Basic Data Structures

Unit 1 : Introduction to Data Structures

Prepared by :

Prof. Khushbu Chauhan

Computer Engg. Dept.

MPSTME, NMIMS

Outlines

- Introduction
- Basic Terminology: Elementary Data Structure Organization
- Classification of Data Structures
- Operations on Data Structures
- Abstract Data Type

Introduction

- A computer is an **electronic device** that used for storing, processing and retrieving the information/data.
- It is an Electronic device which accept data from User and manipulates or process it according to instruction which gives correct output.
- It can solve highly complicated problems quickly and accurately.

Common Problems

- Data Search − Consider an inventory of 1 million(10^6) items of a store. If the application is to search an item, it has to search an item in 1 million(10^6) items every time slowing down the search. As data grows, search will become slower.
- Processor Speed − Processor speed although being very high, falls limited if the data grows to billion records.
- Multiple Requests − As thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.

Need of Data structure

- When programmer collects such type of data for processing, he would require to store all of them in computer's main memory.
- In order to make computer work we need to know
  - Representation of data in computer.
  - Accessing of data.
  - How to solve problem step by step.
- For doing this task we use data structure.

Data structure

- **Data structure** is a representation of the logical relationship existing between individual elements of data.
- Data Structure is a systematic way of organizing all data items that considers not only the elements stored but also their relationship to each other.
- The representation of particular data structure in the main memory of a computer is called as **storage structure.**
- The storage structure representation in auxiliary memory is called as **file structure**.

- Following terms are the foundation terms of a data structure.
- Interface − Each data structure has an interface. Interface represents the set of operations that a data structure supports. An interface only provides the list of supported operations, type of parameters they can accept and return type of these operations.
- Implementation − Implementation provides the internal representation of a data structure. Implementation also provides the definition of the algorithms used in the operations of the data structure.

Characteristics of a Data Structure

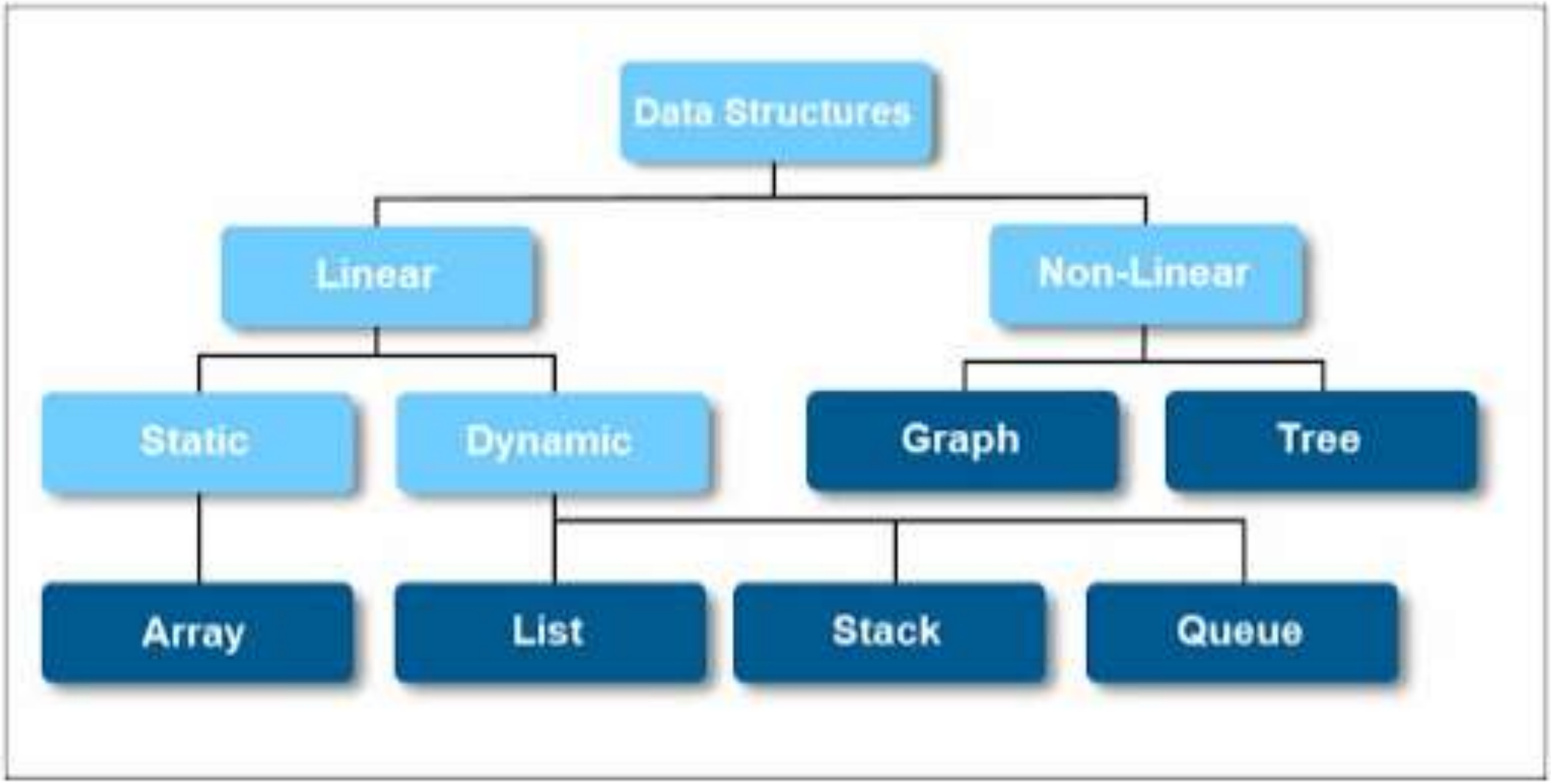
- Correctness – Data structure implementation should implement its interface correctly.
- Time Complexity – Running time or the execution time of operations of data structure must be as small as possible.
- Space Complexity – Memory usage of a data structure operation should be as little as possible

Basic Terminology

- Data – Data are values or set of values.
- Data Item – Data item refers to single unit of values.
- Group Items – Data items that are divided into sub items are called as Group Items.
- Elementary Items – Data items that cannot be divided are called as Elementary Items.
- Attribute and Entity – An entity is that which contains certain attributes or properties, which may be assigned values.

- Entity Set – Entities of similar attributes form an entity set.
- Field – Field is a single elementary unit of information representing an attribute of an entity.
- Record – Record is a collection of field values of a given entity.
- File – File is a collection of records of the entities in a given entity set.

Classification of Data Structures



Linear data structures

- A data structure is said to be Linear, if its elements are connected in linear fashion by means of logically or in sequence memory locations.
- There are two ways to represent a linear data structure in memory,
  - Static memory allocation – Fixed memory size
  - Dynamic memory allocation – Memory allocation at runtime

Array

Index: 0	1	2	3	4	5
1	2	3	4	5	6
Arr[0]	**arr[1]**	**arr[2]**	**arr[3]**	**arr[4]**	**arr[5]**

Array

- An array is a collection of similar data elements. These data elements have the same data type. The elements are stored in consecutive/contiguous memory locations and are referenced by an ***index***.

data type variable_name[size];

- Arrays are of fixed size.
- Insertion and deletion of elements can be problematic because of shifting elements from their positions.

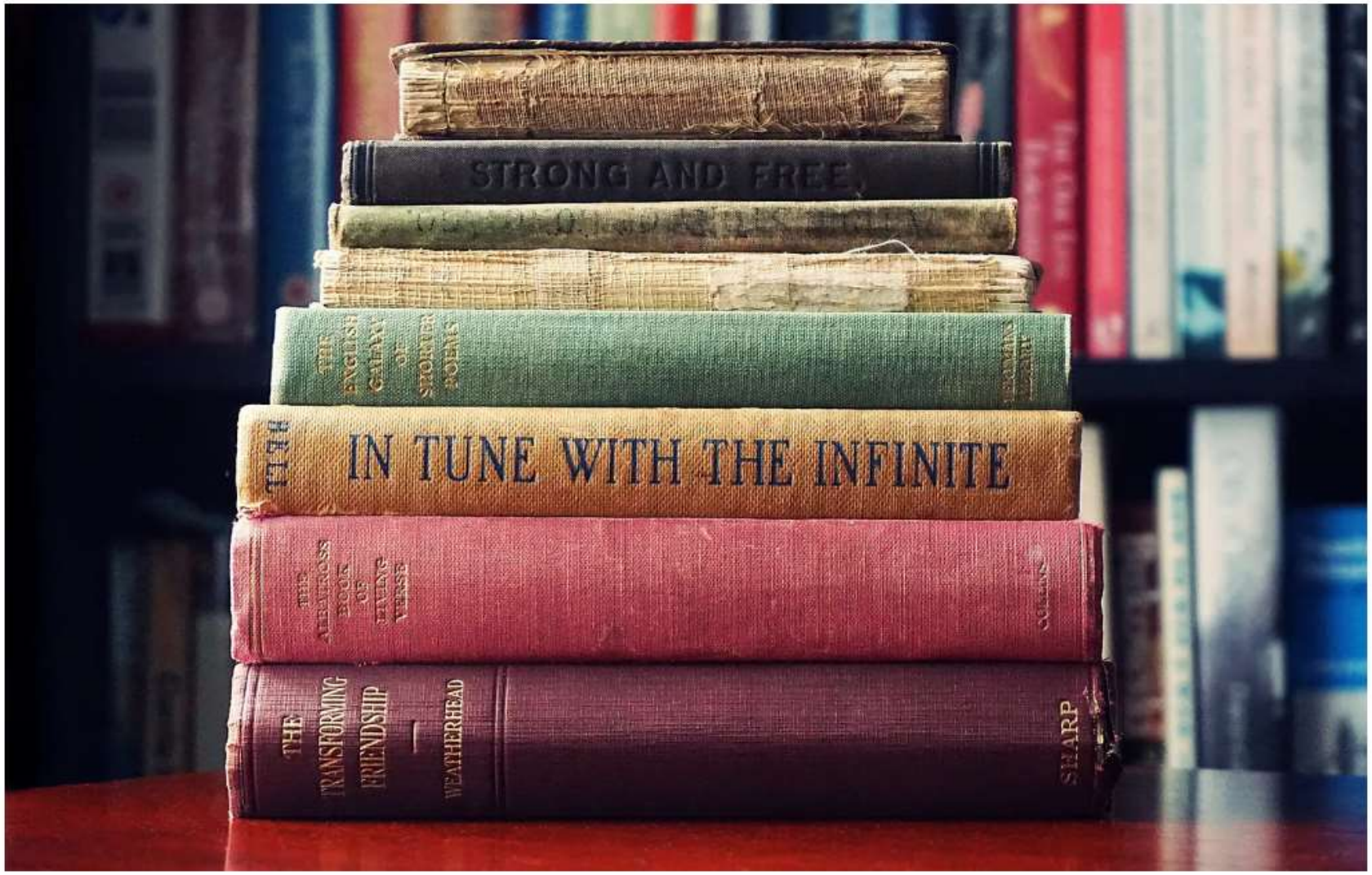
SVKM'S NMIMS
Deemed to be UNIVERSITY

Linked List

Linked List

- A linked list is a very flexible, dynamic data structure in which elements form a sequential list.
- Each node is allocated space as it is added to the list. Every node in the list points to the next node in the list. Therefore, in a linked list, every node contains the following two types of data:
    - The value of the node
    - A Pointer or link to the next node

Stack


STRONG AND FREE
THE ENGLISH GALAXY OF SHORTER POEMS
IN TUNE WITH THE INFINITE
THE ALBATROSS BOOK OF LIVING VERSE
THE TRANSFORMING FRIENDSHIP
WEATHERHEAD
SHARP

Stack

- Stack is called a last-in, first-out (LIFO) structure because the last element which is added to the stack is the first element which is deleted from the stack.
- Insertion and deletion operations are done only from one end.
- In the computer's memory, stack can be implemented using arrays or linked list.

- **PUSH**
- **POP**
- **PEEp**

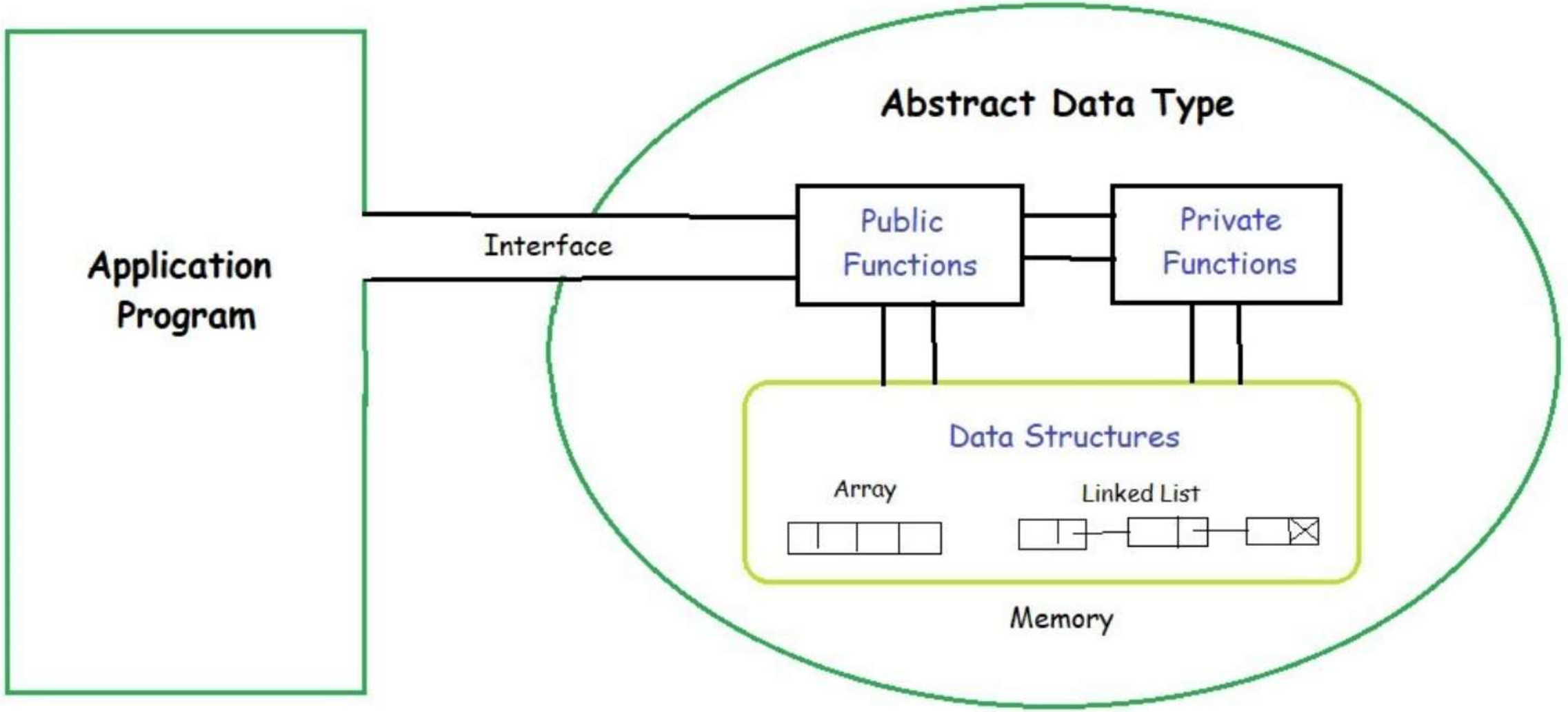
Queue

Queue

- A queue is a first-in, first-out (FIFO) data structure in which the elements that is inserted first is the first one to be taken out.
- The elements in a queue are added at one end called the ***rear*** and removed from the other end called the ***front***.
- Operations performed on queue are:
  - Enqueue
  - Dequeue

Abstract Data Type

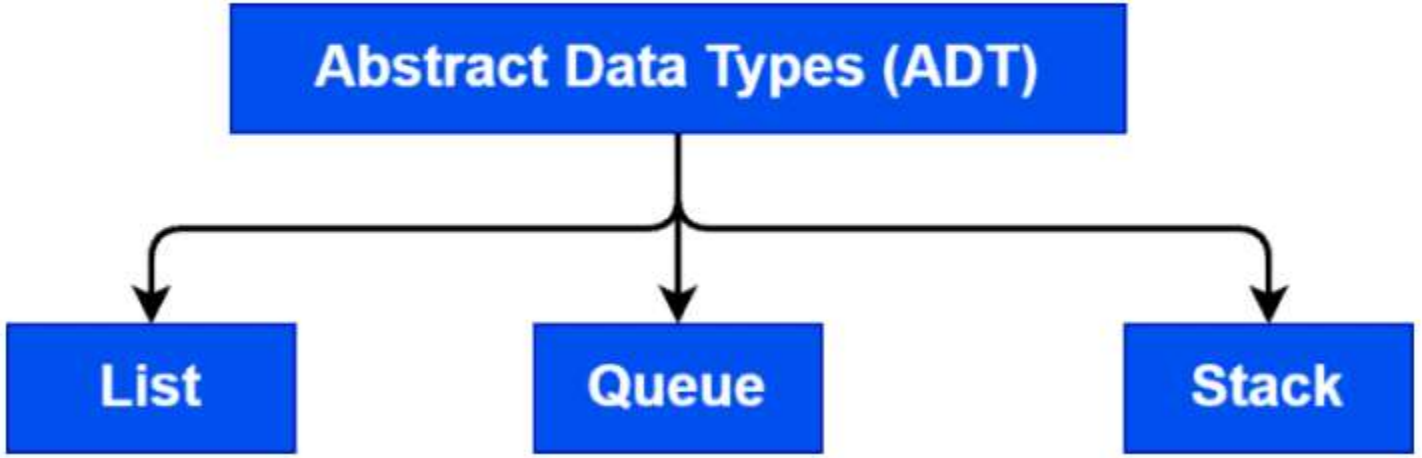
- Built-in data types like int, float, and double support basic operations (addition, subtraction, etc.). For user-defined data types, we define specific operations as needed. To simplify problem-solving, we create data structures with their own operations. These non built-in data structures are called Abstract Data Types (ADTs).
- An Abstract Data Type (ADT) defines a type by its values and operations without specifying how they are implemented. It focuses on what operations are performed, not how they are executed, providing an implementation-independent view.


Abstract Data Type
Application Program
Interface
Public Functions
Private Functions
Data Structures
Array
Linked List
Memory


ABSTRACT DATA TYPES
Its enough for me to know how to use a mobile phone
I dont want to know how a mobile phone works

- A user does not need to know how a data type is implemented to use it. For example, we use primitive types like int, float, and char without knowing their internal workings. Similarly, ADTs are like black boxes that hide their inner structure and design. Users only need to know what a data type can do, not how it is implemented. Examples of ADTs include List ADT, Stack ADT, and Queue ADT.

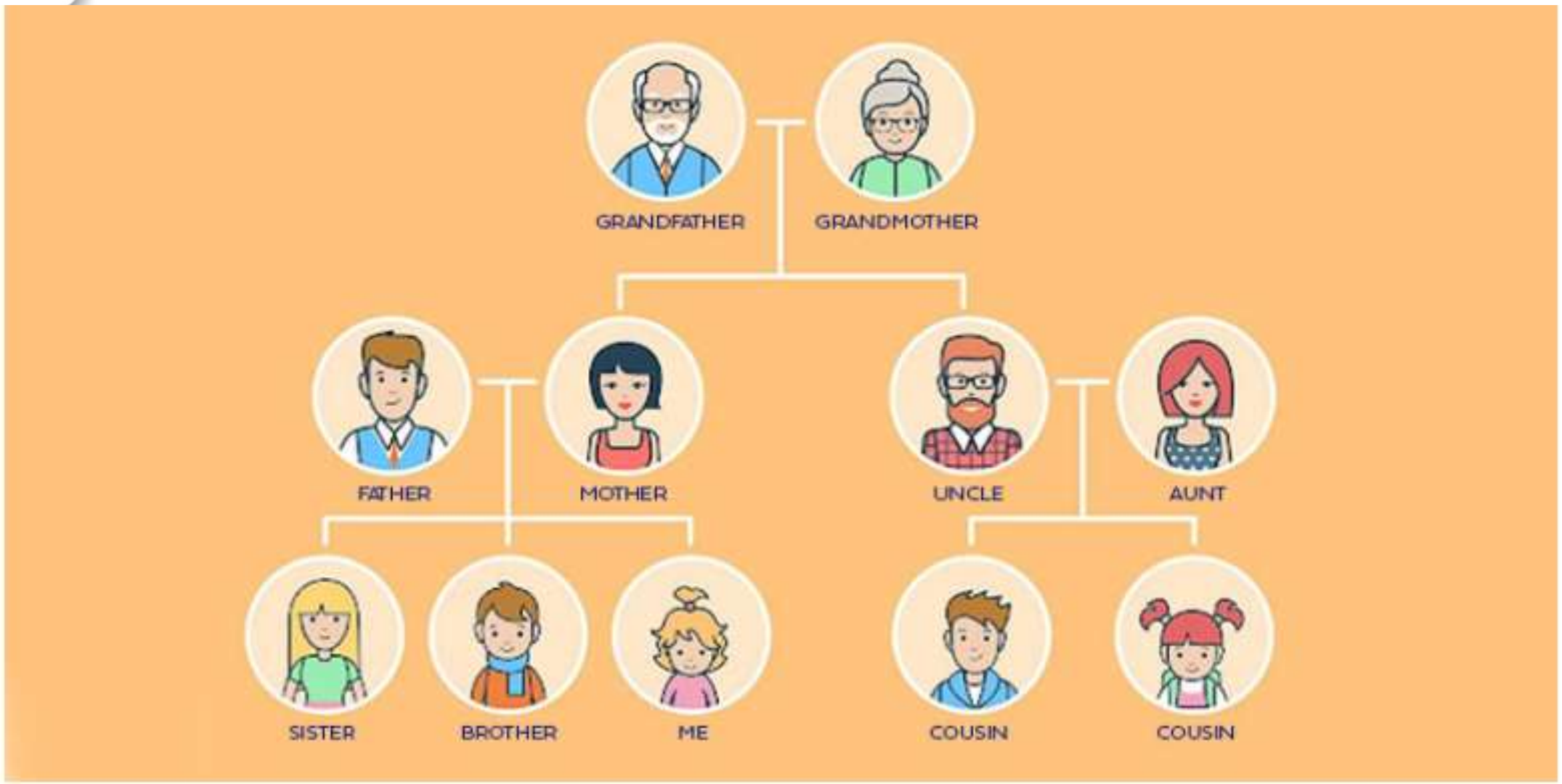
There are mainly three types of ADTs:



Non linear data structures

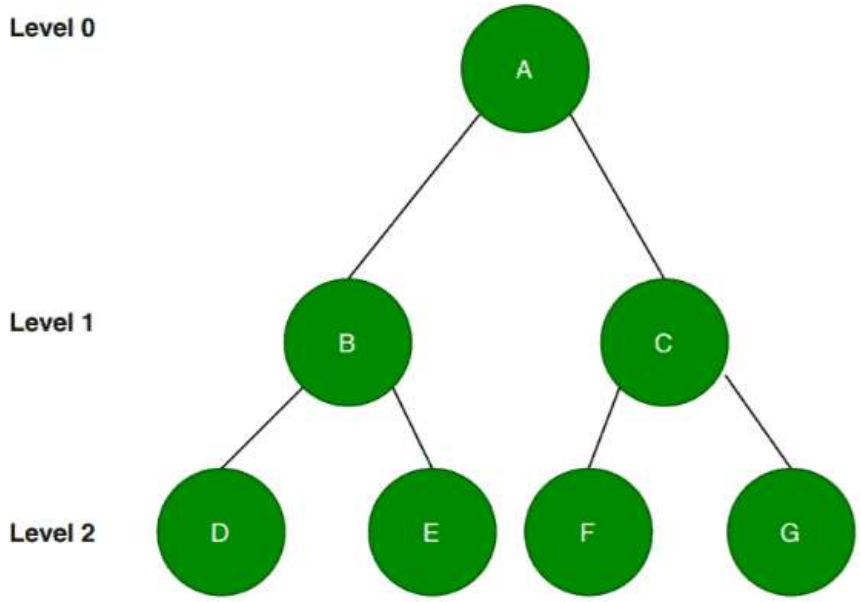
- Nonlinear data structures are those data structure in which data items are not arranged in a sequence.
- In a non-linear data structure, we can’t traverse all the elements in a single run only.
- Examples of Non-linear Data Structure are Tree and Graph.

Tree


GRANDFATHER
GRANDMOTHER
FATHER
MOTHER
UNCLE
AUNT
SISTER
BROTHER
ME
COUSIN
COUSIN

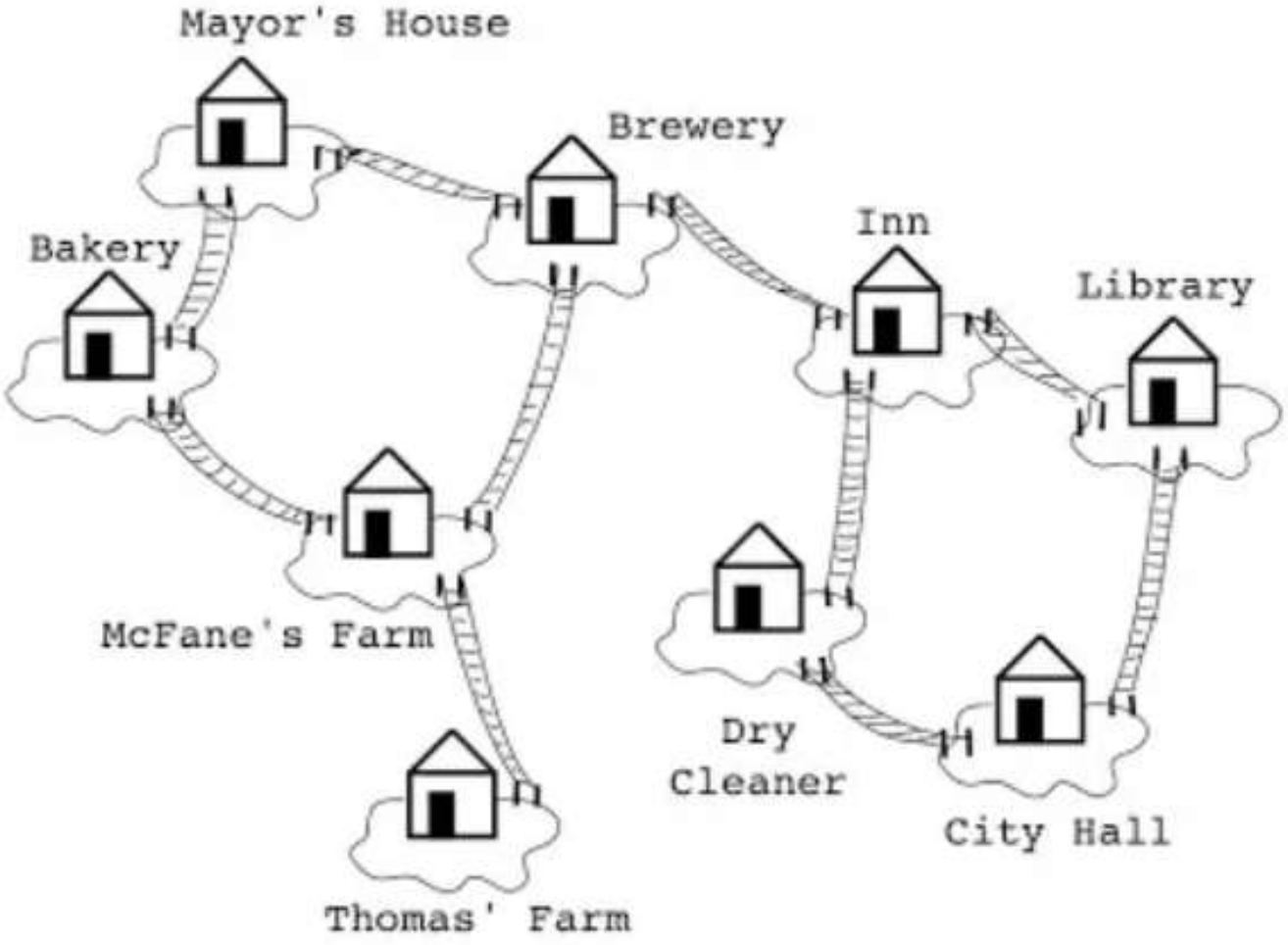
Tree

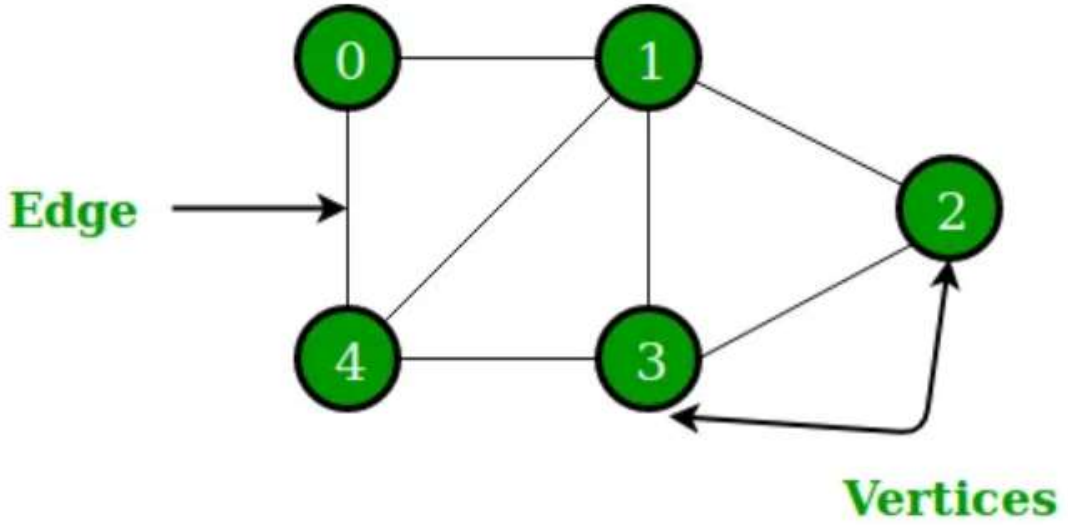
- A tree is a non linear data structure which consists of a collection of nodes arranged in a hierarchical order.
- One of the nodes is designated as the root node and the remaining nodes can be partitioned into disjoint sets such that each set is a sub-tree of the root.



Binary Tree

Graph


Mayor's House
Brewery
Bakery
Inn
Library
McFane's Farm
Dry
Cleaner
City Hall
Thomas' Farm


Edge
0
1
2
4
3
Vertices

Graph

- A Graph is a collection of ***vertices*** (also called *nodes*) and ***edges*** that connect these vertices.
- A graph is often viewed as a generalization of the tree structure, where instead of a purely parent-to-child relationship between tree nodes, any kind of complex relationship between the nodes can exist.

Linear Data Structure	Non-Linear Data Structure
Every item is related to its previous and next items.	Every item is attached with many other items.
Data is arranged in linear sequence.	Data is not arranged in sequence.
Data items can be traversed in a single run.	Data cannot be traversed in a single run.
Eg. Array, Stacks, linked list, queue.	Eg. tree, graph.
Implementation is easy.	Implementation is difficult.

Operations on Data Structures

- Design of efficient data structure must take operations to be performed on the data structures into account. The most commonly used operations on data structure are broadly categorized into following types

1. Create

- The create operation results in reserving memory for program elements. This can be done by declaration statement. Creation of data structure may take place either during compile-time or run-time. malloc() function of C language is used for creation.

2. Destroy

- Destroy operation destroys memory space allocated for specified data structure. free() function of C language is used to destroy data structure.

3. **Selection**

- Selection operation deals with accessing a particular data within a data structure.

4. **Updation**

- It updates or modifies the data in the data structure.

5. **Searching**

- It finds the presence of desired data item in the list of data items, it may also find the locations of all elements that satisfy certain conditions.

6. **Sorting**

- Sorting is a process of arranging all data items in a data structure in a particular order, say for example, either in ascending order or in descending order.

7.Merging

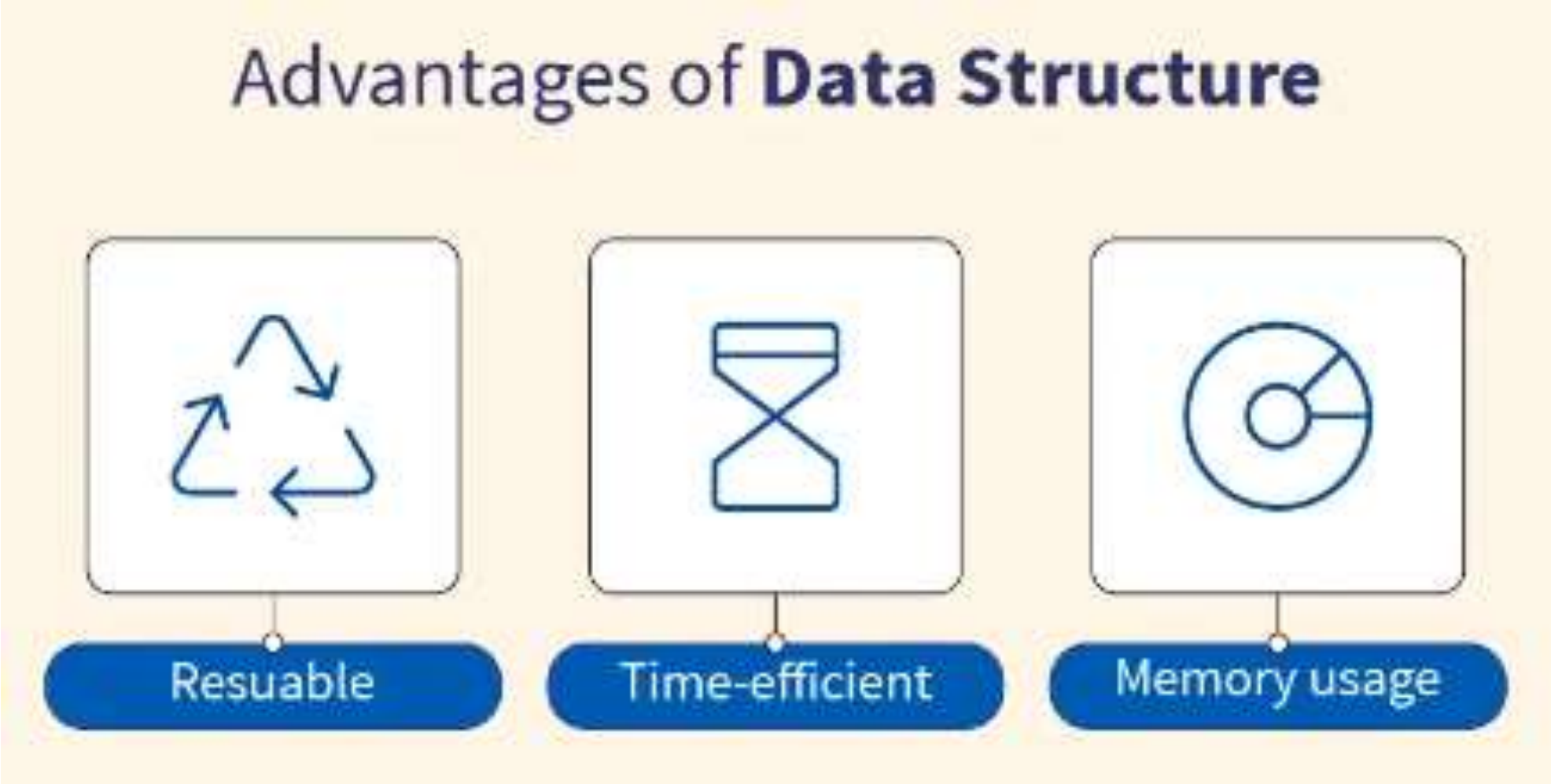
- Merging is a process of combining the data items of two different sorted list into a single sorted list.

8.Splitting

- Splitting is a process of partitioning single list to multiple list.

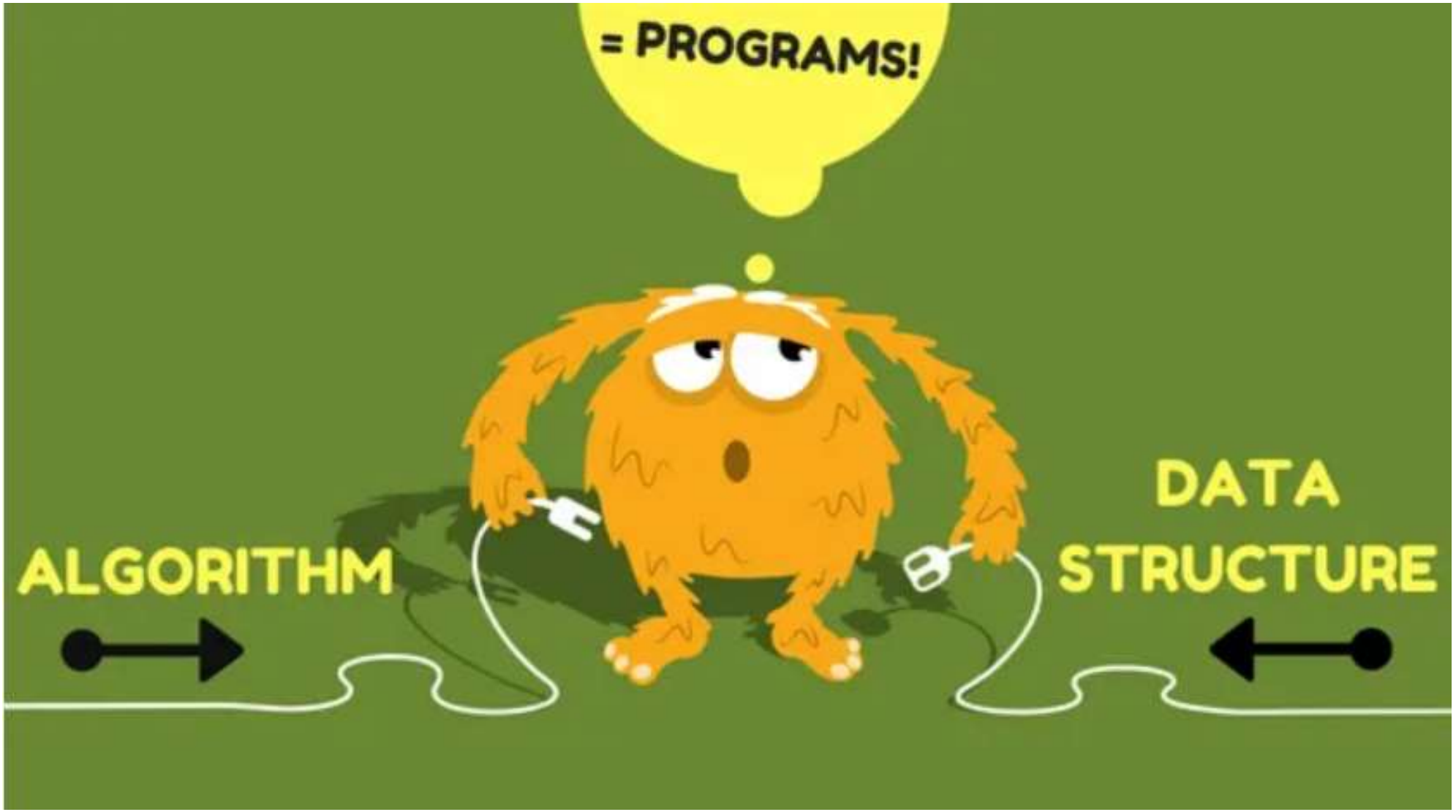
9.Traversal

- Traversal is a process of visiting each and every node of a list in systematic manner.


Advantages of Data Structure
Resuable
Time-efficient
Memory usage

Algorithms – Basics

- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.
- Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language


= PROGRAMS!
ALGORITHM
DATA
STRUCTURE

Time and space analysis of algorithms

- **Time Complexity** is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm.
- "Time" can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take.

- **Space complexity** is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm.
- We often speak of "extra" memory needed, not counting the memory needed to store the input itself. Again, we use natural (but fixed-length) units to measure this.
  - We can use bytes, but it's easier to use, say, number of integers used, number of fixed-sized structures, etc. In the end, the function we come up with will be independent of the actual number of bytes needed to represent the unit.
  - Space complexity is sometimes ignored because the space used is minimal and/or obvious, but sometimes it becomes as important an issue as time.

Execution Time Cases

- Worst Case − This is the scenario where a particular data structure operation takes maximum time it can take. If an operation's worst case time is ƒ(n) then this operation will not take more than ƒ(n) time, where ƒ(n) represents function of n.
- Average Case − This is the scenario depicting the average execution time of an operation of a data structure. If an operation takes ƒ(n) time in execution, then m operations will take mƒ(n) time.
- Best Case − This is the scenario depicting the least possible execution time of an operation of a data structure. If an operation takes ƒ(n) time in execution, then the actual operation may take time as the random number which would be maximum as ƒ(n).

Big O notation (O)

- It is defined as upper bound and upper bound on an algorithm is the most amount of time required (the worst case performance). Big O notation is used to describe the asymptotic upper bound.
- Mathematically, if **f(n)** describes the running time of an algorithm; **f(n)** is **O(g(n))** if there exist positive constant **C** and **n0** such that,

$$0 <= f(n) <= Cg(n) \text{ for all } n >= n0$$

- **n** = used to give upper bound a function. If a function is **O(n)**, it is automatically **O(n-square)** as well.

Big Omega notation (Ω)

- It is define as lower bound and lower bound on an algorithm is the least amount of time required (the most efficient way possible, in other words best case).
  Just like O notation provide an asymptotic upper bound, Ω notation provides asymptotic lower bound.
- Let **f(n)** define running time of an algorithm; **f(n)** is said to be **Ω(g (n))** if there exists positive constant **C** and **(n0)** such that

  $$0 <= Cg(n) <= f(n) \text{ for all } n >= n0$$

- **n** = used to given lower bound on a function
  If a function is **Ω(n-square)** it is automatically **Ω(n)** as well.

Big Theta notation (Θ)

- It is define as tightest bound and tightest bound is the best of all the worst case times that the algorithm can take.
- Let **f(n)** define running time of an algorithm. **f(n)** is said to be **Θ(g(n))** if **f(n)** is **O(g(n))** and **f(n)** is **Ω(g(n)).**
- Mathematically,

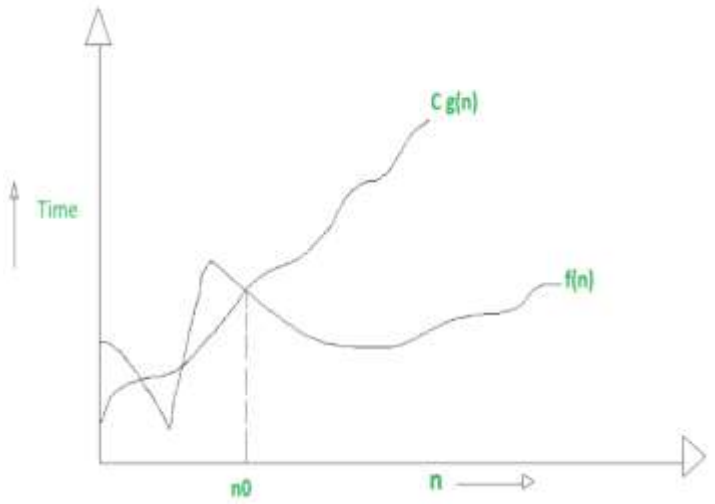
$$0 <= f(n) <= C1g(n) \text{ for } n >= n0$$
$$0 <= C2g(n) <= f(n) \text{ for } n >= n0$$

- Merging both the equation, will get ***:***

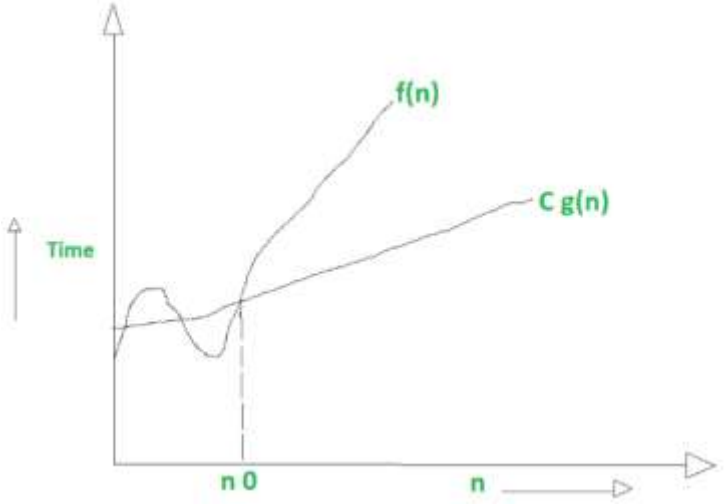
$$0 <= C2g(n) <= f(n) <= C1g(n) \text{ for } n >= n0$$

- The equation simply means there exist positive constants C1 and C2 such that f(n) is sandwich between C2 g(n) and C1g(n).

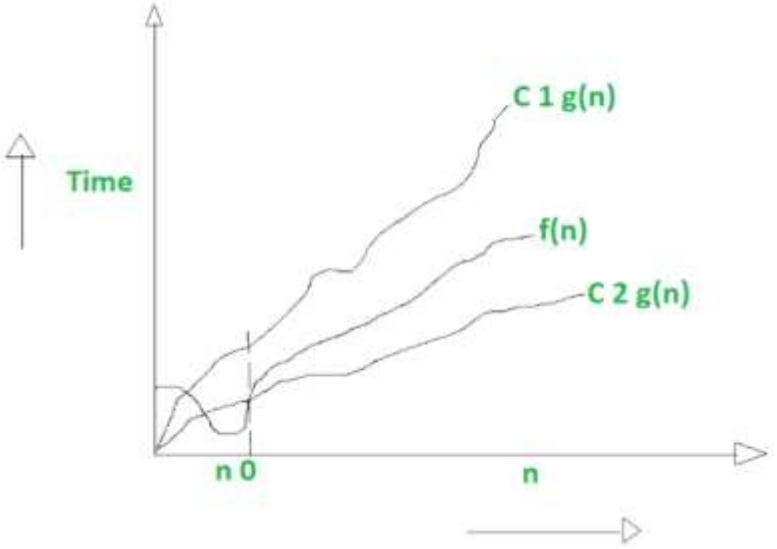
Graphical representation



Big O



Big Omega (Ω)



Big Theta (Θ)

DISCUSSION…

SVKM'S NMIMS
Deemed to be UNIVERSITY

THANK YOU