Time Complexity

DEFINITION

- Time complexity is defined as the amount of time taken by an algorithm to run, as a function of the length of the input.
- It measures the time taken to execute each statement of code in an algorithm.
- It is not going to examine the total execution time of an algorithm.

DIFFERENT SCENARIOS

- **Categorized into two cases:**

1. **Best-case**
2. **Worst-case**

BEST-CASE TIME COMPLEXITY (OMEGA)

- The best-case time complexity represents the minimum amount of time an algorithm will take to complete when given the input in the most favorable conditions.
- It provides an idea of how efficient an algorithm can be when everything goes perfectly.
- In many cases, the best-case time complexity is expressed using the Omega (Ω) notation.
- Omega notation represents the lower bound on the time complexity of an algorithm.
- It describes the best-case scenario, or the minimum amount of time an algorithm will take as a function of the input size.

EXAMPLE :OMEGA NOTATION

- **Linear Search in a Sorted List :**
  - Algorithm: Searching for a specific element in a sorted list by iterating through it sequentially.
  - Best-Case Scenario: The element you're looking for is at the beginning of the list.
  - Best-Case Time Complexity: O(1) (constant time), as you find the element with a single comparison.

WORST-CASE TIME COMPLEXITY (BIG O)

- The worst-case time complexity represents the maximum amount of time an algorithm will take to complete when given the input in the most unfavorable conditions.
- It describes the upper bound on the running time and is often used to ensure that the algorithm doesn't perform poorly in any input scenario.
- The Big O (O) notation is commonly used to express the worst-case time complexity.
- It describes the maximum amount of time an algorithm will take as a function of the input size.

EXAMPLE :BIG O NOTATION

- **Binary Search in a Sorted List**
  - Algorithm: Searching for a specific element in a sorted list by repeatedly dividing the search space in half.
  - Worst-Case Scenario: The element is at the end of the list, or it's not in the list.
  - Worst-Case Time Complexity: O(log n) (logarithmic time), as the search space is halved in each step.

BEST-CASE AND WORST-CASE TIME COMPLEXITY (THETA)

- **Theta notation** represents both an upper and a lower bound on the time complexity of an algorithm.
- It describes the best-case and worst-case scenarios when they are the same or very similar.
- In other words, if an algorithm has a time complexity of **$\Theta(f(n))$**

TIME COMPLEXITY NOTATION

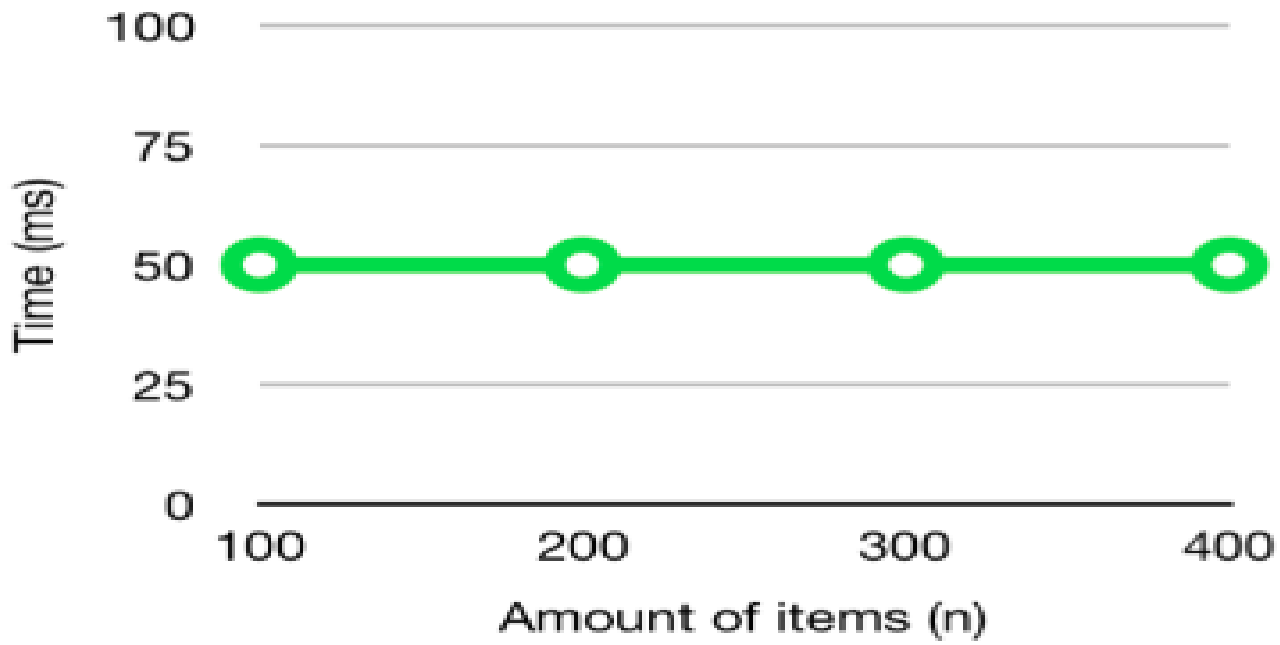
- There exists a relation between the input data size (n) and the number of operations performed (N) with respect to time.
- This relation is denoted as Order of growth in Time complexity and given notation O[n] where O is the order of growth and n is the length of the input.
- It is also called as 'Big O Notation'.
- Big-O notation, sometimes called "asymptotic notation", is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity.

TIME COMPLEXITY NOTATION

- **Constant time – O (1)**
- **Linear time – O (n)**
- **Logarithmic time – O (log n)**
- **Quadratic time – O (n^2)**
- **Cubic time – O (n^3)**

- An algorithm is said to have constant time with order O (1) when it is not dependent on the input size n.
- Irrespective of the input size n, the runtime will always be the same.

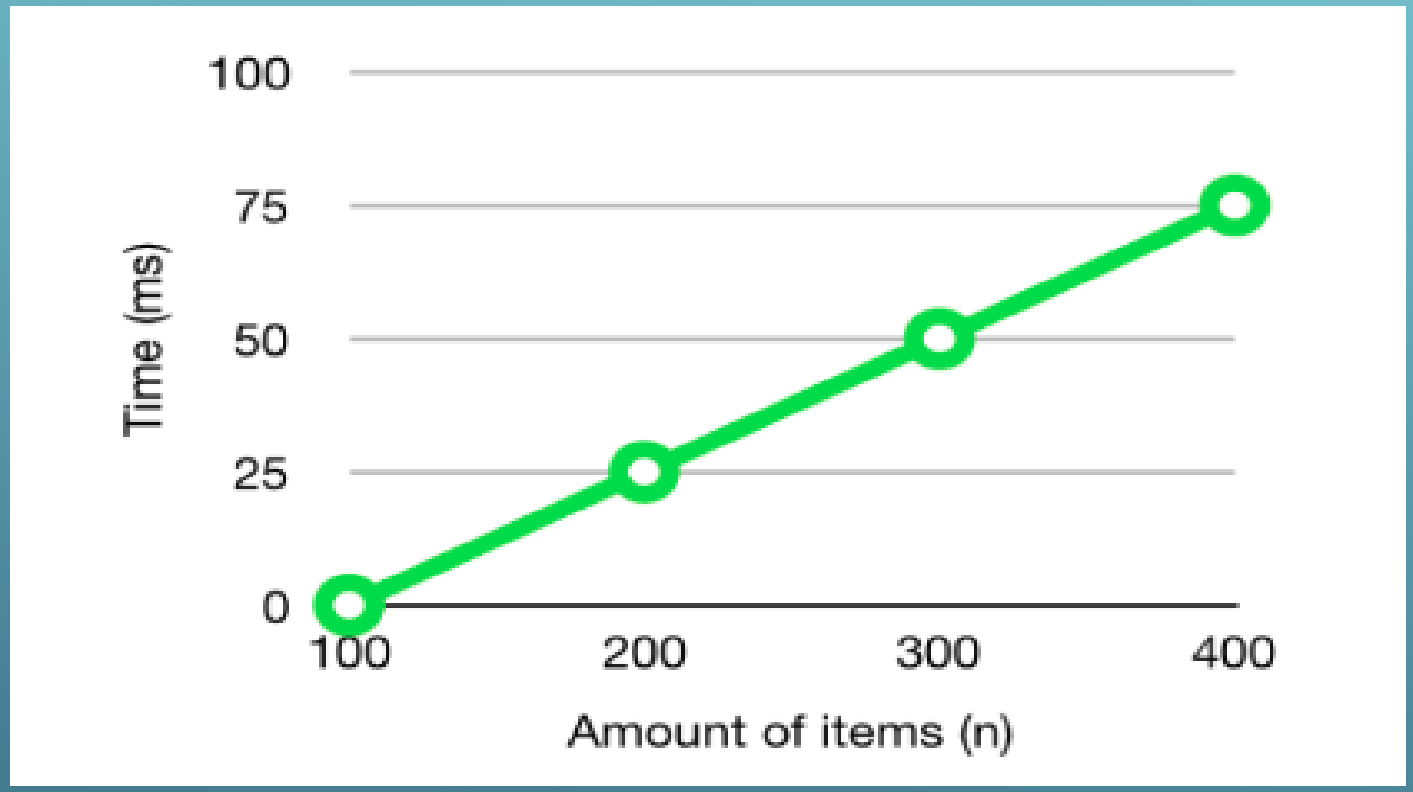
Constant time – O (1)



GRAPH INPUT WITH TIME

- An algorithm is said to have a linear time complexity when the running time increases linearly with the length of the input.
- When the function involves checking all the values in input data, with this order O(n).

Linear time – O (n)



GRAPH INPUT WITH TIME

- An algorithm is said to have a logarithmic time complexity when it reduces the size of the input data in each step.
- This indicates that the number of operations is not the same as the input size.
- The number of operations gets reduced as the input size increases.

Logarithmic time – O (log n)

- An algorithm is said to have a non-linear time complexity where the running time increases non-linearly (n^2) with the length of the input.
- Generally, nested loops come under this order where one loop takes O(n) and if the function involves a loop within a loop, then it goes for O(n)*O(n) = O(n^2) order.
- Similarly, if there are 'm' loops defined in the function, then the order is given by O (n ^ m), which are called polynomial time complexity functions.

Quadratic time – O (n^2)

- An algorithm is said to run in cubic time if the running time of the three loops is proportional to the cube of N. When N triples, the running time increases by N * N * N

Cubic time – O (n^3)

- Quasilinear time complexity, denoted as O(n log n), indicates that the time required to perform an operation or execute a function grows at a rate proportional to n times the logarithm of n.
- This time complexity often arises in algorithms that efficiently process or sort data.

Quasilinear Time – O(n log n)

- An algorithm is said to have an exponential time complexity when the growth doubles with each addition to the input data set.

Exponential Time – O(2^n)

Time complexity:

	Worst Case Scenario	Average Case Scenario	Best Case Scenario
Delete (Stack)	O(1)	O(1)	O(1)
Insert (Stack)	O(1)	O(1)	O(1)
Search (Stack)	O(n)	O(n)	O(1)
Peek/Top (Stack)	O(1)	O(1)	O(1)
Delete (Queue)	O(1)	O(1)	O(1)
Insert (Queue)	O(1)	O(1)	O(1)
Search (Queue)	O(n)	O(n)	O(1)