Algorithms basics

What is an Algorithm?

An **algorithm** is a finite sequence of well-defined instructions that can be used to solve a computational problem. It provides a step-by-step procedure that convert an input into a desired output.

Algorithms typically follow a logical structure:

•**Input:** The algorithm receives input data.

•**Processing:** The algorithm performs a series of operations on the input data.

•**Output:** The algorithm produces the desired output.

What is the Need for Algorithms?

Algorithms are essential for solving complex computational problems efficiently and effectively. They provide a systematic approach to:

- **Solving problems:** Algorithms break down problems into smaller, manageable steps.
- **Optimizing solutions:** Algorithms find the best or near-optimal solutions to problems.
- **Automating tasks:** Algorithms can automate repetitive or complex tasks, saving time and effort.

Algorithms characteristics

- Inputs and output
    - What does algorithm accept, and what are the result?
- Algorithm complexity
    - Space complexity - How much memory does it require?
    - Time complexity – How much time does it require to complete?

Big-O notation

Big O notation is a way to characterize the time or resources needed to solve a computing problem. It's particularly useful in comparing various computing algorithms and approaches under consideration, such as those used in Machine Learning.

Below is a table summarizing Big O functions. The four most commonly referenced and important to remember are:

- **O(1)** - *Constant* access time such as the use of a hash table.
- **O(log n)** - *Logarithmic* access time such as a binary search of a sorted table.
- **O(n)** - *Linear* access time such as the search of an unsorted list.
- **O(n^2)** - Nested iterations.
- **O(n log(n))** - *Multiple* of log(n) access time such as using Quicksort or Merge sort.

Big-O notation table

Notation	Type	Examples	Description
O(1)	Constant	Hash table access	Remains constant regardless of the size of the data set
O(log n)	Logarithmic	Binary search of a sorted table	Increases by a constant. If n doubles, the time to perform increases by a constant, smaller than n amount
O(<n)	Sublinear	Search using parallel processing	Performs at less than linear and more than logarithmic levels
O(n)	Linear	Finding an item in an unsorted list	Increases in proportion to n. If n doubles, the time to perform doubles
O(n log(n))	n log(n)	Quicksort, Merge Sort	Increases at a multiple of a constant
$O(n^2)$	Quadratic	Bubble sort	Increases in proportion to the product of n*n
$O(c^n)$	Exponential	Travelling salesman problem solved using dynamic programming	Increases based on the exponent n of a constant c
O(n!)	Factorial	Travelling salesman problem solved using brute force	Increases in proportion to the product of all numbers included (e.g., 1*2*3*4...)