

**Time Complexity**

CPU Operations

Input Size

Time Complexity

$O(n!)$   $O(2^n)$   $O(x^3)$   $O(x^2)$   $O(n \log n)$   $O(n)$   $O(1)$

**Time Complexity**

algorithms365 Performance with purpose

Bookmark icon, Share icon, More options icon

A man with a beard and glasses, wearing a blue polo shirt with the "algorithms365" logo, is speaking into a microphone while gesturing with his hands. He is standing in front of a presentation slide. The slide has a dark blue background with a large yellow "Time Complexity" title. To the left of the title is a graph showing CPU operations versus input size. The graph features several curves representing different time complexities:  $O(n!)$  (red, very steep),  $O(2^n)$  (orange, second steepest),  $O(x^3)$  (yellow),  $O(x^2)$  (light orange),  $O(n \log n)$  (green),  $O(n)$  (dark green), and  $O(1)$  (light blue, horizontal). The x-axis is labeled "Input Size" and the y-axis is labeled "CPU Operations". The top right corner of the slide shows navigation icons for a presentation slide.

## Time Complexity.

- \* Time Complexity is a way to describe how long an algorithm (a step-by-step method to solve a problem) takes to run, depending on the size of the input it's given.

### 1. Identify the Basic Operation's

Determine the most significant operations in the algorithm. These are operations that most affect the algorithm's runtime, such as comparisons, assignments, and Arithmetic operations.

### 2. Count the Basic operation.

Count how many times these basic operations are executed as a function of the input size  $n$ .

This involves analysis loops, recursive calls, and any other constructs that affect the runtime.

### 3 Express the count as a function

Express the total no. of operations as a mathematical function of  $n$ , where  $n$  is the size of input

### 4 Simplify the function

Simplify the function by keeping only the dominant term, which is the term that grows the fastest as  $n$  increases. Discard constants and less significant terms because they have a negligible effect on the growth rate for large  $n$ .

### 5 Use Big O Notation

Express the simplified function using Big O notation to describe the time complexity. Big O notation

Provides an upper bound on the growth rate, focusing on the dominant term

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_

Constant  $O(1)$

Index	Value
0	1
1	2
2	3
3	4
4	5
5	6

numbers [5]

here time is constant

Logarithmic  $O(\log n)$  → data half / compare

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

Binary search key = 9 ↴ sorted

Element end → worst case scenario

1	2	3	4	5	6	7	8	9	10
X	X	X	X	X	X	X	X	✓	

$n$  |  $n/2$  |  $n/4$  |  $100 \rightarrow 10$

Linear  $O(n)$

23	48	523	65	150	72	93
					↑	

Search = 150

## Quadratic $O(n^2)$

#	ID
1	123
2	765
3	999
4	500

#	ID
1	459
2	227
3	234
4	567

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

find the common element in these two arrays

$$n \times n = n^2$$

## Cubic $O(n^3)$

#	ID	#	ID	#	ID
1	123	1	643	1	56
2	765	2	344	2	765
3	999	3	123	3	45
4	500	4	456	4	67

$$n \times n \times n = n^3$$

Exponential  $O(2^n)$  → takes more time

1	2	3	4	5	6	7	8	9	10

All possible subset

Factorial  $O(n!)$  → takes more time than Exponen-  
-tial

Generates all possible seating arrangements for a list  
of friends' IDs

1	2	3	4	5	6	7	8	9	10

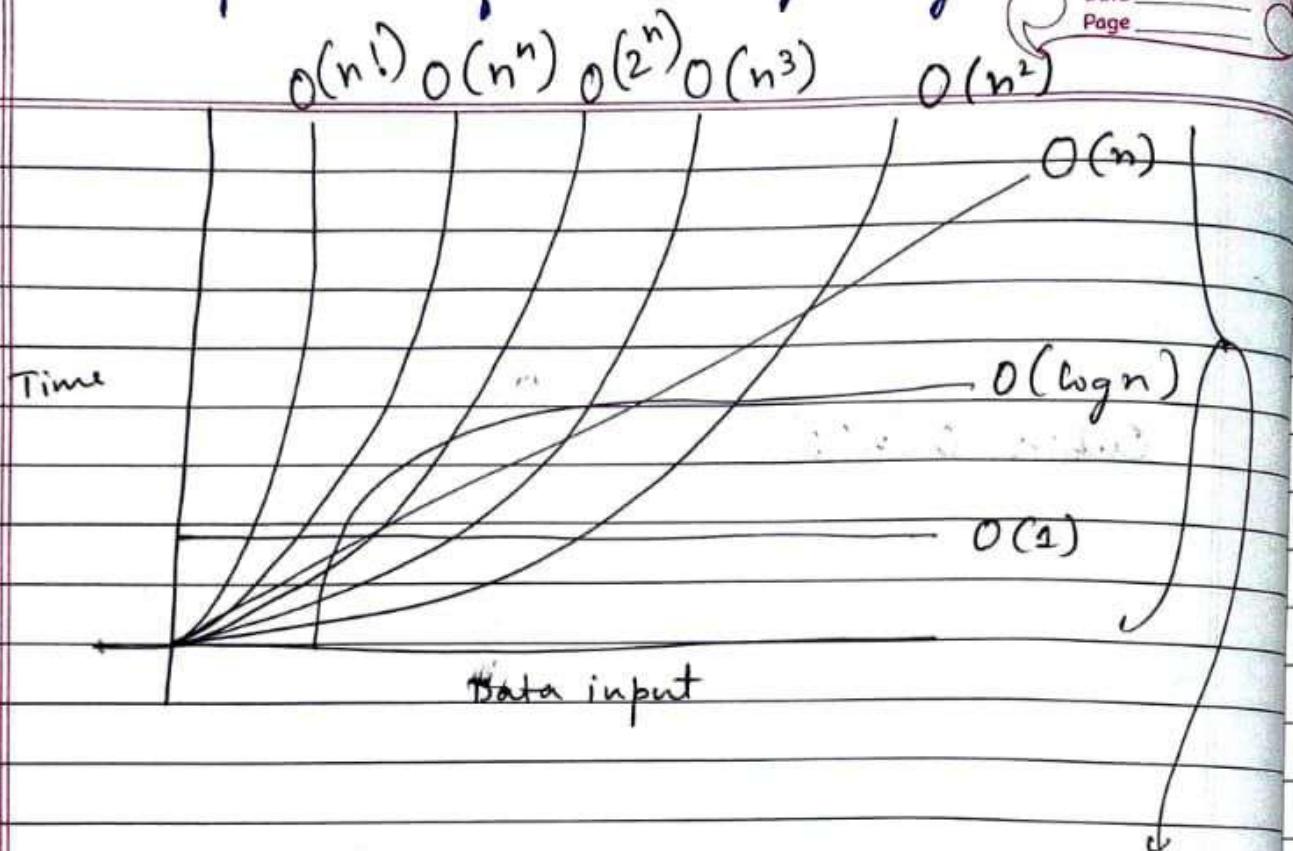
123  
231  
132

## Comparison of time complexity

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_



These 3 are the  
most used complexity  
in every programming