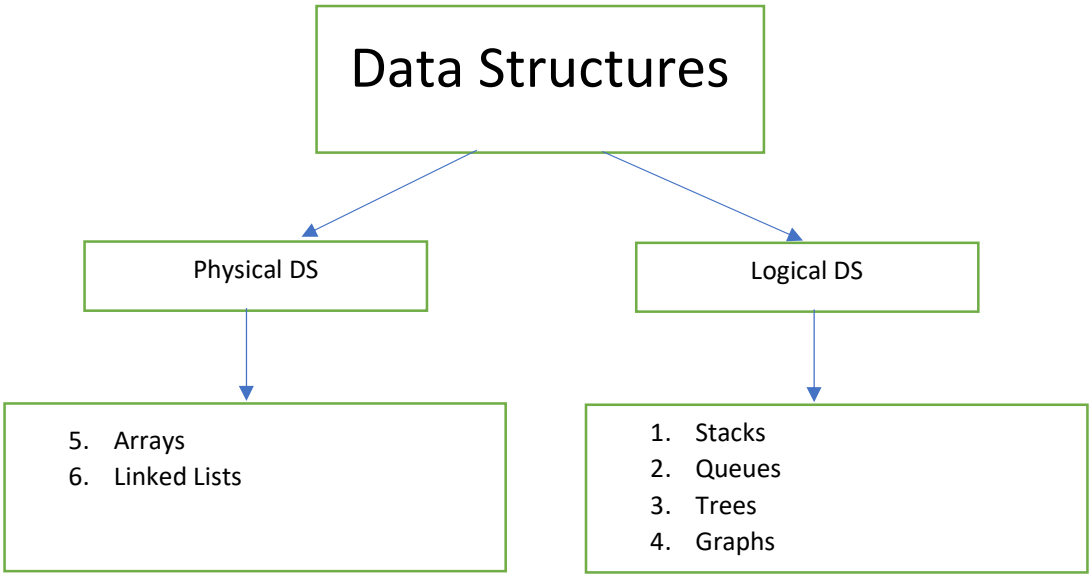
What are the different types of Data Structures to learn?

1. LinkedList
2. Stacks
3. Queues
4. Trees
5. Graphs



What is Data Structure?

It is nothing but the way the data is organized inside the memory. For example, in a linked list, the data is organized in form of nodes that are interconnected with each other.

Real time usages of various Data Structures: -

- **LinkedList**: - Music player.
- **Stack**: - Recently opened web pages inside a browser.
- **Queue**: - Job scheduling in Operating systems.
- **Trees**: - Folder structures in Operating Systems.
- **Graphs**: - Google Maps.
- **Trie**:- Red lines in a editor.

There are different ways to achieve something. Like, if you want to travel from one place to other, you can do that using different ways, like you take a flight, train, bus, car or even you can walk. Every approach has its own pros and cons. Like taking a flight is faster but not cost efficient, and this may not be applicable every time. You cannot take a flight to travel to next street.

How to compare Algorithms?

Algorithm comparison should be independent of programming language and external factors like hardware. Ideal way to compare two algorithms is to compare their growth rate.

Say, you need to send a file to your friend. How can you do that?

1- By sending an email or via FTP.
2- By taking that file in a hard drive and by travelling by yourself to you friend and handing it over.

In case 2, the time would be constant and it is irrespective of the file size. Like even if it is 1TB file, the travel time taken is the same.

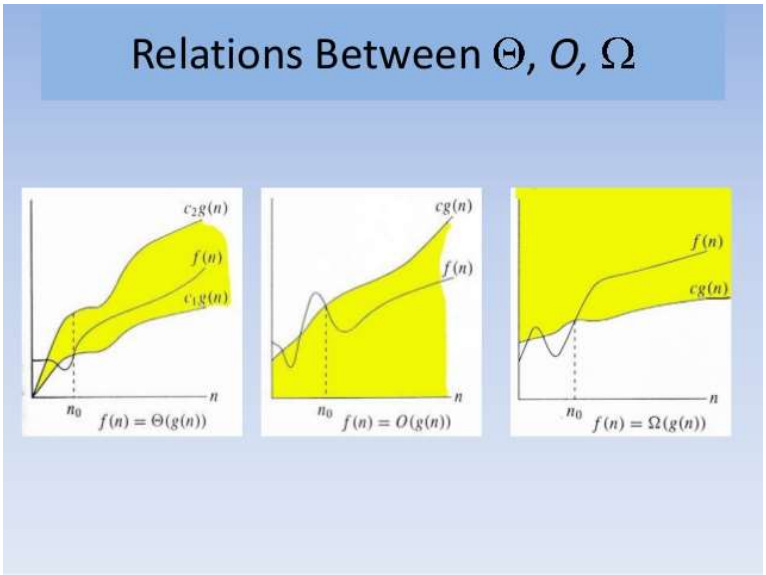
But, in case 1, if it's a small text file, it is sent over internet in no time. But, if the file is larger, say 1 TB, it takes days to transfer that file electronically. So the growth rate depends on the input size of your file.

Case 2 has constant Time Complexity. We denote it using O(1).

Types of Analysis: -

1. **Best Case** − Minimum time required for program execution. **Ω (Big-Omega)**
2. **Average Case** − Average time required for program execution. **Θ (Theta)**
3. **Worst Case** − Maximum time required for program execution. **O (Big-O)**

Diagrammatic notation: -



Mercy Technologies

An algorithm can be represented in form of an expression.

Example: -

f(n)=3n+2
3n+2<=4n for all n>=3

Hence 3n+2 = O(n)

Say, you went to buy a pair of Shoes and Socks. Your friend came and asked you what you are buying. You simply say that you came to buy pair of shoes. You simply ignore socks, even if you had already bought them. The reason why you ignore is that the value of socks is negligible in comparison to that of shoes.

Similarly,
in **f(n)=3n+2**, we ignore 3 and 2 as they are constants the time complexity becomes **O(n)**
If, **f(n)=3n^2+2n+3**, we ignore all and the time complexity becomes **O(n^2)**

We only concentrate on Big-O notations as we analyse worst cases. There is no point of analysing best case because, most algorithms will be O(1) in the best case.

Problem	Number of Steps	Time Complexity	Space Complexity
`for (int i = 0; i < 5; i++) { //Some action }`	5	O(1)	O(1)
`for (int i = 0; i < n; i++) { //Some action }`	n	O(n)	O(1)
`for (int i = 0; i < n; i++) { for (int j = 0; j < n; j++) { //Some action } }`	n^2	O(n^2)	O(1)
`for (int i = 0; i < n; i++) { //Some action } for (int j = 0; j < n; j++) { //Some action }`	n+n	O(n)	O(1)
`for (int i = 0; i < n; i=i+2) { //Some action }`	n/2	O(n)	O(1)
`for (int i = 1; i <= n; i=i*2) { //Some action }`	log n	O(log n)	O(1)
`for (int i = n; i > 0; i=i/2) { //Some action }`	log n	O(log n)	O(1)
`int[] input; int sum; private int sum(input) { for (int i = 0; i < input.length; i++) { sum+=input[i]; } return sum; }`	n	O(n)	O(n)