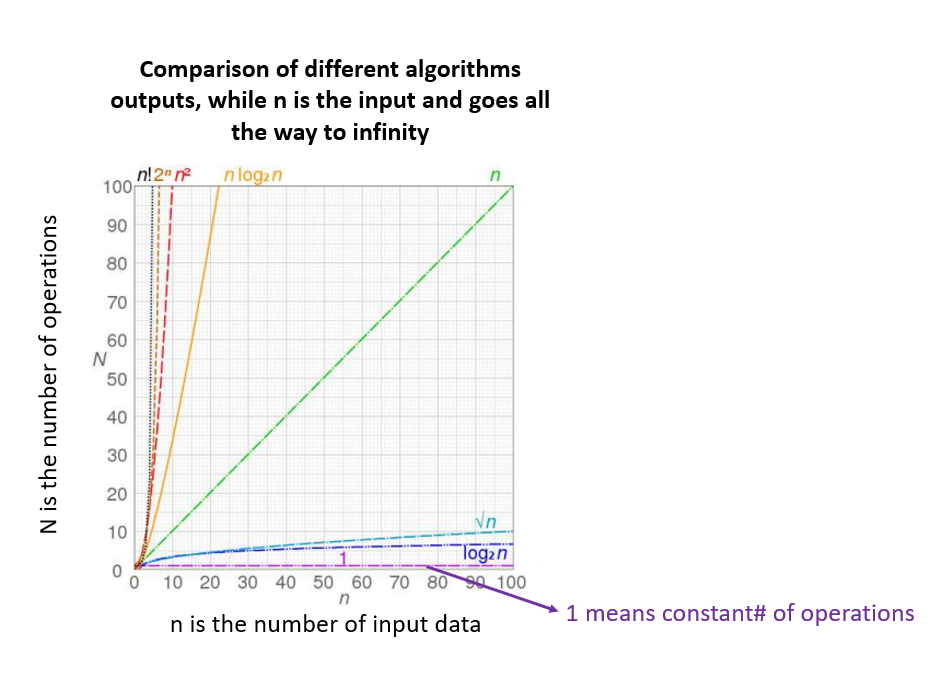
**The n-Notation Introduction**

**N-notation is NOT an analysis of how long an algorithm will take to run, but an analysis of how the algorithm will scale with more and more input data.**

* Is the idea that serves as the basics for the Big O Notation
* ***Def***: It is a representation of a relationship in which n is a function of n.
* Input > operations of the algorithm > output
* We do not know exactly the value of the input. Therefore, we give an arbitrary value of n
* Also, because we do not know exactly the input & do not know the output. But we know that the output is a function of the input n. Then in the simplest terms the input n will need n number of operations to produce the output. Therefore, the output can be looked at as n also
* For Example: when saying n is a function of n we are saying when the input n is equal to 100 pieces of data. Then the output’s solution will at least include 100 operations. Therefore, n output data/operations are a function of n input data i
* Exp: is moving 100 files from one folder to another folder. This requires 100 moving operations. Therefore, n is a function of n
* Notes
* In different types of scenarios, we can have different relationships. For Example:- n > operations of the algorithm > n^2. In this case we will have n^2 operations. Thus, it is larger than n > operations for algorithms > n.
* In **a linear relationship** between the input and the output. We will take n as the input, and the output is a function of n. Therefore, because we do not know exactly how the output will look like we will write it as n
* ***Purpose***:
* The purpose of this notation is that it allows us to analyze how an algorithm scales up or down depending on the input size
* Also, it allows us to compare multiple algorithms without the need to implement these algorithms
* If we design an algorithm we want to know if the algorithm is efficient with more data or less data
* This allows us to make sure that our algorithm will succeed because it will provide scalable solutions
* **A scalable solution**: is a solution that can work efficiently for large sets of input data
* For Example, when n= 100 and we have two algorithms
* Algorithm A -> n -> algorithm -> n. Here we will have 100 inputs and 100 operations for the output
* Algorithm B -> n -> algorithm -> n^2. Here we will have 100 inputs and 1000 operations for the output. The 1000 operations will take much longer to solve that the 100 operations

**The n-Notation Scaling**

* ***Goals of n-Notation***
* The goal is to look for large patterns on the output of multiple algorithms theoretically, while having an infinite number of input data and compare their efficiency
* Large patterns: are a significant change within graphs. For exp, a large exponential change or a constant unchanged line in a graph due to log
* ***Rules of n-Notation during LARGE scale operations when our input goes to infinity***
* 1. We do not care about multiples
* For example, n^3 -> algorithm -> 3,000,00 (n^3)
* We do not care about the multiple number, because at some point, when n becomes large enough that multiple no matter how big it is. It will not matter. Remember n can go to infinite
* 2. We take the largest in an equation
* For example, n^3 -> algorithm -> (n^2 + n log(n))
* Here, n long(n) will produce some insignificant while the input goes to infinity in comparison to n^2. Therefore, we focus on n^2 when comparing this type of algorithm, because it will produce the largest results
* Below you can see that the most optimum algorithm (based on a large scaled input) is when the number of operations is constant or when we are using log n or sqrt(n)



**The n-Notation Example**

* Compare two algorithms in terms of number of cycles and amount of time that each algorithm takes to run the same input. If each cycle takes 0.001 seconds. Calculate the faster algorithm
* Compare nlogn and n^2. The log is base 2
* At 1000 cycles
* At 1000 cycles -> nlongn -> 1000log(1000) =9965 cycles -> 9965 \*0.001 = 9.9 seconds
* At 1000 cycles -> n^2 -> 1000^2 =1000000 cycles -> 1000000\*0.001 = 1000 seconds
* At 25000 cycles
* At 25000 cycles -> nlongn -> 25000 log(25000) =365241.0119 cycles -> 109948.5 \*0.001 = 365.24 seconds
* At 25000 cycles -> n^2 -> 25000 ^2 =625000000 cycles -> 625000000 \*0.001 = 625000 seconds = 173.61 hours