Introduction to DSA.

Basic terminology

A program is said to efficient when the code executes in minimum time and minimum memory space.

DSA Contains Complex tasks like

→ Data Collection
→ organization of data
→ developing and maintaining routines

DSA is applied on following areas:

→ Compiler design → storing the variable, functions
→ syntax Analysis
→ semantic Analysis
→ operating system → Process, memory, File management.
→ DBMS → managing large data sets
→ statical analysis package → store and manage data sets.
→ Numerical analysis → vectors, ..etc (eigen values)(matrix)
→ AI → search Algo, M.L,
→ simulation → event, Discret event simulation.
→ Graphics - collision detection → (BVH's, K-d trees are used)

Classification of data structures

Primintive and non-primitive

Primintive datatypes are those which are fundamental datatypes of a programming lang.

Non-primitive datatypes are those which are created by the help of P.D.T. → linear
→ Non-linear.

Non-primitive datatype — linear
non-linear

Linear :- it is the data, where it is stored in the sequential order. Ex:- Arrays, linked lists, stacks, queue

→ In memory it can be represented in two ways
① Sequential memory locations
② means of links.

Non-linear: it is the data, where it is not stored in the sequential order.
Ex: trees and graphs.

Array :- it is a collection of similiar data elements.
→ in this data is stored in consecutive memory locations.

Syntax:

type name [size];

Ex:

int stu[60];

Limitations of Array :-
→ Arrays are fixed
→ memory locations are stored consecutively which it may not be available.
→ Insertion and deletion is problematic.

Linked list :-
→ flexible, dynamic data structure, nodes are of sequential order.
→ Every node in the list points out to the next node (elements)
→

* Every node contains the following two types of data:-

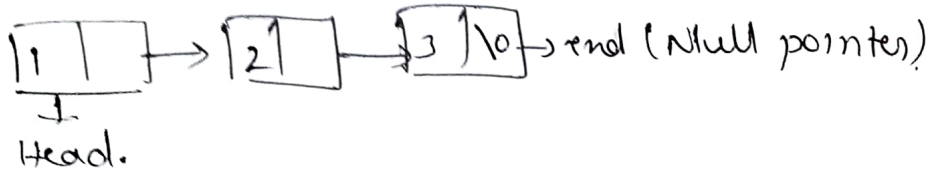
① The value of the node

② A pointer.

→ the last node points to the NULL, so, it indicates the final node.

→ As it is dynamical allocated, it will stored nodes to list limited to the memory avaliable

Eg:



Stacks :-

→ It is linear D.S in which insertion and deletion of elements are done at only one end, which is known as the top of the stack

→ It is (LIFO) Last-In, first out structure.

→ when the new element is added at last then the first one will be deleted.

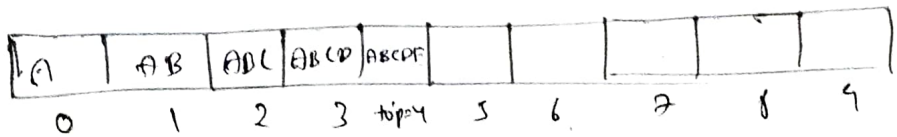
→ Every stack has variable "top", It is the position where the element is added (or) deleted.

→ Another variable "MAX", which is used to store the maximum elements that the stack can store.

if top = Null (Stack is empty)

if top = MAX-1 (Stack is full).

Eg:



Array representation of stack

Three operations of Stack :-

1. push :- adds an element to the top of the Stack
2. pop :- removes an element from the top of the stack
3. peep :- returns the value of the topmost of the stack

→ before inserting any value in stack, we should check for overflow, it means checking for availability of memory in the storage.

Queues :- (FIFO) (First In first out)

→ element that is inserted first is the first one to be taken out.

→ elements in a queue added at one end called the rear and elements that are removed from other end is called front.

→ every queue has "front" and "rear" variable.

For example :-

12	9	7	18	14	36			
0	1	2	3	4	5	6	7	8	9
front					rear			

Now, if you add new element 45 at rear then rear will be incremented by 1.

After addition :

12	9	7	18	14	36	45			
0	1	2	3	4	5	6	7	8	9
top						rear			

Now, you want to remove an element, the variable front will be increased, deletion is done only from this end.

	9	7	18	14	36	45		
0	1	2	3	4	5	6		
	top					rear		

→ overflow : checking the availability of memory before inserting. if not overflow takes place

→ underflow : checking the availability of data in Queues before deleting. if not underflow takes place.

Trees :

A tree is a non-linear data structure, where the data is arranged in a hierarchical order.

→ one of the node is the root node, remaining nodes are partitioned into sub nodes. (sub-tree of the root).

Binary tree : it consists of one root node and left, right sub-trees, both the sub-trees are also binary trees.

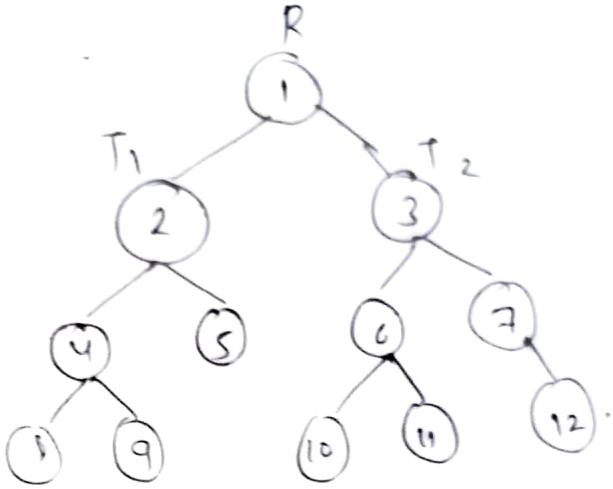
→ each node contains a data element

→ A left pointer which points to the left sub-tree

→ A right pointer which points to the right sub-tree

→ The root element is the topmost node, which pointed by name 'root' pointer.

→ if root = Null then the tree is empty.



R is the main root element.

T_1, is the non-empty so, it is called left successor

T_2 is the non-empty so, it is called right successor

Graphs :- (Non-linear DS)

→ Collection of vertices (called nodes) and edges that connect that vertices.

→ Here, instead of parent-child relationship between tree nodes, any kind of complex relationships b/w the nodes can exist.

→ In tree node, it contains one parent, but in graphs you may contain many ~~ne~~-

→ it is used for graph operations, searching the graph, finding the shortest path

Operations on Data structures :-

Traversing : it is going through the collection of data once and doing something with it along the way - reading, processing, analyzing.
→ every element gets attention.

Searching : it is to find the location of one or more elements satisfy the constraint.

Inserting : add of new items to the given list.

Deleting : remove a particular data item from the collection of data items.

Sorting : Data items can be arranged in some order like ascending (or) descending order. depending on the type of application.

Merging : list of two sorted data items can be combined. to form a single list of sorted data items.

Abstract data type :-

data type :- it is the common data types like int, char, float, double, boolean .. etc.

Abstract : it means considered apart from the detailed specifications (or) implementations.
→ focusing on the essential features and behaviours of a data structure while ignoring the details of how it is implemented

Algorithms :-

→ procedure to performing some calculations

→ it is like blueprint for solving the problem.

Different Approaches to designing an algorithm

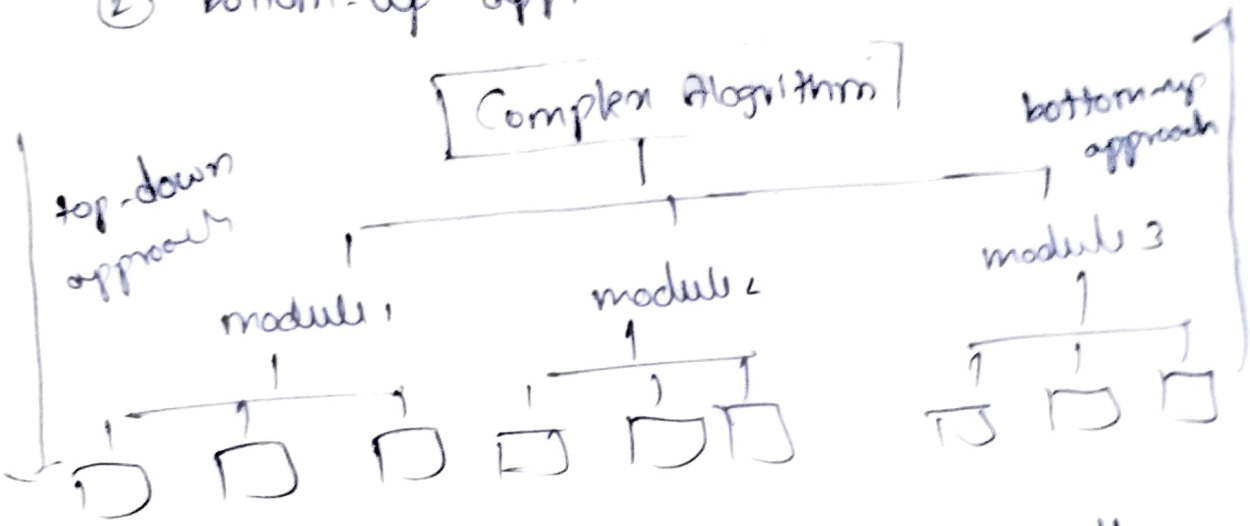
→ A Complex Alogrithm is divided into smaller parts.

→ each module is designed independently.

→ Two approaches for Algorithm designing

① Top - down approach

② bottom - up approach.



Top - down approach :- It is the making the Complex Alogrithm into smaller modules and solving them

→ it is a stepwise refinement

→ process of decomposition.

Bottom - up approach :- It is the reverse of top-down approach. In the bottom-up design, we start with designinning the most basic or higher levels.

As we compared top-down approach and bottom up approach, bottom-up approach is used because of Abstraction, where the data is encapsulated within a module, the user cannot see it.

* TIME AND SPACE COMPLEXITY:-

~~Time Complexity: It is which a program executes~~

is

Time Complexity :- It is basically the running time of a program as a function of the input size.

Space Complexity :- It is the storage required during the program execution as a function of the input size.