DATA STRUCTURES

Topic

1.) Introduction of Data structure

2.) DS Array
- DS Array
- 2D Array

3.) DS Linked List
- Introduction of LL
- Types of LL
- singly Linked List
- Doubly Linked List
- circular Linked List
- circular doubly Linked List

4.) DS Stack
- Intro of stack
- Array Implementation
- Linked List implementation
  - Push, Pop & other operation

5.) DS Queue
- Intro
- Types of queue
- Array Represent
- circular queue
- Deque
- Priority queue

6.) DS Tree

- → Introduction of Tree
- → Binary tree
- → Binary search tree
- → AVL Tree
- → B Tree
- → B+ tree

7.) DS Graph

- → DS Graph
- → Graph Implementation
- → BFS & DFS Algorithm
- → Spanning Tree

8.) DS Searching

- → Linear search
- → Binary search

9.) DS Shorting

- → Bubble sort
- → Insertion sort
- → merge sort
- → Quick sort
- → selection sort
- → Bucket sort
- → Heap sort
- → counting sort
- → Radix sort

11.) Differences

- → Linear vs non linear
- → Array vs Linked List
- → Stack vs Queue
- → Linear vs circular queue
- → LS vs BS
- → Singly Linked List & Doubly Linked list
- → Binary or Binary search tree
- → tree vs Graph
- → BST vs AVL tree
- → Red Black tree vs AVL tree
- → B tree vs B+ tree
- → Quick ~~test~~ sort & Merge sort
- → BSF vs DSF
- → stack vs Heap
- → Stack vs Array
- → Bubble sort vs selection sort
- → Full binary tree vs complete Binary tree
- → Binary tree vs B tree

References :- Javapoint

1.) Introduction to Datastructure

Data structure

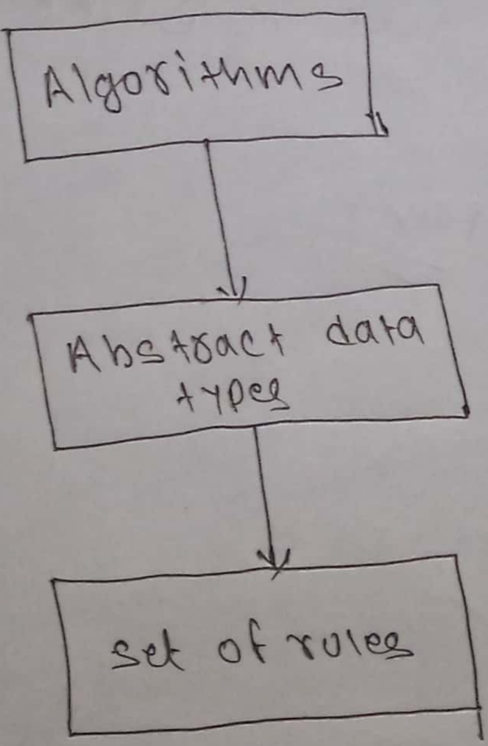
A data structure is a particular way to organizing data in a computer, so that it can be used effectively. There are many ways of organizing the data in memory.

EX.:- Array, Linked List, Stack, Queue

The data structures is not a programming language like C, C++, Java etc. It is a set of algorithms that we can use in any programming language to structure the data in memory.

NOTE:-

To structure the data in memory, 'n' number of algorithms were proposed, and all these algo are known as Abstract data types. These abstract data type are the set of rules.



* Types of DS

1.) Primitive data structures

2.) Non primitive data structure

1.) Primitive Data structure

The primitive data structures are primitive data types. like int, char, float, double and pointer are the primitive data structures that can hold a single value.

2.) Non-primitive Data structure

Non-primitive data structures are 2 types

1.) Linear data structure

2.) Non-Linear data structure

1.) Linear Data structure

All the data and elements are organized in the linear order. In linear data structures the elements are stored in non-hierarchical way where each elements each elements has the successors and predecessors expect the first and last element.

Types of Linear Data structures

1.) Array

2.) Linked List

3.) Stack

4.) Queue

2.) Non Linear Data structures

The data structures does not form a sequence i.e. each item or element is connected with two or more other item in a non-linear arrangement. The DS are not arranged in sequential structure.

Types of Non-Linear Data structures

1.) Tree

2.) Graph

* Operations on data structures

1.) Traversing

2.) Insertion

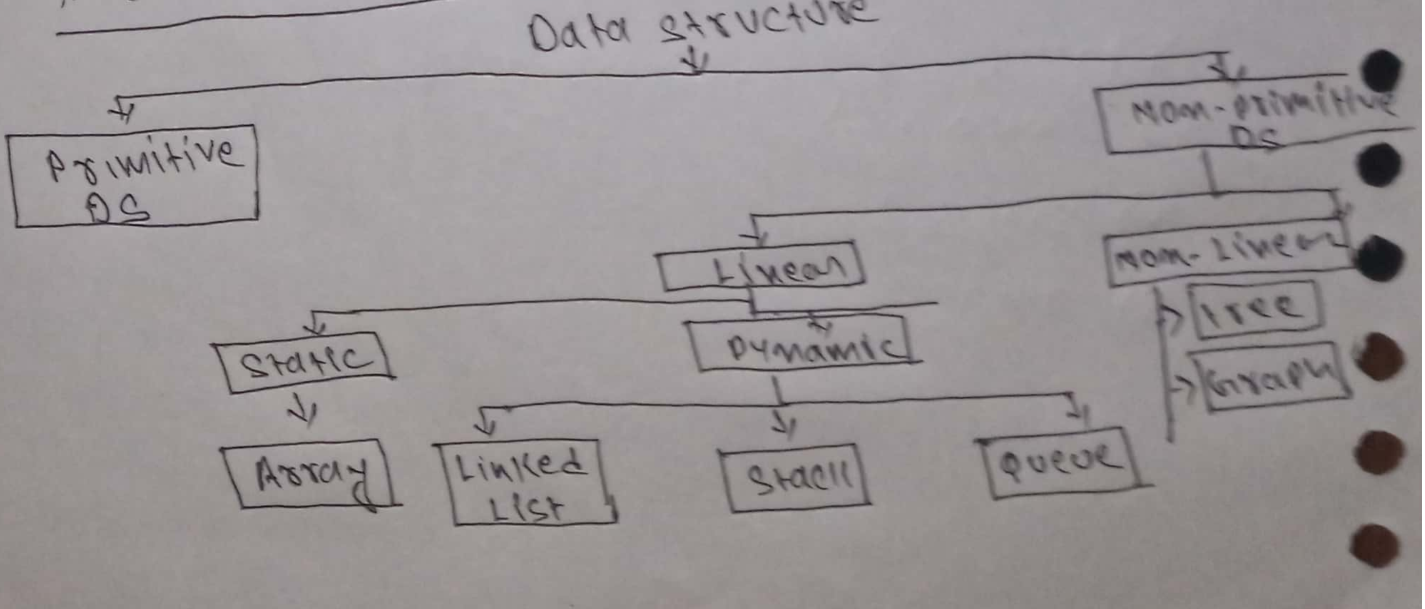
3.) Deletion

4.) Searching

5.) Sorting

6.) Merging

* Data structure classification



* Ds Algorithms

Algorithm is a set of Rules to solve any problems.

Characteristics of an Algorithms

Input, output, unambiguity, finiteness, Effective-ness, Language independent.

Dataflow of an Algorithms

1.) Problems
2.) Algorithms
3.) Input
4.) Processing unit
5.) output.

Ex. of Algorithms

1.) shorting
2.) searching
3.) Delete x
4.) Insert x
5.) update. x

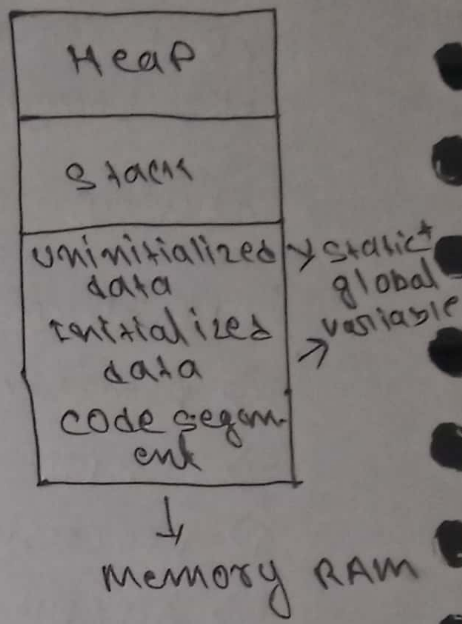
Other technology

1.) Database :- collection of information in permanent storage for faster retrieval and updation.

2.) Data warehousing :- management of huge amount of legacy data for better analysis.

3.) Big data :- Analysis of too large or complex data which cannot be deal with traditional data processing application.

Memory layout



→ when the program starts, its code is copied to the main memory.

→ Stack hold the memory occupied by the functions

→ Heap contains the data which is requested by the program as dynamic memory.

→ Initialized and uninitialized data segment hold initialized and uninitialized global variables respectively.

* Time complexity & Big O Notation

Time complexity :- Time complexity is the amount of time taken by an algorithms to run, as a function of the length of the input.

EX :- consider 2 developers who created an algorithm to sort n numbers. when ran for input size n, following results were recorded.

No. of elements(n)	Algo 1	Algo 2 5
10 elements	90ms	122 ms
70 elements	110 ms	124 ms
110 elements	180 ms	131 ms
2000 elements	2s	800 ms

As we can say that Algo1 was shining for similar input but as the number of elements ~~input but as the numb~~. increases Algo 2 looks good.

Ex:- Sending GTA V to a friend

Let us say you have a friend living 5 kms away from your home. You want to send him a 60 GB file game. How will you send it to him?

Note
Both of you are using JIO 4G with 1 Gb/day data limit.

Best way to send the game is by delevering it to his home. copy the game to a HDD and send it.

But when you do the same thing for sending small size of game like 1 mb, 2 mb. you can send it by online.

→ As the file size grows, time taken by online sending increases linearly → $O(n^1)$

→ As the file size grows, time taken by physical sending remains constant → $O(n^0)$ or $O(1)$

Calculating order in terms of input size

Algo 1 → $\underset{\substack{\text{Highest}\\ \text{order}\\ \text{terms}}}{K_1 n^2} + \underbrace{K_2 n + 36}_{\substack{\text{can ignore lower}\\ \text{order terms}}}$ → $O(n^2)$

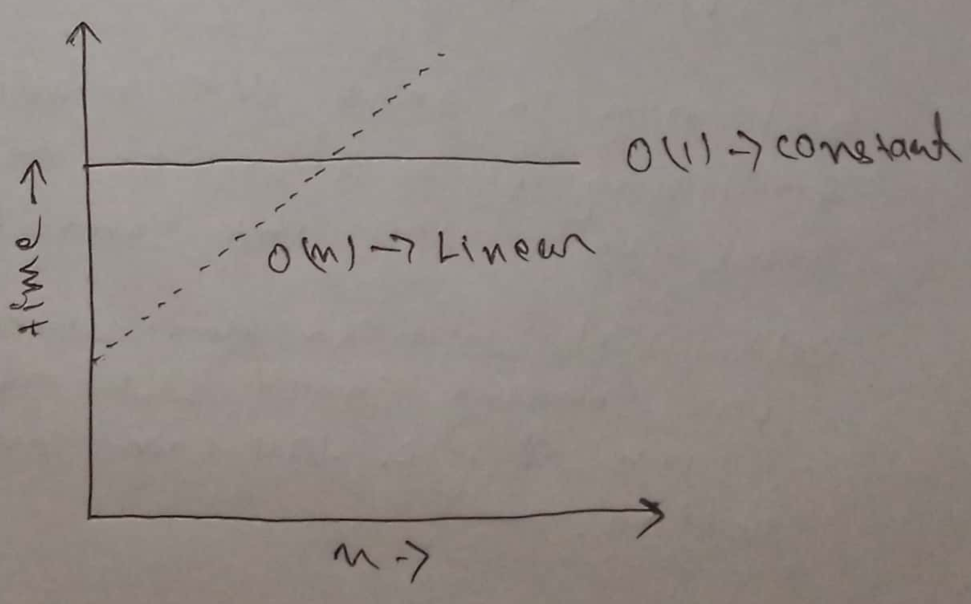
Algo 2 → $K_1 K_2^2 + K_3 K_2 + 8$

↓

$K_1 K_2^2 n^0 + K_3 K_2 + 8$ → $O(n^0)$ or $O(1)$

Visualising Big O

Plot $O(1)$ and $O(n)$ on a graph



Asymptotic Notation

It gives an idea about how good a given algorithm is compare to some other algorithm

1.) Big oh Notation (O)

2.) Omega Notation (Ω)

3.) Theta notation (θ)

1.) Big oh Notation

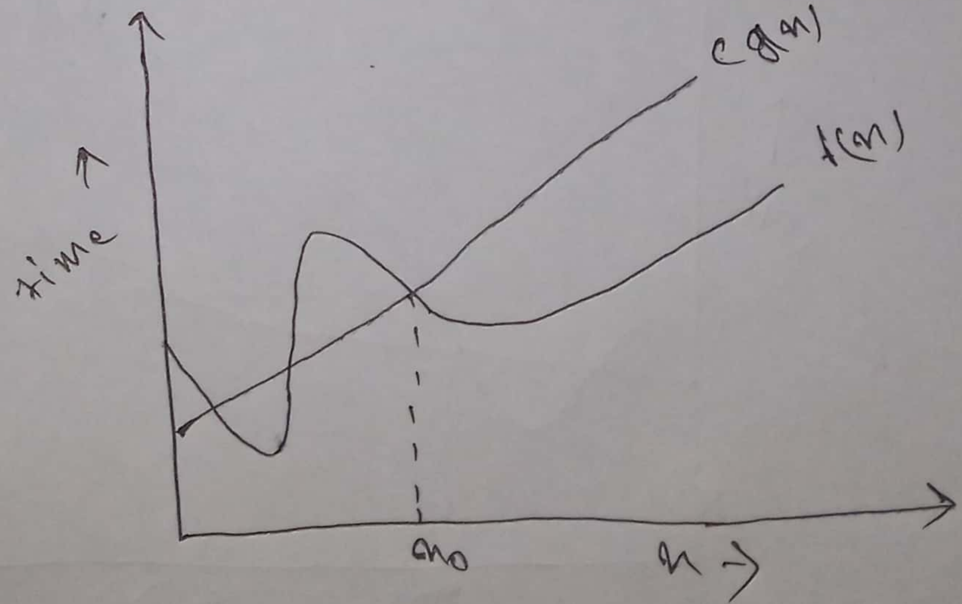
Big oh notation is used to describe asymptotic upper bound.

$0 \le f(n) \le c\, g(n)$ for all $n \geq n_0$

Used to give upper bound on a fun.

if a function is O(n), it is automatically O(n^2) as well.

Graph



2) Big omega notation

Just like O notation provides an asymptotic upper bound, Ω notation provides asymptotic lower bound. Let f(n) define running time of an algorithms.

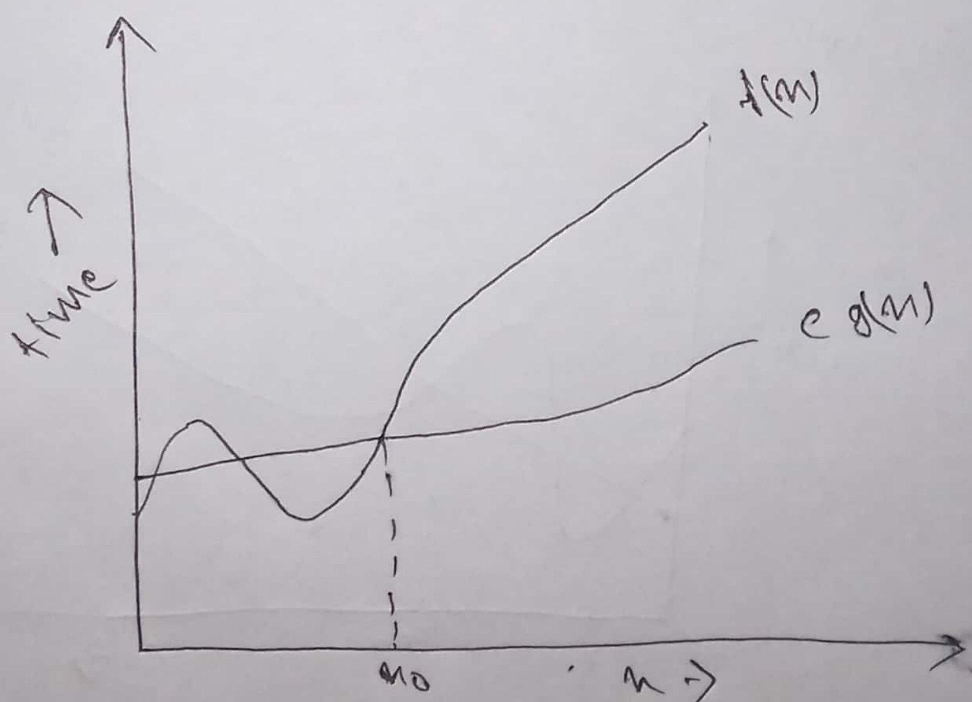
f(n) is said to be Ω g(n) if there exists positive constants c and no such that

$0 \le c\,g(n) \le f(n)$ for all $n \geq n_0$

$\Downarrow$ used to give lower bound on a function

If a function is $O(n^2)$ it is automatically $O(n)$ as well

Graph



3.) Big theta notation

Let f(n) define running time of an Algo f(n) is said to be θ g(n) if f(n) is O g(n) and f(n) is Ω(g(n))

Mathematically

$0 \leq f(n) \leq c_1 g(n) \ \forall \ n \geq n_0$

$0 \leq c_2 g(n) \leq f(n) \ \forall \ n \geq n_0$

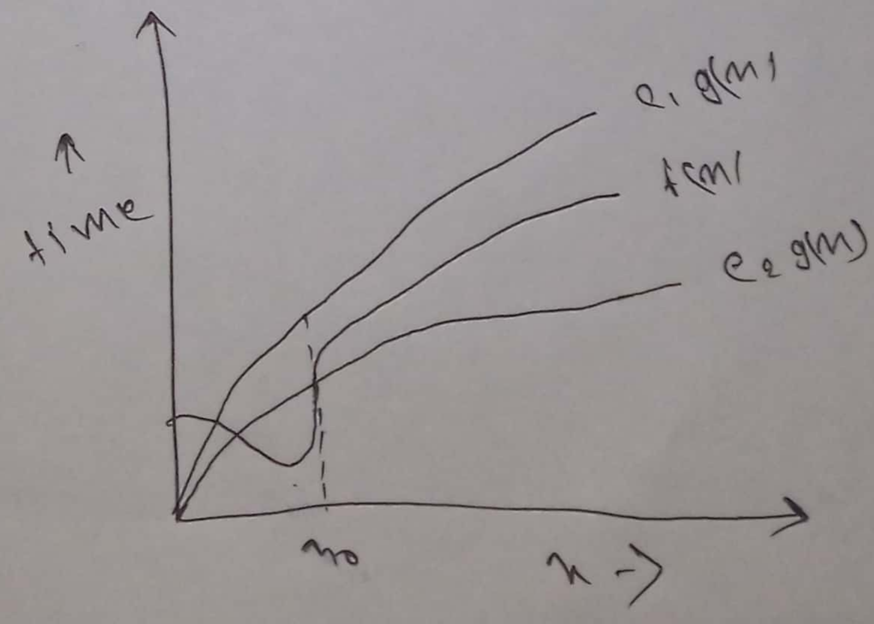
$\forall$ sufficiently large value of n

we get

$$0 \leq c_2 g(n) \leq f(n) \leq c_1 g(n) \ \forall \ n \geq n_0$$

The equation simply means there exist positive constants c_1 and c_2 such that f(n) is sandwiched between c_2 g(n) and c_1 g(n)

Graph



Since Big θ give a better picture of runtime for a given algorithms, most of the man- expect you to provide an answer in terms of Big theta when they say "order of"

Worst case, best case, Average case

1.) Worst case :- It defines the input for which the algorithms take a huge time.

2.) best case :- It takes average time for the program execution.

3.) Average case :- It defines the input for which the algorithm take the lowest time.

2.> Data structure Array and ADT

ADT's are the way of classifying data structures by providing a minimal expected interface and set of methods.

ADT → minimal required functionality
ADT → operations

* Array ADT

An array ADT holds the collection of given elements accessible by an index.

minimal functionality → get (i) → get element i
set (i, num) → set element i to num.
represent.

Operations

→ max()
→ min()
→ search()
→ Insert (i, num)
→ Append (i)

Static Array

→ size cannot be changed

Dynamic Array

→ size can be changed

* memory represent of Arrays

Index → 0 1 2 3

2	4	10	12

→ Array of size 4

address → 10 14 18 22 26

→ Element in array are stored in contiguous memory location.

→ Elements in an array can be accessed using the base address in constant time → O(1)

Array

The collection of similar type of data item stored at contiguous memory locations.

Ex:- int arr[10], char ary[10], float ary[20]

Operation on Array

	AC	WC
1.) Access	O(1)	O(1)
2.) search	O(n)	O(n)
3.) insertion	O(n)	O(n)
4.) deletion	O(n)	O(n)