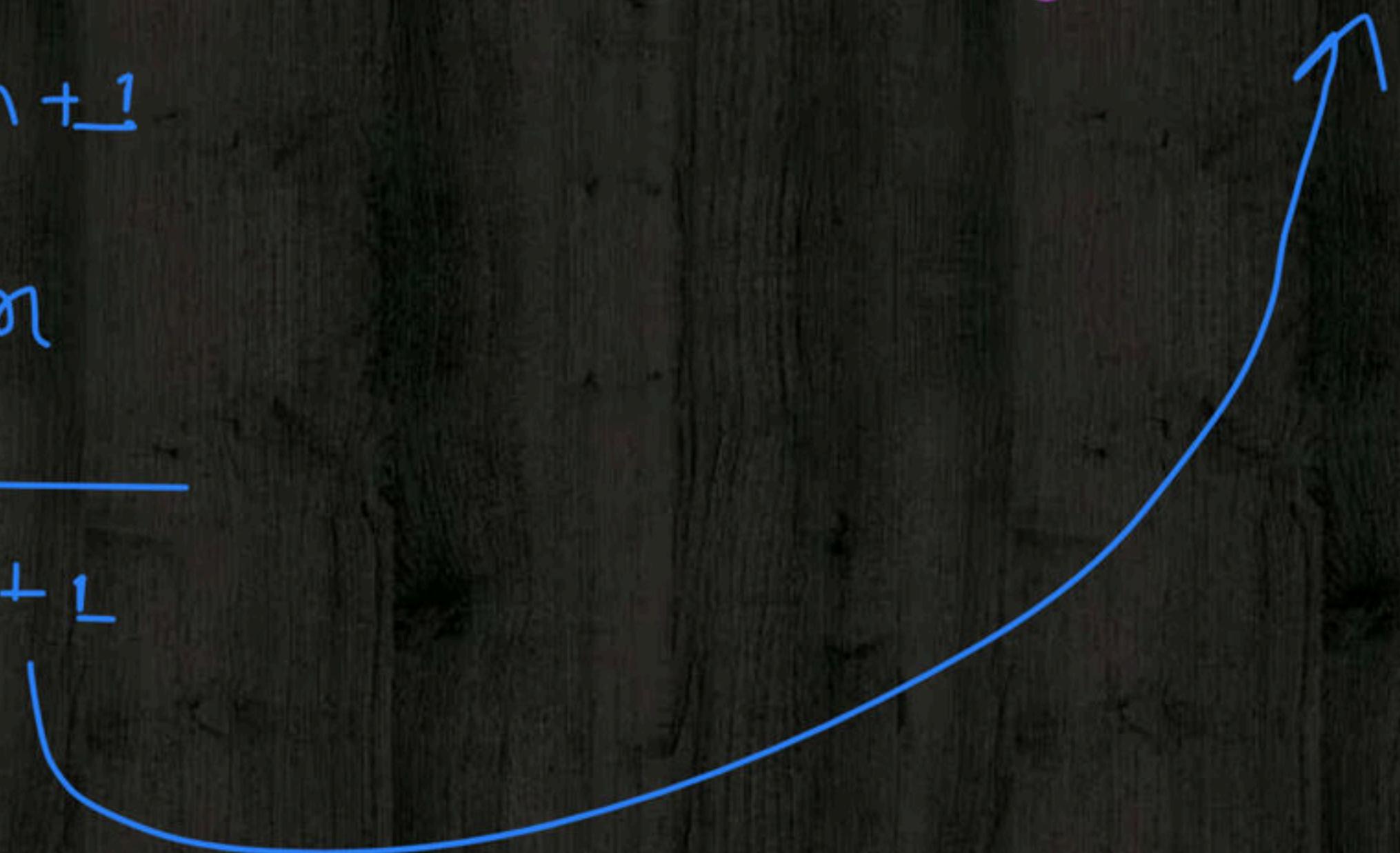
{

$x = A[i] + y;$   $\longrightarrow n$

$y$

Total  $\Rightarrow 2n + 1$

R.T. Complexity  $\Rightarrow$  linear



## Example: 2

```
for(i = 0; i < n; i++) ————— n + 1 - n + 1  
{
```

```
  for(j = 0; j < n; j++) ————— n * (n + 1) = n2 + n  
{
```

```
    x = A[i][j] + y[j] ————— n * n = n2  
}
```

```
}
```

Total  $\Rightarrow 2n^2 + 2n + 1 \Rightarrow$  R.T.  
Complexity  $\Rightarrow$  Quadratic

## Example: 3

```
for(i = 0; i < n; i++) {  
    for(j = 0; j < n; j++) {  
        for(k = 0; k < n; k++) {  
            y = A[i][j][k] + y;  
        }  
    }  
}
```

R-T. Complexity =  $n^3$

```

for (i = 0; i < n; i++)
{
    for (k = 0; k < n; k++)
    {
        x = A[i][k] + y;
    }
}

for (j = 0; j < n; j++)
{
    q = A[i][j] + y;
}

```

$n^2$

$n^2$

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

R.T. Complexity = Quadratic

```

for (i = 0; i < n/2; i++) ----->  $\frac{n}{2} + 1$ 
{
    for (k = 0; k < n; k++) ----->  $\frac{n}{2} * (n + 1)$ 
    {
        x = y + A[i][k]; ----->  $\frac{n}{2} * n$ 
    }
}

```

$$\frac{n}{2} * n = \frac{n^2}{2} \Rightarrow \begin{array}{l} \text{R.T.} \\ \downarrow \text{Complexity} \\ \text{Quadratic} \end{array}$$

```
for (i = 1; i <= h; i = i + 2) | for (i = n; i >= 1; i = i / 2)  
{  
}  
}  
.
```

if  $n = 50$

$i = 1$   
 $= 2$   
 $= 4$   
 $= 8$   
 $= 16$   
 $= 32$

$R - \text{Time Complexity} = \log_2 n$

$$x = \log_2 n$$



$$2^x = n$$

# Asymptotic Notation

# Asymptotic Notation

# Question 1

Consider an algorithm which takes  $n$  number of inputs and performs an operation on it, which requires  $n-1$  operations. The best possible run time complexity for the algorithm can be represented as:

- (A)  $O(n)$
- (B)  $\Theta(n)$
- (C)  $O(n \log_2 n)$
- (D) A & B both

# Question 2

Consider an algorithm which takes  $n$  number of inputs and performs an operation on it. The operation is performed by algorithm in such a way that it is not dependent on number of inputs. Which of the following can be the run time complexity for the algorithm?

- (A)  $O(1)$
- (B)  $\Theta(1)$
- (C)  $O(n)$
- (D) All

---

# Happy Learning



---