Time Complexity
IT Batch 400

What is time complexity?

To recap **time complexity** estimates how an algorithm performs regardless of the kind of machine it runs on. You can get the time complexity by "counting" the number of operations performed by your code. This time complexity is defined as a function of the input size n using Big-O notation. n indicates the input size, while O is the worst-case scenario growth rate function.

We use the Big-O notation to classify algorithms based on their running time or space (memory used) as the input grows. The O function is the growth rate in function of the input size n.

Big O Notation	**Name**	**Example(s)**
O(1)	Constant	# Odd or Even number, # Look-up table (on average)
O(log n)	Logarithmic	# Finding element on sorted array with **binary search**
O(n)	Linear	# Find max element in unsorted array, # Duplicate elements in array with Hash Map
O(n log n)	Linearithmic	# Sorting elements in array with **merge sort**
O(n^2)	Quadratic	# Duplicate elements in array **(naïve)**, # Sorting array with **bubble sort**
O(n^3)	Cubic	# 3 variables equation solver
O(2^n)	Exponential	# Find all subsets
O(n!)	Factorial	# Find all permutations of a given set/string

O(1) - Constant time

O(1) describes algorithms that take the same amount of time to compute regardless of the input size. For instance, if a function takes the same time to process ten elements and 1 million items, then we say that it has a constant growth rate or O(1). Let's see some cases.

Examples of constant runtime algorithms:

- Find if a number is even or odd.
- Check if an item on an array is null.
- Print the first element from a list.
- Find a value on a map.

O(n) - Linear time

Linear running time algorithms are widespread. These algorithms imply that the program visits every element from the input.

Linear time complexity *O(n)* means that the algorithms take proportionally longer to complete as the input grows.

Examples of linear time algorithms:

- Get the max/min value in an array.
- Find a given element in a collection.
- Print all the values in a list.

O(n^2) - Quadratic time

A function with a quadratic time complexity has a growth rate of n^2. If the input is size 2, it will do four operations. If the input is size 8, it will take 64, and so on.

Here are some **examples of quadratic algorithms**:

- Check if a collection has duplicated values.
- Sorting items in a collection using bubble sort, insertion sort, or selection sort.
- Find all possible ordered pairs in an array.

O(n^c) - Polynomial time

Polynomial running is represented as $O(n^c)$, when $c > 1$. As you already saw, two inner loops almost translate to $O(n^2)$ since it has to go through the array twice in most cases. Are three nested loops cubic? If each one visit all elements, then yes!

Usually, we want to stay away from polynomial running times (quadratic, cubic, n^c, etc.) since they take longer to compute as the input grows fast. However, they are not the worst.

O(log n) - Logarithmic time

Logarithmic time complexities usually apply to algorithms that divide problems in half every time. For instance, let's say that we want to look for a book in a dictionary. As you know, this book has every word sorted alphabetically. If you are looking for a word, then there are at least two ways to do it:

Algorithm A:

1. Start on the first page of the book and go word by word until you find what you are looking for.

Algorithm B:

1. Open the book in the middle and check the first word on it.
2. If the word you are looking for is alphabetically more significant, then look to the right. Otherwise, look in the left half.
3. Divide the remainder in half again, and repeat step #2 until you find the word you are looking for.

Which one is faster? The first algorithms go word by word *O(n)*, while the algorithm B split the problem in half on each iteration *O(log n)*. This 2nd algorithm is a **binary search**.

O(n log n) - Linearithmic

Linearithmic time complexity it's slightly slower than a linear algorithm. However, it's still much better than a quadratic algorithm (you will see a graph at the very end of the post).

Examples of Linearithmic algorithms:

- Efficient sorting algorithms like merge sort, quicksort, and others.

O(2^n) - Exponential time

Exponential (base 2) running time means that the calculations performed by an algorithm double every time as the input grows.

Examples of exponential runtime algorithms:

- Power Set: finding all the subsets on a set.
- Fibonacci.
- Travelling salesman problem using dynamic programming.

O(n!) - Factorial time

Examples of O(n!) factorial runtime algorithms:

- Permutations of a string.
- Solving the traveling salesman problem with a brute-force search