Unit 1: Data structure (07M)

- Introduction to data structure
- Need of data structure
- Abstract data type

Introduction to data structure:

- What is Data structure
- Why Data structure
- Classification
- Operations on Data structure
- ADT (Abstract Data Structure)

Data: Anything to give Information is called Data.
Ex. Student name, roll no. branch

Structure: Representation of data is called structure.
Ex. Stack, array, queue

Data structure: Data + Structure

- It is a way to store and organize data so that it can be used easily for the further process.
- It is a way of organizing all data items and relationship to each other.

What is Data structure

- The data structure is a way of storing and organizing data in a computer system. So that we can use the data quickly, which means the information is stored and held in such a way that it can be easily accessed later at any time.
- Data structures are used widely in almost every aspect of computer science,

There are two types of data structure:

1. Primitive data structure
2. Non-primitive data structure

Need of Data structure:

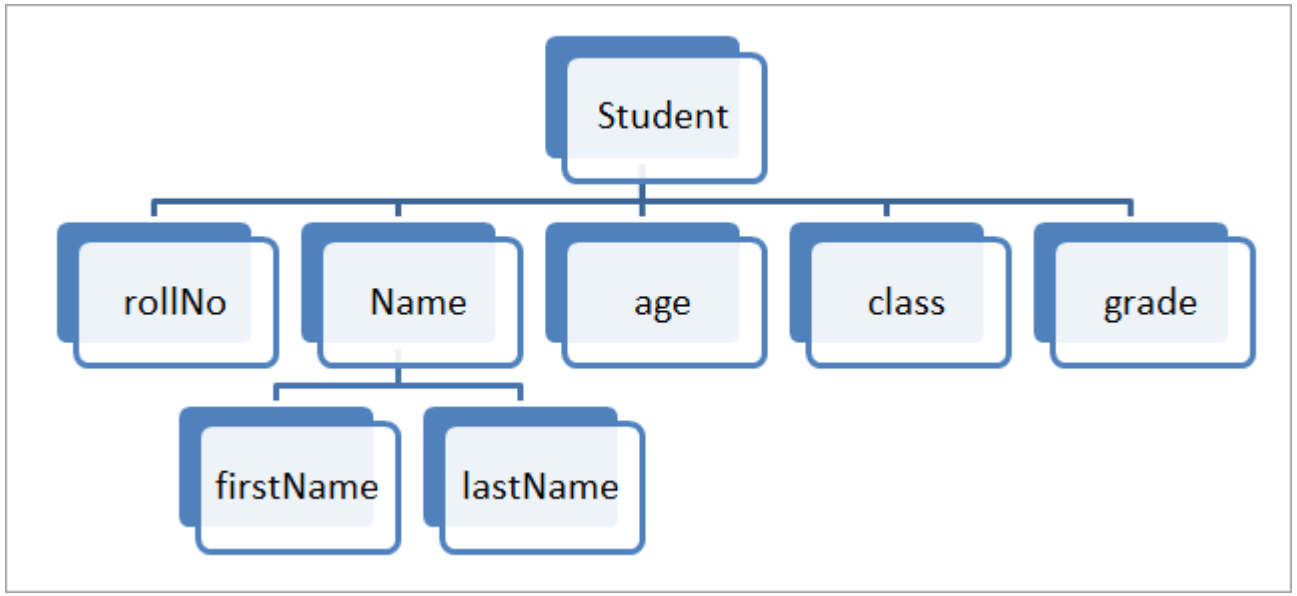
Organize data has major impact on performance

- To understand it, consider some real life examples such as
  - Queue at a ticket counter.
  - Dictionary
  - Queue in printer
  - Arrangement of books in library etc.
- Programming point of view

  Program

  Algorithm + Data structure
- Therefore choosing an appropriate Data structures improves the performance of a program or a software.

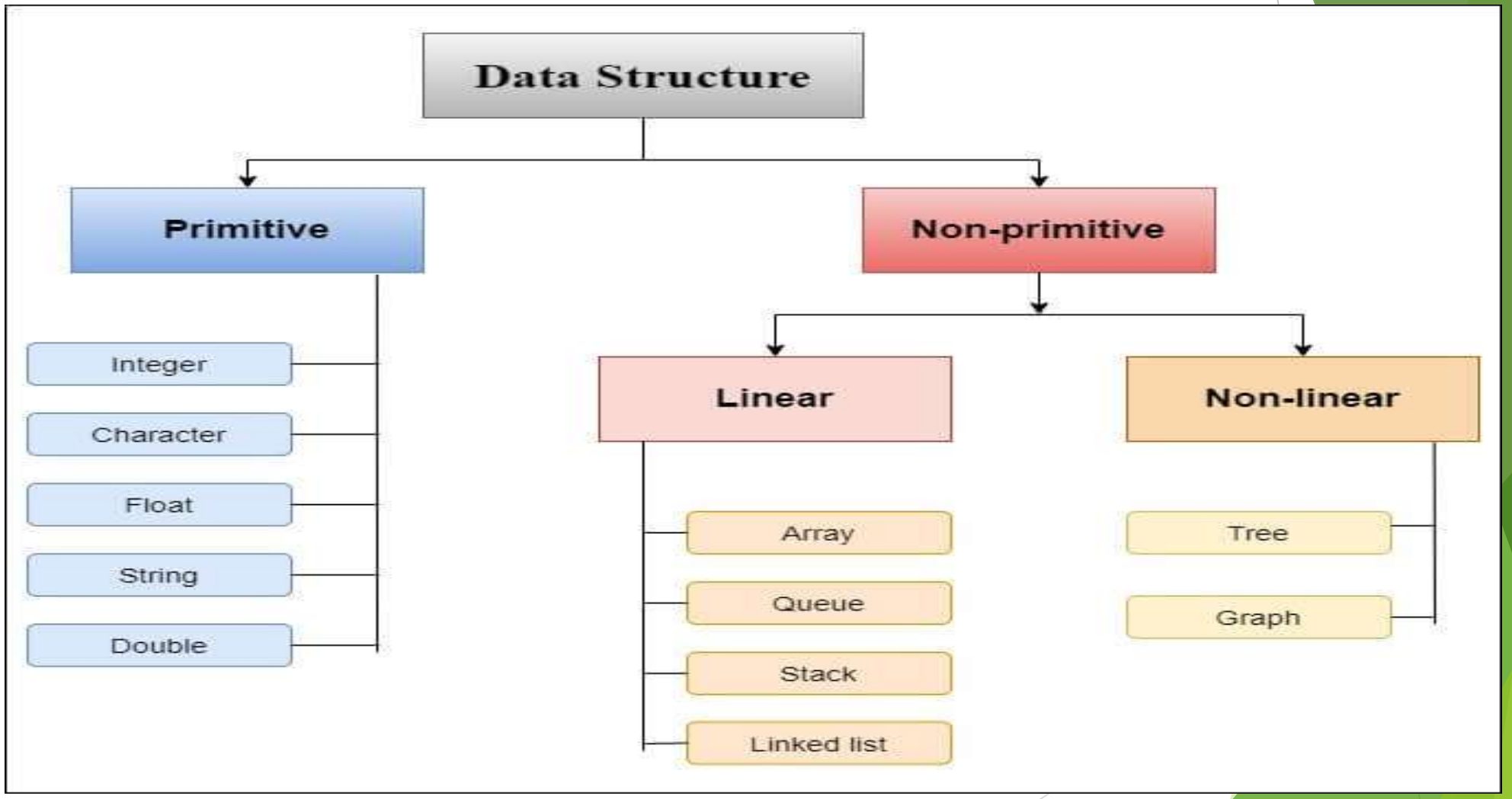
Need of Data Structure



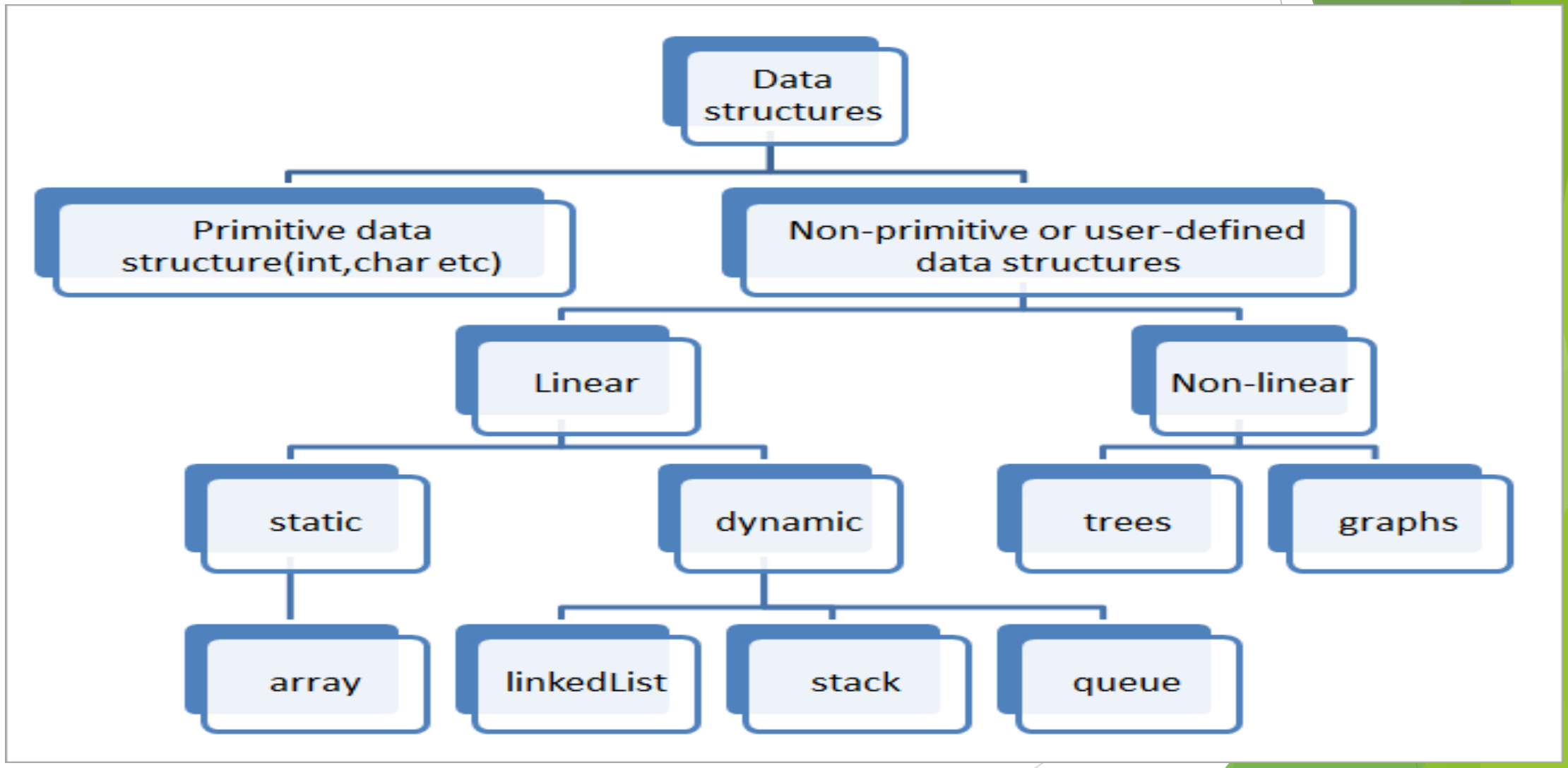
WHERE TO USE DATA STRUCTURE

- Computer networks: Computer networks use data structures such as graphs, tables, and trees **to store and route information**.
- Machine learning: Machine learning algorithms use data structures such as matrices, arrays, and trees to store and manipulate data, and to implement models.
- Data Analysis is essential as it **helps businesses understand their customers better, improves sales, improves customer targeting, reduces costs, and allows for the creation of better problem-solving strategies**.

Classification of Data Structure



Classification of Data structure



Static Data Structure

- Data structures that are of a fixed size are called **static data structures**.
- Memory is allocated at the compiler time for static data structures and user cannot change their size after being compiled but, we can change the data which is stored in them.
- The fixed size provides many benefits as well as a lot of drawbacks to static data structures.
- With the fixed memory allocation there is no need to worry about the overflow and underflow while inserting or deleting the element in/from the static data structures but it consumes a lot of memory and is not space-efficient. Ex. Array

Dynamic Data Structure

- Data structures that are of the dynamic size are called **dynamic data structures**.
- Memory is allocated at the run time for dynamic data structure and the size of the dynamic data structures varies at the run-time of the code. Also, both the size and the elements stored in the dynamic data structure can be changed at the run time of the code.
- The dynamic size provides many benefits as well as a lot of drawbacks of dynamic data structures. With the dynamic memory allocation, there is no memory loss occurs and we can allocate space equal to the required number of elements.
- Users must have to check and carefully insert or delete data in/from the dynamic data structure to be safe from overflow and underflow conditions.
- Linked lists and trees are common examples of dynamic data structures and a list of examples for dynamic data structures is never-ending.

Difference between Static and Dynamic Data Structure

Static Data Structure

- Memory is allocated at Compile time
- Size is Fixed and can not be Modified
- Memory utilization is inefficient due to fixed size
- Access time is faster as it is Fixed
- Ex. Array

Dynamic Data Structure

- Memory is allocated at Run Time
- Size can be modified during run time
- Memory utilization is efficient as memory can be utilised
- Access time may be slower due to indexing and pointer usage
- Ex. Tree , Graph Stack ,Queue

Classification of Data structures

Primitive Data structures:

- These are pre defined, built in data types.
- Machine dependent i.e, on 16 bit machine int data type is 2 byte in size while on 32 bit or 64 bit it is of 4 bytes.

Non primitive Data Structures:

- User defined Data types.
- Also called Derived data types as they are derived from Primitive Data structures.

Primitive data structure

The primitive data structure can be directly controlled by computer commands. That means it is defined by the system and compiler.

There are the following types of Primitive data structure, that is shown in the figure:

1. Integer
2. Character
3. Double
4. Float
5. String

Integer: In the integer, it includes all mathematical values, but it is not include decimal value. It is represented by the int keyword in the program.

Character: The character is used to define a single alphabet in the programming language. It is represented by the char keyword in the program.

String: The group of the character is called a string. It is represented by the string keyword in the program. The string is written with a double quotation mark ("-"). For example: "My name is Bob".

Float and Double: Float and double is used for real value.

Primitive Data Structures

DATA STRUCTURE	DESCRIPTION	EXAMPLE
INTEGER	Used to represent a number without decimal point.	1 , 3, 34, 6 , 45 , 1000
FLOAT	Used to represent a number with decimal point.	4.5 , 1.8 , 5.987 ,
CHARACTER	Used to represent single character.	A , B , G , H
BOOLEAN	Used to represent logical values either true or false.	TRUE , FALSE

www.learncomputerscienceonline.com

Non-primitive data structure

- The Non-primitive data structure cannot be directly controlled by computer commands. The Non-primitive data structure is derived from the primitive data structure.
- There are two types of non-primitive data structure:
- 1. Linear data structure
- 2. Non-linear data structure

Linear data structure:

- Linear data structures are those data structures in which data elements are stored and organized in a linear manner, in which one data element is connected to another as a line.
- **For example**: array, linked list, queue, stack.

Non-linear data structure

- Non-linear data structures are those data structures in which data elements are not organized in a linear manner.
- In this, a data element can be associated with any other data element.
- **For example**: tree and graph.

Data Structure Operations

Various data structure operations are used to process the data in a data structure which is as follows:

- **Traversing:** Visiting each element of the data structure only once is called traversing.
- **Searching:** Finding the location of the record or data in the data structure is called searching.
- **Inserting:** Adding the same type of element to the data structure is called insertion. An element can be added anywhere in the data structure.
- **Deleting:** Removing an element from a data structure is called Deletion. An element can also be removed from anywhere in a data structure.
- **Sorting:** Arranging a record in a logical order in the data structure is called sorting.

- **Merging:** In the data structure, the record is stored in many different files. Adding these different files to a single file is called merging.

Abstract Data type

- Present only the simple view of any Object but hide the implementation details.

ABSTRACTION

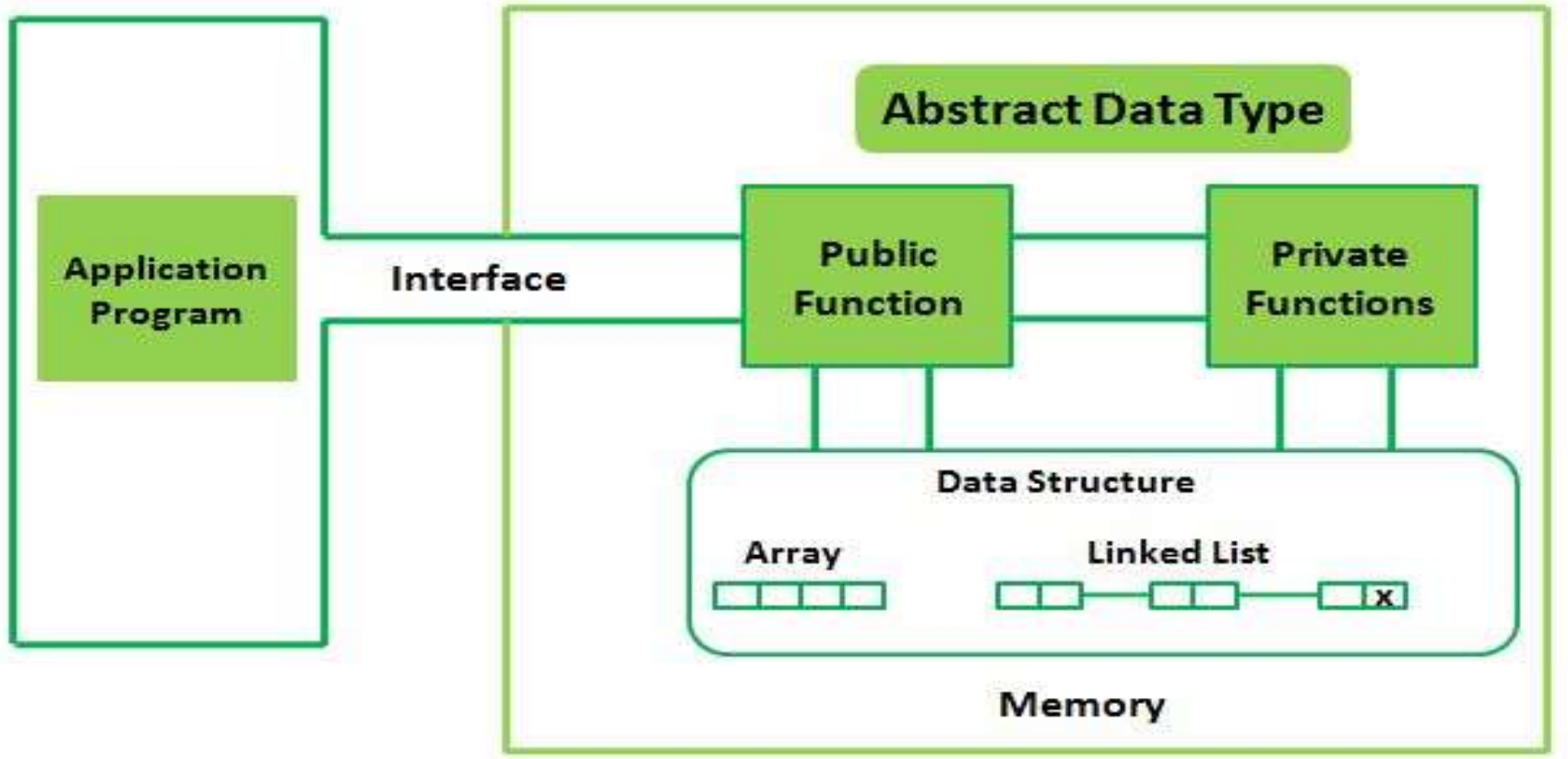
- It focuses on WHAT rather than HOW.

Ex. TV Remote
To start the TV , we simply press the ON button without knowing the operation behind this

Abstract Data type

- Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of values and a set of operations.
- The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented.
- It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called “abstract” because it gives an implementation-independent view.
- The process of providing only the essentials and hiding the details is known as abstraction.
- So a user only needs to know what a data type can do, but not how it will be implemented. Think of ADT as a black box which hides the inner structure and design of the data type. Now we’ll define three ADTs namely List ADT, Stack ADT, Queue ADT

Abstract data type with diagram



Classification of Data structure

Linear Data structures:

- Elements are arranged in a sequential manner.

E1---- E2----E3----E4----E5----------En

- Except the first and last element, every element is connected with processor and successor.
- Easy to Implement.

  Ex. Arrays, Stack, Queue etc.

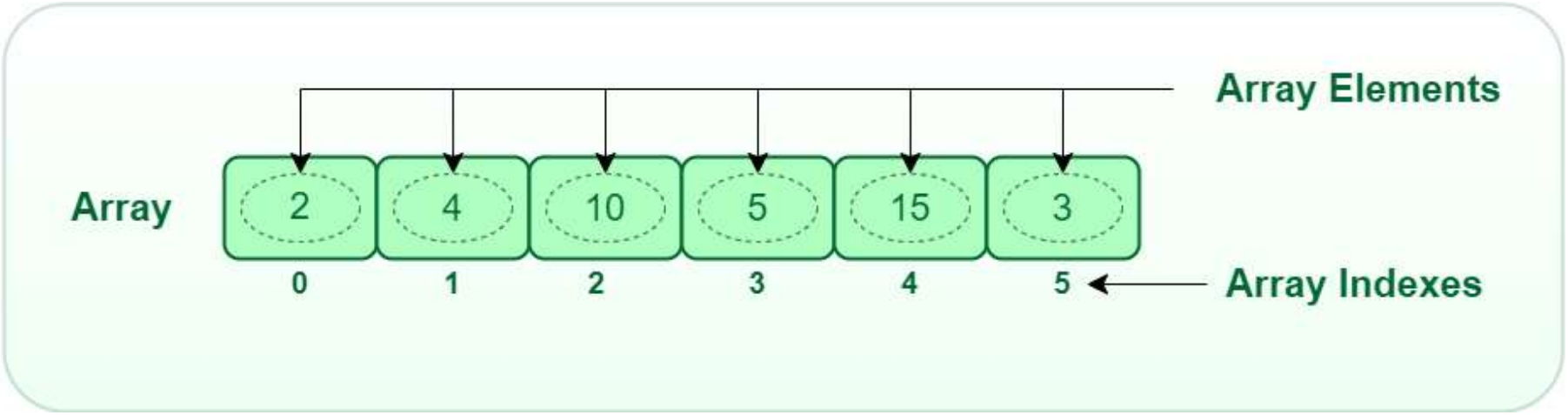
Array Data structure:

- An array is a collection of same type of elements which stored contiguously in memory. So it is a Linear Data Structure.

  Ex. Int A[4]

- An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array
- Also called Subscripted variable.
- Application are searching, sorting, implement matrics .

Structure of Array

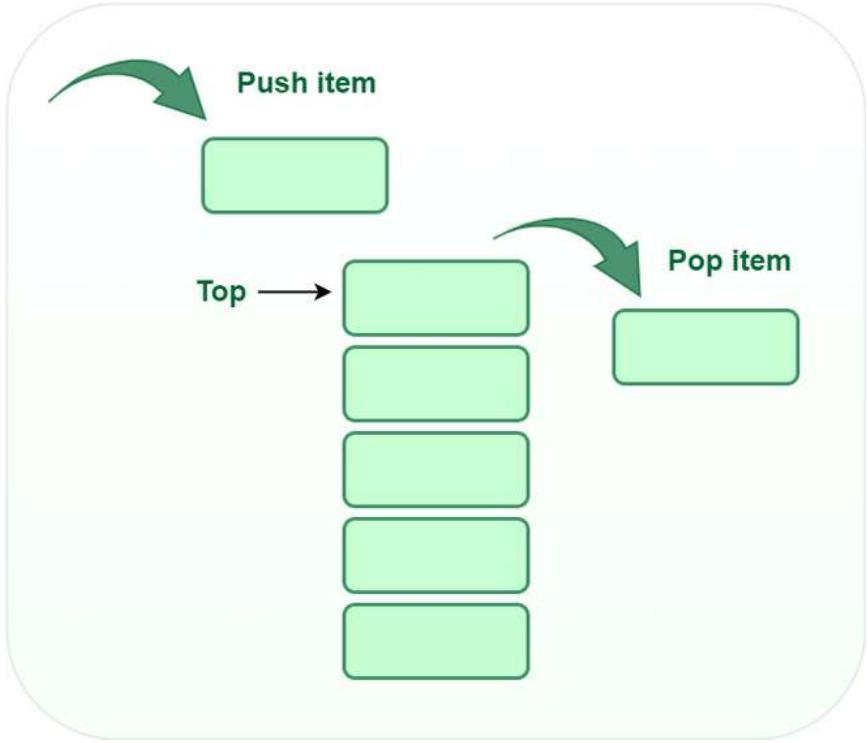


x[6] = {19, 10, 8};

Classification of Data Structures

Stack Data Structure:

- Stack is a linear Data Structure.
- Elements are arranged and deleted from one end, call TOP of the stack.



- Stack is a linear data structure that follows a particular order in which the operations are performed.
- The order may be LIFO(Last In First Out) or FILO(First In Last Out). LIFO implies that the element that is inserted last, comes out first and FILO implies that the element that is inserted first, comes out last.
  Consider an example of plates stacked over one another in the canteen.

The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO(Last In First Out)/FILO(First In Last Out) order.

The Basic operations performed on Stack are as follows:

- **PUSH** : The process of adding new element to top of the stack is called PUSH Operation. When new element is inserting at the TOP after every PUSH operation that TOP is incremented by one. In case the array is full and no element can be accommodated, it is called STACK FULL condition. This condition is called Stack overflow condition.

- **POP** : The process of deleting an element from the TOP of stack is called POP Operation. After every POP operation stack is decremented by one. If there is no element in stack and POP is performed then this will results into STACK UNDERFLOW condition.

- **PEEP** : If one is interested only about an information stored at some location in a stack then PEEP operation is required. In short we can say extract any position information from the stack.

- **UPDATE**: Update information is required when the content of some location in a stack is to be changed.

- **MAXSIZE**: This term is not a standard one , we use this term to refer the maximum size of the stack.

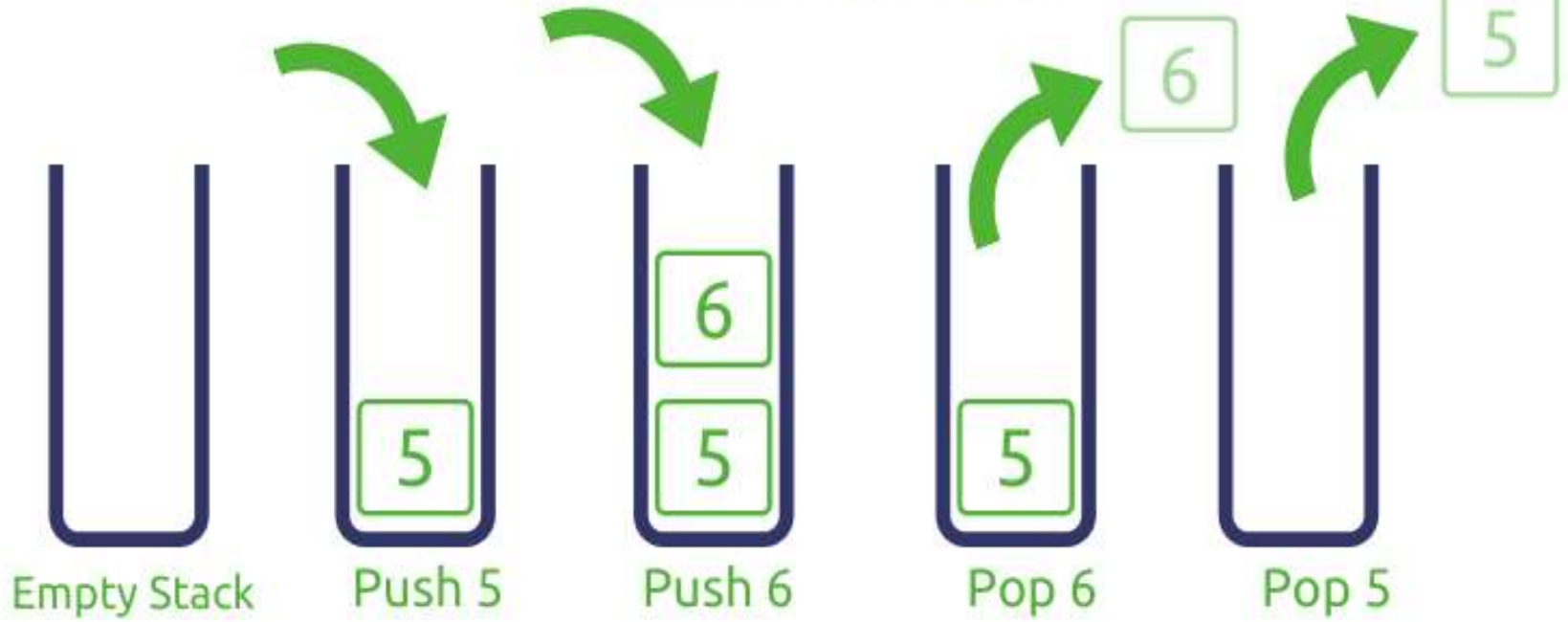
Types of Stack

- Register Stack :This type of stack is also a memory element present in the memory unit and can handle a small amount of data only. The height of the register stack is always limited as the size of the register stack is very small compared to the memory.

- Memory Stack: This type of stack can handle a large amount of memory data. The height of the memory stack is flexible as it occupies a large amount of memory data.

Stack

Last in first Out



Applications are

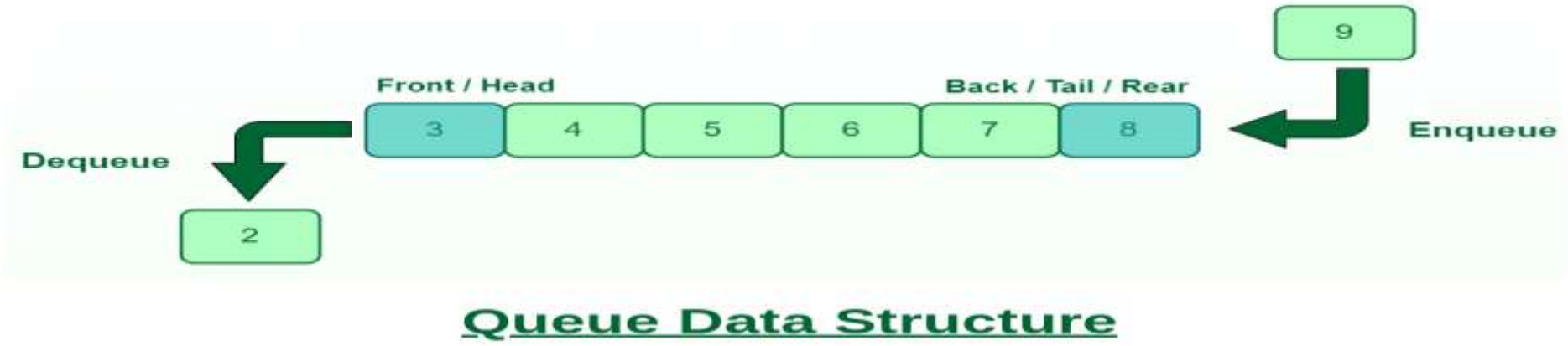
- Function call.
- Infix and post fix conversion
- Evaluation of postfix expression, in DFS Graph Algorithm

etc…..

Queue Data structure:

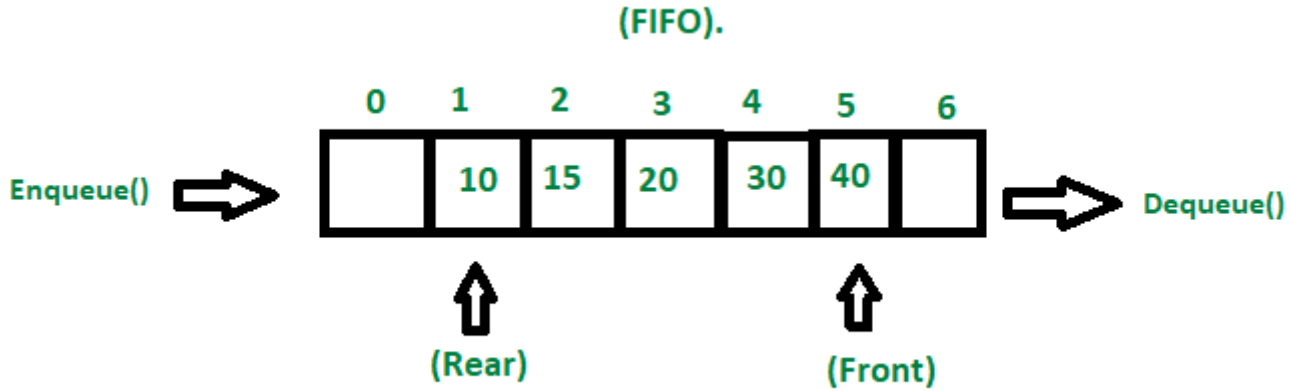
- Queue is a linear data structure.
- Elements are inserted at one end called Rear and deleted from other end which is called Front.
- A Queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order.

.



Queue Data Structure

- A Queue is like a line waiting to purchase tickets, where the first person in line is the first person served. (i.e. First come first serve).
- Position of the entry in a queue ready to be served, that is, the first entry that will be removed from the queue, is called the **front** of the queue(sometimes, **head** of the queue), similarly, the position of the last entry in the queue, that is, the one most recently added, is called the **rear** (or the **tail**) of the queue. See the below figure.



Characteristics of Queue:

- Queue can handle multiple data.
- We can access both ends.
- They are fast and flexible.

Queue Representation:

- Like stacks, Queues can also be represented in an array: In this representation, the Queue is implemented using the array. Variables used in this case are
- **Queue:** the name of the array storing queue elements.
- **Front**: the index where the first element is stored in the array representing the queue.
- **Rear:** the index where the last element is stored in an array representing the queue.

Queue Data Structure:

Queue can be implemented using arrays as well as using linked list
Some real life examples are queue at ticket counter,vehicles at toll plaza etc

Applications are

- CPU scheduling and Disk scheduling algorithms in operating system,
- Interrupt handling.
- Job queue at printer
- BFS GRAPH ALGORITHM

Compare between Stack and Queue

Stack

- **Stack follows LIFO Operation i.e;**
- **Insertion and deletion takes place on same end**
- **Stack perform two basic operations PUSH**

  POP
- **In Stack PUSH is used to insert the element in the stack.**
- **POP is used to delete**

Queue

- Queue follows FIFO Operation i.e; First in first out
- Insertion and deletion takes place on opposite end
- Queue perform two operations

  REAR

  FRONT
- In Queue REAR is used to insert the new element in the queue.

Compare between Stack and Queue

STACK	QUEUE
Stack doesn't have any type	Queue has various types like General Queue, Circular Queue, Double queue
In stack only one pointer is used.i.e; TOP of the Stack	In Queue two pointer are used at two different ends . REAR end and FRONT End.

Linked List data structure

- Linked list is a linear Data structures which stored non contiguously in memory.
- It is a collection of nodes where each node has two fields:

  INFO field: It actually stores value or set of values.

  Link Field: It actually stores address to the next node in the list.

Linked List Data structure:

- It provides memory utilization when memory is not available contiguously.
- It is basically implemented using Dynamic memory allocation.

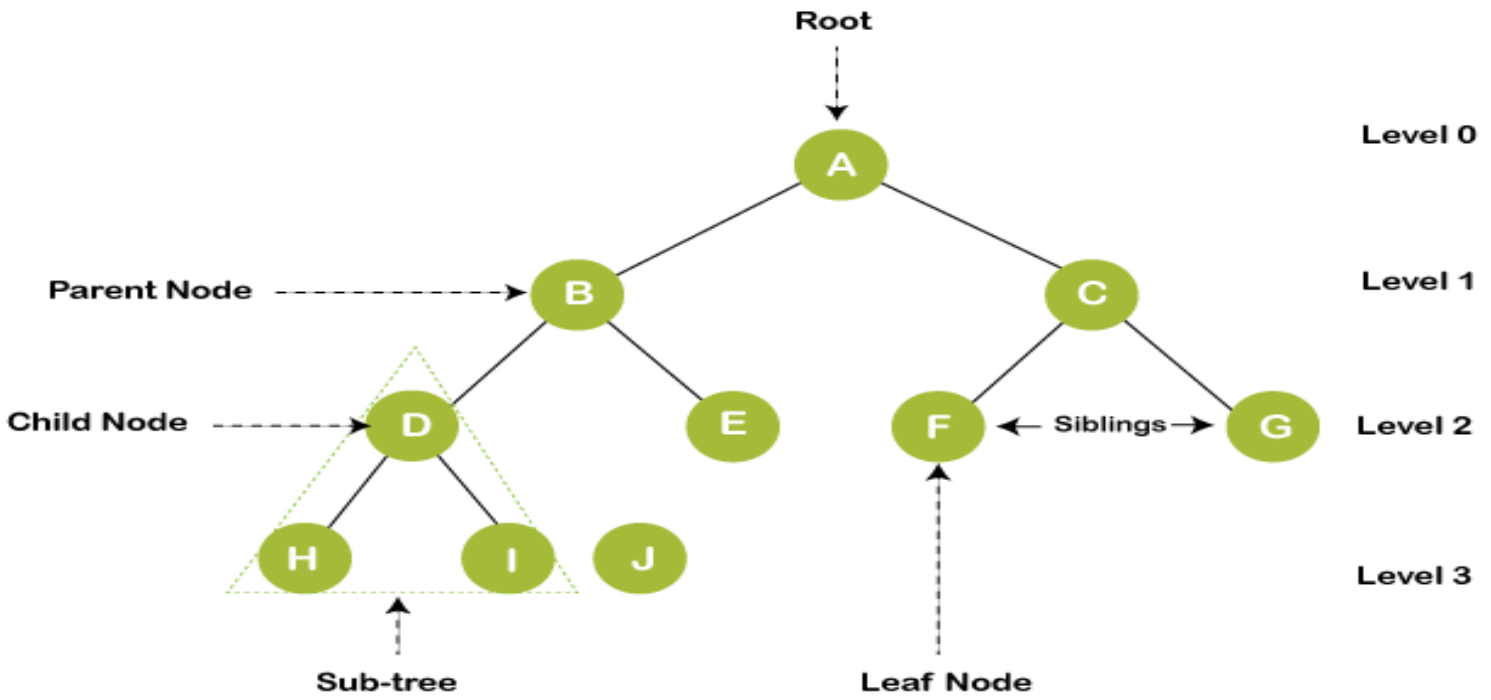
Applications are:

- Used to implement Stack ,Queue ,Tree , Graph
- Polynomial operations , Sparse matrices representation

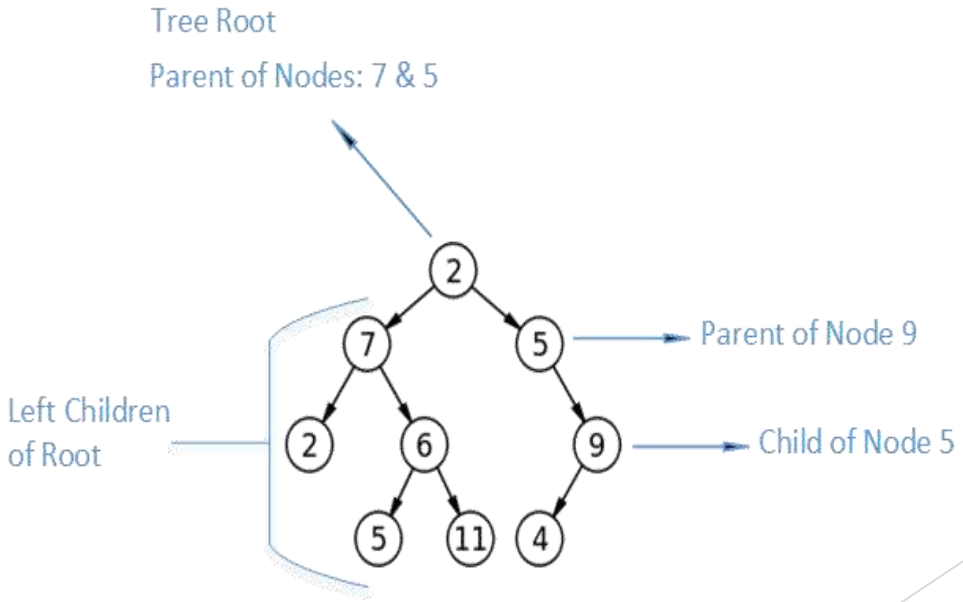
Non Linear Data structure

- Elements are arranged in Non sequential manner.
- Each element may be connected to two or more element
- It is a form of data structure where the data elements don't stay arranged linearly or sequentially. Since the data structure is non-linear, it does not involve a single level. Therefore, a user can't traverse all of its elements in a single run.
- Ex. Tree

Tree data structure:



- Trees are a collection of data formed of data elements called Nodes;
- Nodes are connected to each other by edges; each node element may or may not have child nodes.
- In each Tree collection, we have one root node, which is the very first node in our tree.
- If a node is connected to another node element, it then becomes a parent node and its connected node is its child node.



Characteristics of Tree data structure

- It is a nonlinear ,hierarchical data structure.
- It follows parent child relationship.
- Each node in tree can have more than one children but at most one parent node.
- A tree has following general properties:
- One node is distinguished as a **root**;
- Every node (exclude a root) is connected by a directed edge *from* exactly one other node; A direction is: *parent -> children*

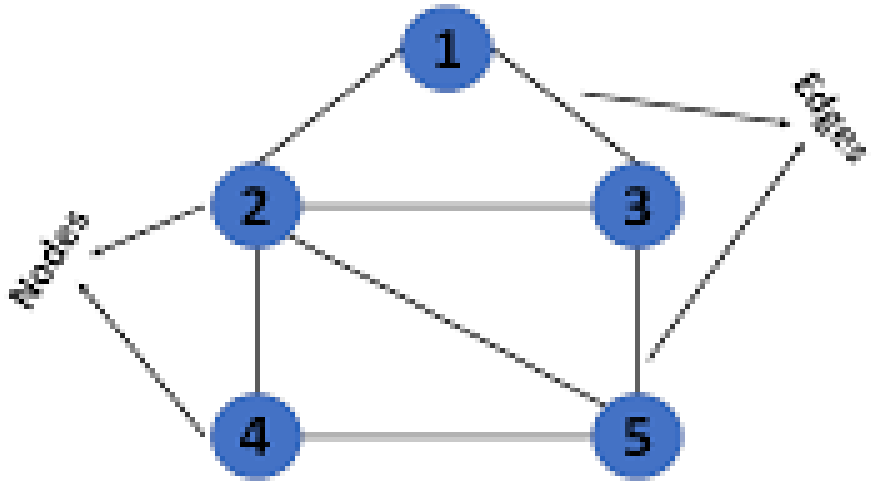
Applications of Tree data Structure

- File system on a computer
- Efficient searching using binary search tree
- Minimal spanning Tree implementation
- Heap sort using Heap Data structure.

Non linear data structure:
Graph

- It in a non linear, non hierarchical data structure.
- It consist of vertices or nodes and edges which connect these nodes.
- Each node may be connected to two or more elements.
- Graphs in data structures are **non-linear data structures made up of a finite number of nodes or vertices and the edges that connect them**.
- For example, it can represent a single user as nodes or vertices in a telephone network, while the link between them via telephone represents edges.

This graph has a set of vertices V= { 1,2,3,4,5} and a set of edges E= { (1,2),(1,3),(2,3),(2,4),(2,5),(3,5),(4,5}.



Applications of Graph

- Used to represent network like rail network, road network, telephone network.
- Social network such as whats app, facebook…….
- Google map, Navigation tools.
- World wide web.

Comparision of Linear and Non linear Data structure

Linear Data structure

- In a linear data structure, data elements are arranged in a linear order where each and every element is attached to its previous and next adjacent.
- In linear data structure, single level is involved.
- Its implementation is easy in comparison to non-linear data structure.

Non linear Data structure

- In a non-linear data structure, data elements are attached in hierarchically manner.
- Whereas in non-linear data structure, multiple levels are involved.
- While its implementation is complex in comparison to linear data structure.

Comparision of Linear and Non linear Data structure

Linear Data structure

- In linear data structure, data elements can be traversed in a single run only.
- In a linear data structure, memory is not utilized in an efficient way.
- Applications of linear data structures are mainly in application software development.
- Linear data structures are useful for simple data storage and manipulation.

Non linear Data Structure

- While in non-linear data structure, data elements can’t be traversed in a single run only.
- While in a non-linear data structure, memory is utilized in an efficient way.
- Applications of non-linear data structures are in Artificial Intelligence and image processing.
- Non-linear data structures are useful for representing complex relationships and data hierarchies, such as in social networks, file systems, or computer networks.

Compare Tree and Graph

Tree

- **It is a collection of nodes and edges.**
- **In a tree, each node can have at most one parent, except for the root node, which has no parent.**
- **Trees are commonly used to represent data that has a hierarchical structure, such as file systems, organization charts, and family trees.**

Graph

- **It is a collection of vertices/nodes and edges.**
- **In a graph, nodes can have any number of connections to other nodes.**
- **Graphs are commonly used to model complex systems or relationships, such as social networks, transportation networks, and computer networks.**

Tree

- **They are always directed**
- **There is a unique node called root(parent) node in trees.**
- **Application: For game trees, decision trees, the tree is used.**

Graph

- **They can be directed or undirected**
- **There is no unique node called root in graph.**
- **Application: For finding shortest path in networking graph is used.**

Algorithm

- An algorithm is a step-by-step procedure to solve a problem. A good algorithm should be optimized in terms of time and space. Different types of problems require different types of algorithmic techniques to be solved in the most optimized manner.

What makes a Good Algorithm?

•Input and output should be defined properly.

•Each step in the algorithm should be precisely clear and unambiguous.

•Algorithms should be the most effective among the other different ways to solve a problem.

•It shouldn't include computer code. Instead, it should be written in such a way that it can be used in a number of programming languages.

Use of the Algorithms:-

Algorithms play a crucial role in various fields and have many applications. Some of the key areas where algorithms are used include:

Computer Science: Algorithms form the basis of computer programming and are used to solve problems ranging from simple sorting and searching to complex tasks such as artificial intelligence and machine learning.

Mathematics: Algorithms are used to solve mathematical problems, such as finding the optimal solution to a system of linear equations or finding the shortest path in a graph.

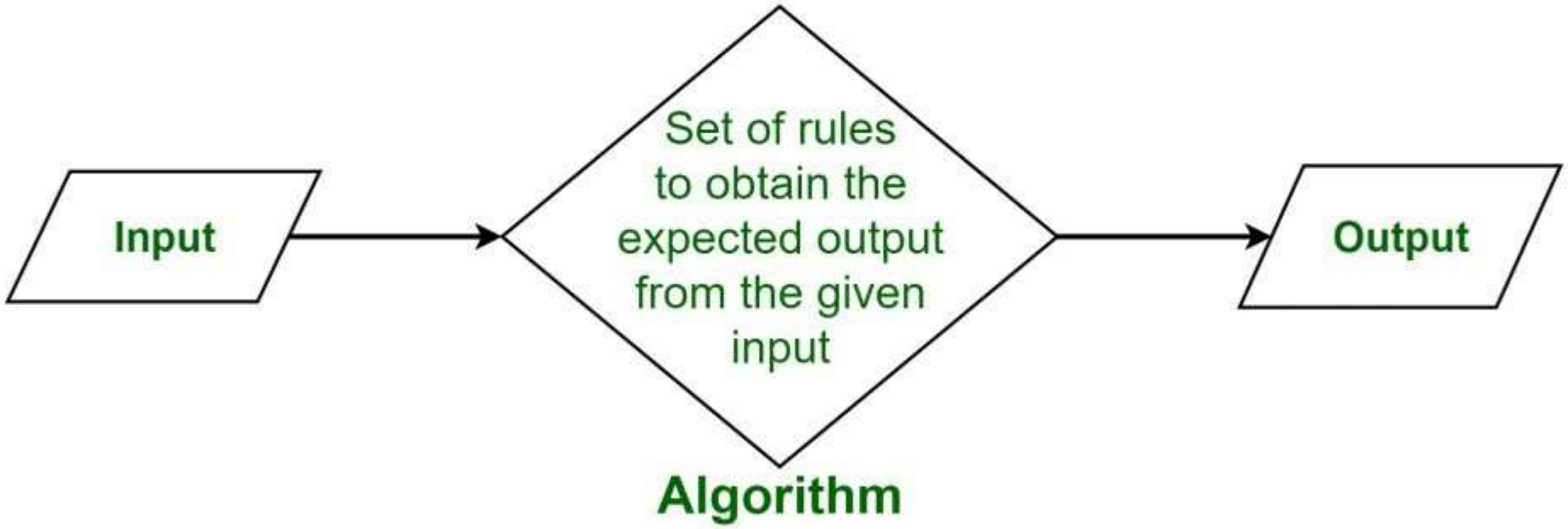
Operations Research: Algorithms are used to optimize and make decisions in fields such as transportation, logistics, and resource allocation.

Artificial Intelligence: Algorithms are the foundation of artificial intelligence and machine learning, and are used to develop intelligent systems that can perform tasks such as image recognition, natural language processing, and decision-making.

Data Science: Algorithms are used to analyze, process, and extract insights from large amounts of data in fields such as marketing, finance, and healthcare.

These are just a few examples of the many applications of algorithms. The use of algorithms is continually expanding as new technologies and fields emerge, making it a vital component of modern society.

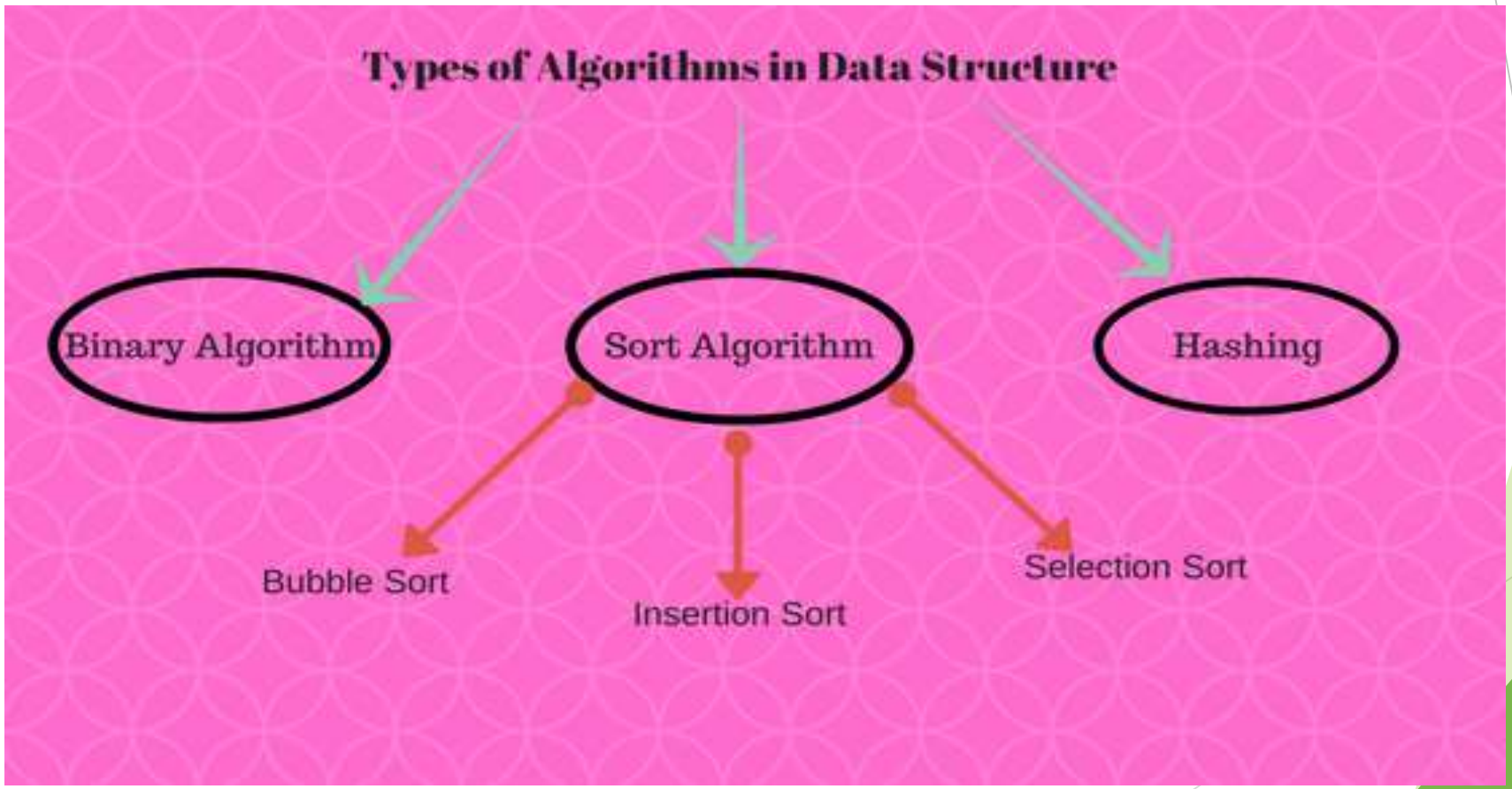
What is Algorithm?



Properties of Algorithm:

- It should terminate after a finite time.
- It should produce at least one output.
- It should take zero or more input.
- It should be deterministic means giving the same output for the same input case.
- Every step in the algorithm must be effective i.e. every step should do some work.

Types of Algorithm



Algorithm Efficiency

- Some algorithms perform better than others. We always prefer to select an efficient algorithm, hence metrics for assessing algorithm efficiency would be useful.
- The complexity of an algorithm is a function that describes the algorithm's efficiency in terms of the amount of data it must process.
- There are two basic complexity metrics of the efficiency of an algorithm:

Time complexity is a function that describes how long an algorithm takes in terms of the quantity of input it receives.

Space complexity is a function that describes how much memory (space) an algorithm requires to the quantity of input to the method.

Time complexity

The time complexity of an algorithm is the amount of time the algorithm takes to complete its process. Time complexity is calculated by calculating the number of steps performed by the algorithm to complete the execution.

Space complexity

- The space complexity of an algorithm is the amount of memory used by the algorithm.
- Space complexity includes two spaces: Auxiliary space and Input space. The auxiliary space is the temporary space or extra space used during execution by the algorithm. The space complexity of an algorithm is expressed by **Big O, Omega Ω , Theta θ** notation.
- Many algorithms have inputs that vary in memory size. In this case, the space complexity that is there depends on the size of the input.

Comparison of time complexity and space complexity

time complexity

- Calculates the time required
- Time is counted for all statements
- The size of the input data is the primary determinant.
- More crucial in terms of solution optimization

Space complexity

- Estimates the space memory required
- Memory space is counted for all variables, inputs, and outputs.
- Primarily determined by the auxiliary variable size
- More essential in terms of solution optimization

Searching Algorithm

Searching Algorithms

1. Linear Search

2. Binary Search

Searching In Data Structure:

- The process of finding the desired information from the set of items stored in the form of elements in the computer memory is referred to as 'searching in data structure’.
- These sets of items are in various forms, such as an array, tree, graph, or linked list.

Searching Methods

- Searching in the data structure can be done by implementing searching algorithms to check for or retrieve an element from any form of stored data structure.
- These algorithms are categorized based on their type of search operation, such as: Linear search or sequential search
  Binary search or interval search

Linear Search

- The linear search algorithm searches all elements in the array sequentially. Its best execution time is one, whereas the worst execution time is n, where n is the total number of items in the search array.
- It is the most simple search algorithm in data structure and checks each item in the set of elements until it matches the search element until the end of data collection. When data is unsorted, a linear search algorithm is preferred.
- Linear search has some complexities as given below:

Space Complexity

- Space complexity for linear search is O(n) as it does not use any extra space where n is the number of elements in an array.

•Time Complexity

*Best- case complexity = O(1) occurs when the search element is present at the first element in the search array.

*Worst- case complexity = O(n) occurs when the search element is not present in the set of elements or array.

*Average complexity = O(n) is referred to when the element is present somewhere in the search array.

Example,

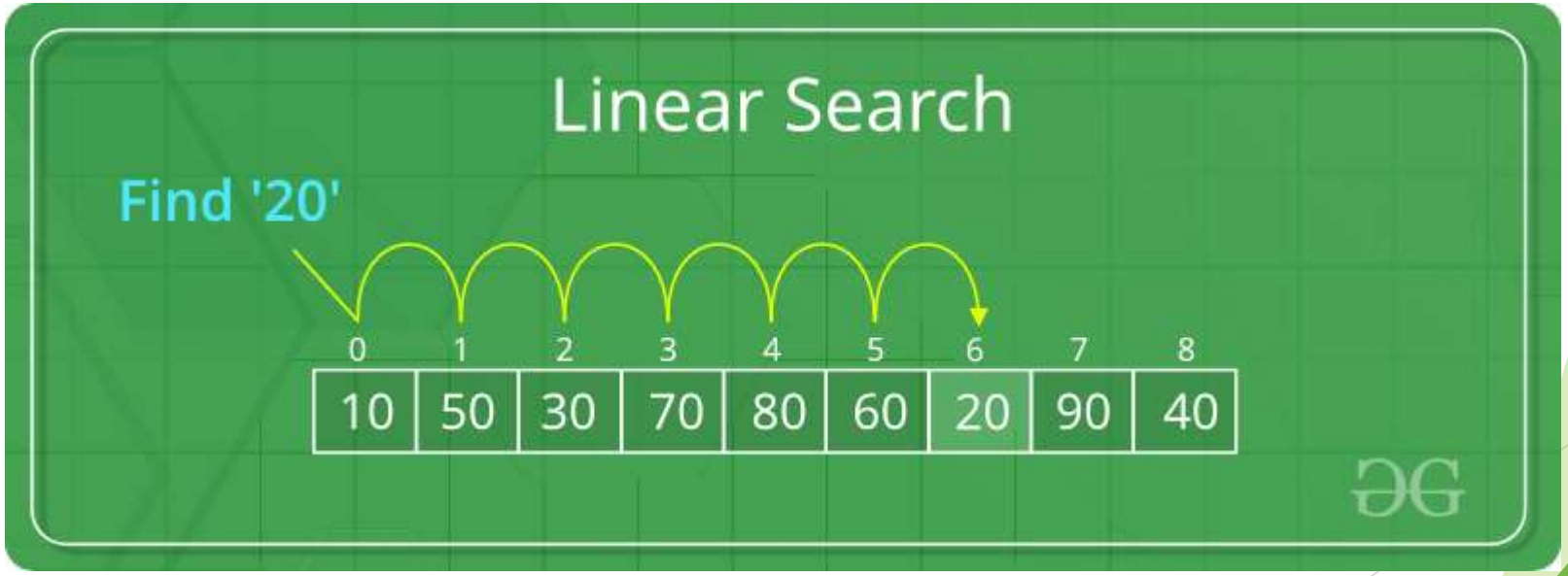
Let’s take an array of elements as given below:

45, 78, 12, 67, 08, 51, 39, 26

To find ‘51’ in an array of 8 elements given above, a linear search algorithm will check each element sequentially till its pointer points to 51 in the memory space. It takes O(6) time to find 51 in an array. To find 12, in the above array, it takes O(3), whereas, for 26, it requires O(8) time.

Linear search is implemented using following steps...

- **Step 1** - Read the search element from the user.
- **Step 2** - Compare the search element with the first element in the list.
- **Step 3** - If both are matched, then display "Given element is found!!!" and terminate the function
- **Step 4** - If both are not matched, then compare search element with the next element in the list.
- **Step 5** - Repeat steps 3 and 4 until search element is compared with last element in the list.
- **Step 6** - If last element in the list also doesn't match, then display "Element is not found!!!" and terminate the function.



Linear search Algorithm

Step 1 i=0
Step 2 if i > n; go to step 7
Step 3 if A [i] = x ; go to step 6
Step 4 i=i+1
step 5 go to step 2
step 6 Print element x is found at I
step 7 Print element is not found
step 8 Exit

List

Indexing 0 1 2 3 4 5

i =0 , x=? (no. what we have to find)

x=3

0 > 6---------false

A [i] = 3

A[0]=3

2=3 -------- false

i= i+1

i= 0+1=1

Now same procedure for i=1

1> 6---------- false

A[i]=3

A[1]=3

4=3 false

Now i=2

2>6----------false

A[i] =3

a[2] = 3

6= 3----------False

i=i+1=2+1=3

Now i=3

A[i]= 3

A[3]=3

3=3 ---------- TRUE

Print {found the element in the list at location 3}

Terminate the function

Exit

```c
// C implementation of the approach
#include <stdio.h>

// Linearly search x in arr[].
// If x is present then return the index,
// otherwise return -1
int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++) {
        if (arr[i] == x)
            return i;
    }
    return -1;
}
```

Binary search

- In computer science, binary search, also known as half-interval search, logarithmic search, or binary chop, is a search algorithm that finds the position of a target value within a sorted array.
- Binary search compares the target value to the middle element of the array.
- This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.

Binary searching

- **Binary Search Algorithm:**

The basic steps to perform Binary Search are:

- Sort the array in ascending order.
- Find the middle element of array.
- Compare the middle element with an item

There are three cases:

(a) If it is desired element then search is complete.

(b) If it is less than desired element then search only first half of the array.

(c) If it is greater than the desired element, search in the second half of the array.

Algorithm for binary search

step 1: Set beg= LB , End= UB and
mid= int ((beg + end)/2)

Step 2: Repeat step 3 and 4 while
beg<= end and a[mid] = item

Step 3: if item< a[mid] then
set end= mid-1
else
set beg= mid+1

Step 4: set mid= int((beg + end)/2)
go to step 2

Step 5: if a[mid]= item then
set location=mid
else
set location=null

Consider an example

Sort it first

Search element 8 in the array

beg=0, end=4, mid= int((0+4)/2)= int(4/2)= 2

0<4

Now if a[mid]= item;

a[2]=8;

5=8 ---------- False condition

else

beg= mid+1 = 2+1=3

beg=3 ; end=4

mid=int((3+4)/2)= 7/2= 3

Now mid=3

a[mid]= item (checks)

a[3] = 8

8=8

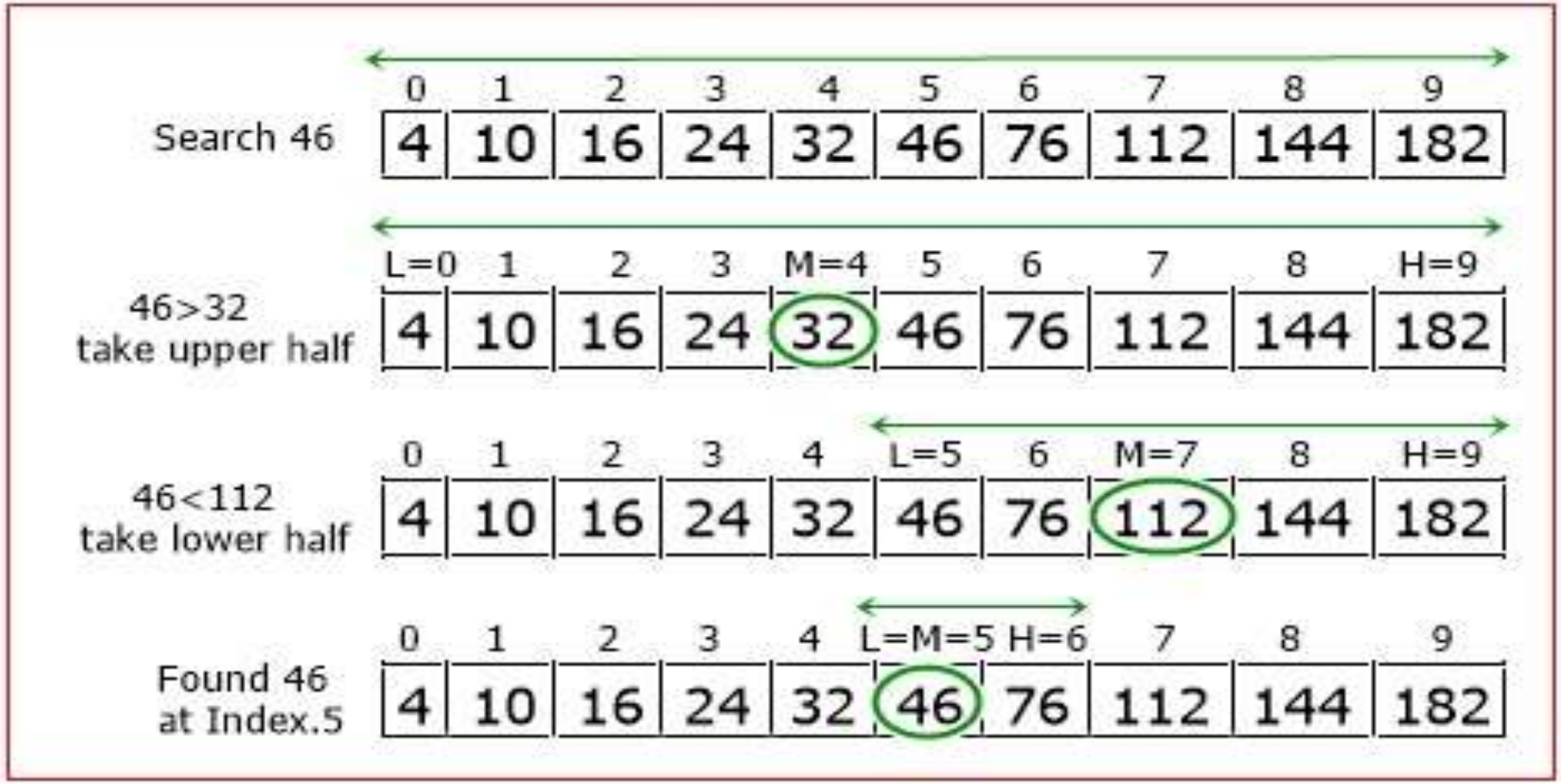
Goes to step no 5

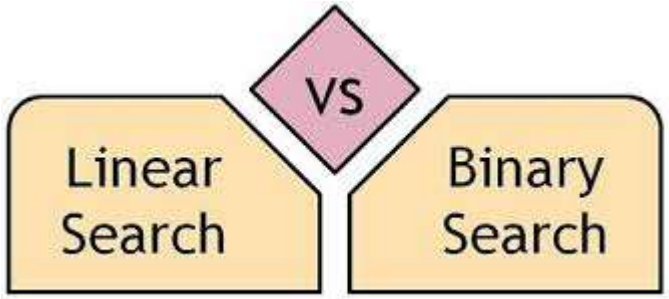
a[mid] =8

location of desired element is 3

Terminate the function

Binary Search





Linear Search	Binary Search
In linear search input data need not to be in sorted.	In binary search input data need to be in sorted order.
It is also called sequential search.	It is also called half-interval search.
The time complexity of linear search O(n).	The time complexity of binary search O(log n).
Multidimensional array can be used.	Only single dimensional array is used.
Linear search performs equality comparisons	Binary search performs ordering comparisons
It is less complex.	It is more complex.

Abstract Data Types

Ex. Stack Abstract Data Type-

Stack follows the LIFO concepts

Operations to be performed are:

- PUSH (data) : Insert an element at the top
- POP () : Remove an element from the top
- PEEK() : view the element which present at the top
- Is empty () : To check, whether stack is empty or not?

List ADT

- The data is generally stored in key sequence in a list which has a head structure consisting of *count*, *pointers* and *address of compare function* needed to compare the data in the list.
- The data node contains the *pointer* to a data structure and a *self-referential pointer* which points to the next node in the list.

The **List ADT Functions** is given below:

- get() – Return an element from the list at any given position.
- insert() – Insert an element at any position of the list.
- remove() – Remove the first occurrence of any element from a non-empty list.
- removeAt() – Remove the element at a specified location from a non-empty list.
- replace() – Replace an element at any position by another element.
- size() – Return the number of elements in the list.
- isEmpty() – Return true if the list is empty, otherwise return false.
- isFull() – Return true if the list is full, otherwise return false.

•**Stack ADT**In Stack ADT Implementation instead of data being stored in each node, the pointer to data is stored.
•The program allocates memory for the *data* and *address* is passed to the stack ADT.
•The head node and the data nodes are encapsulated in the ADT. The calling function can only see the pointer to the stack.
•The stack head structure also contains a pointer to *top* and *count* of number of entries currently in stack.
•push() – Insert an element at one end of the stack called top.
•pop() – Remove and return the element at the top of the stack, if it is not empty.
•peek() – Return the element at the top of the stack without removing it, if the stack is not empty.
•size() – Return the number of elements in the stack.
•isEmpty() – Return true if the stack is empty, otherwise return false.
•isFull() – Return true if the stack is full, otherwise return false.

1.Queue ADT

1. The queue abstract data type (ADT) follows the basic design of the stack abstract data type.
2. Each node contains a void pointer to the *data* and the *link pointer* to the next element in the queue. The program's responsibility is to allocate memory for storing the data.
3. enqueue() – Insert an element at the end of the queue.
4. dequeue() – Remove and return the first element of the queue, if the queue is not empty.
5. peek() – Return the element of the queue without removing it, if the queue is not empty.
6. size() – Return the number of elements in the queue.
7. isEmpty() – Return true if the queue is empty, otherwise return false.
8. isFull() – Return true if the queue is full, otherwise return false.

Features of ADT:

•**Abstraction:** The user does not need to know the implementation of the data structure only essentials are provided.

•**Better Conceptualization:** ADT gives us a better conceptualization of the real world.

•**Robust:** The program is robust and has the ability to catch errors.