Unit – 01 : Introduction to Data Structure

- Data & Information,
- Concept and Need of DS,
- Abstract Data Type.
- Types of Data Structure : Linear & Non-linear.
- Operation on Data Structure:
- Algorithm Complexity : Time & Space.

Questions to be discussed :

1) Define data and information in brief.
2) What is data structure ? Why we need a data structure explain in brief.
3) Explain different types of data structure.
4) Discuss the various operations performed on data structures.
5) Define algorithm. Describe its various characteristics.
6) Define time complexity & space complexity with suitable example.
7) Write short notes on the following :
    a. ADT
    b. Primitive data type
    c. Composite data type

Data & Information :

- The data is just a collection of values and no conclusion.
- It is a collection of numbers, alphabets and special symbols.
- After processing data, it becomes information that can be helpful in making some decision.
- For processing of data it should be in main memory, since processor only operates on the data in main memory.
- In order to represent the data in main memory some model is needed so that data can be processed efficiently.
- This model is known as data structure.

Difference between data & information :

DATA	INFORMATION
Data are the collection of values.	Information is meaningful data.
Data is collection of facts, which it self have no meaning.	After processing those facts & converted into meaningful data is called information.
Data does not directly helps in decision making.	Information directly helps in decision making.
Data doesn't depend on information.	Information depends on data.
Exp : 7634252438	Exp : A person's phone number 7634252438

Data type :

- A data type is defined as the set of values and the set of operations that operate on those values.
- The data types can be classified in several categories including :
  - ❖ Primitive data type
  - ❖ Composite data type
  - ❖ User-defined data type

Primitive data type :

- The data type provided by a programming language are known as Primitive data types or built-in data types.
- Example :

  int, char, float etc.

Composite data type :

- The data types that are derived from primitive data types are known as composite data types or derived data types.
- Example :

  Array, functions, pointers etc.

User defined data types :

- Programming language allow users to define new data types as per their requirements.
- Example :

  Structure, unions, and enumerations etc.

Abstract data type(ADT) :

- An abstract data type is an abstraction of a data structure that provides only the interface.
- The process of providing only the essentials and hiding the details is known as abstraction.
- ADT only mentions what operations are to be performed but not how these operations will be implemented.
- It is called “abstract” because it gives an implementation-independent view.
- The keyword “Abstract” is used if we can use these datatypes.
- Examples of ADT are Stack, Queue, Linked List etc.

Example.

If we consider the smartphone. We look at the high specifications of the smartphone, such as:

- 4 GB RAM
- Snapdragon 2.2ghz processor
- 5 inch LCD screen
- Dual camera
- Android 8.0

The above specifications of the smartphone are the data, and we can also perform the following operations on the smartphone:

- **call():** We can call through the smartphone.
- **text():** We can text a message.
- **photo():** We can click a photo.
- **video():** We can also make a video.

The smartphone is an entity whose data or specifications and operations are given above. The abstract/logical view and operations are the abstract or logical views of a smartphone.

What is data structure ?

- Data structure is a storage that is used to store and organize data.
- It is a way of arranging data on a computer so that it can be accessed and updated efficiently.
- It is an approach of organizing, storing and managing data so that we can access and modify it efficiently.
- The logical or mathematical model used to organize the data in main memory is called data structure.
- Data Structure allows data to be stored in a specific manner in the memory.

Need of Data Structures :

Data structures are needed for the following reasons:

- It is used to organized the storage and retrieval of data and information which is stored in both main memory and secondary memory.
- It helps in the management of large amounts of data such as large databases and indexing services such as a hash table.
- Data Structure helps in efficient data search and retrieval.
- For specific problems, specific Data structures are used.

Types of Data Structure :

Basically, data structures are divided into two categories:

- Linear data structure
- Non-linear data structure

Linear data structure :

- Linear data structure is one in which its elements form a sequence.
- It means each element in the data structure has a unique predecessor and a unique successor.
- An array is the simplest linear data structure.
- Various other linear data structure are linked list, stacks, and queues etc.

Non - linear data structure :

- A non-linear data structure is one in which its elements do not form a sequence.
- It means unlike linear data structure, each element is not have a unique predecessor and a unique successor.
- Tree and graph are the two data structures which come in this category.

Linear Data Structures	**Non Linear Data Structures**
The data items are arranged in sequential order, one after the other.	The data items are arranged in non-sequential order (hierarchical manner).
All the items are present on the single layer.	The data items are present at different layers.
The memory utilization is not efficient.	Different structures utilize memory in different efficient ways.
Example: Arrays, Stack, Queue	Example: Tree, Graph, Map

Operations on data structure

Traversing :

- Every data structure contains the set of data elements.
- Traversing the data structure means visiting each element of the data structure in order to perform some specific operation like searching or sorting.

Insertion:

- It can be defined as the process of adding the elements to the data structure at any location.
- New element can be inserted anywhere in the data structure such as beginning, end or middle.

Deletion :

- The process of removing an element from the data structure is called Deletion.
- We can delete an element from the data structure at any random location.
- If we try to delete an element from an empty data structure then **underflow** occurs.

Searching :

- The process of finding the location of an element within the data structure is called Searching.
- There are two algorithms to perform searching, Linear Search and Binary Search.

Sorting :

- The process of arranging the data structure in a specific order is known as Sorting.
- There are many algorithms that can be used to perform sorting, for example, insertion sort, selection sort, bubble sort, etc.

Merging :

- It is the process of combining the elements of two sorted data structure into a single sorted data structure.
- Note that both the structure to be merged should be similar.

What is an algoritham ?

- An algorithm is the way of writing a programming language step by steps.
- In programming, algorithm are the set of well defined instruction.
- It is a sequence instruction to solution of a problem.
- An algorithm is not the computer code it just the instructions which gives clear idea to write the computer program.

Characteristics of an algorithm :

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

Algorithm Complexity : Time & Space.

- The complexity of an algorithm is a function describing the efficiency of the algorithm.
- There are two main complexity measures of the efficiency of an algorithm :
  1. Time complexity
  2. Space complexity

Time complexity :

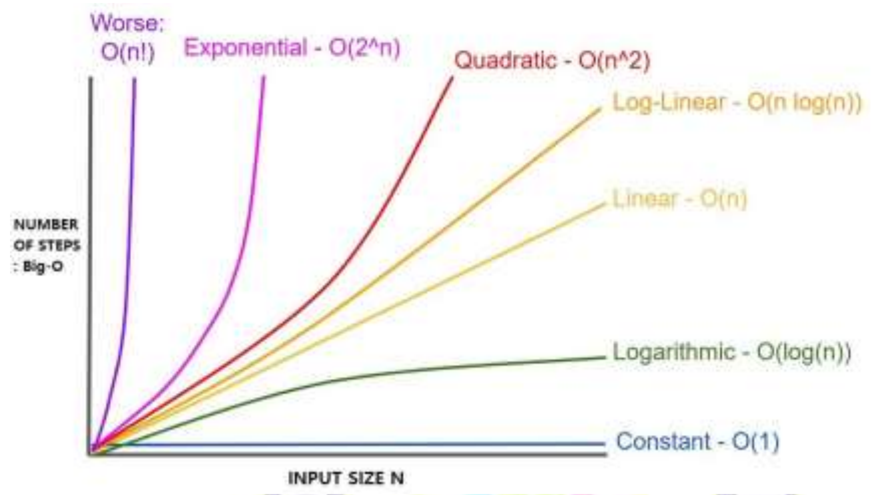
- It is the amount of time required to execute an algorithm.
- Time complexity represents the number of times a statement is executed**.**
- The time complexity depends on programming language, processing power & main memory etc.

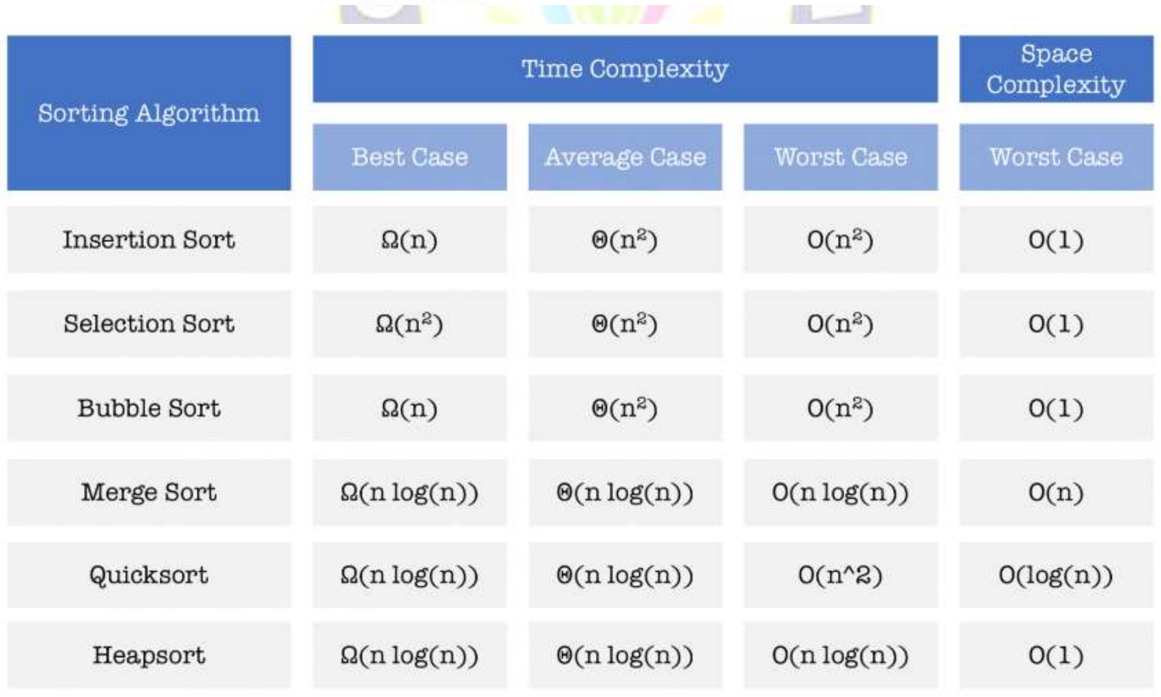
Following are the various notations used for expressing time complexity & this notation is known as asymptotic notation :

- **Big Oh Notation(O) :** It is used to express the upper bound of an algorithm's running time.
- **Omega Notation(Ω) :** It is used to express the lower bound of an algorithm's running time.
- **Theta Notation(θ) :** It lies between Big-Oh and Omega and is used to express the exact asymptotic behavior of an algorithm’s running time.

There are different types of time complexities :

- Constant Time Complexity : **O(1)**
- Linear Time Complexity : **O(n)**
- Logarithmic Time Complexity : **O(log n)**
- Quadratic Time Complexity : **O(n^2)**
- Exponential Time Complexity : **O(2^n)**
- Factorial Time Complexity : **O(n!)**





Sorting Algorithm	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Merge Sort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)

Space complexity :

- The space complexity is the amount of memory space required by the algorithm.
- In other words, we can say space complexity is the approximate total extra space required by the program to run.

Unit – 02 : Searching & Sorting

- Searching : Implementation of Different searching algorithm.
- Sorting : Implementation of Different Sorting algorithm.

Searching :

- The process of finding a particular data item in a list is known as searching.
- Searching is a operation that are performed on a list, which is maintained either in array or in the linked list.
- To perform search operation on a given list, various searching techniques are available :
  1) Linear search
  2) Binary search
  3) Interpolation search
  4) Hashing
- The linear search algorithm can be applied on array or on linked list.
- It does not require any additional data structure to perform the search operation.
- The binary and interpolation search algorithm are applied only on sorted array.
- Binary and interpolation search algorithm also does not require any additional data structure to performing search operation.
- Hashing uses a data structure called hash table which are an array of fixed size.
- Linear search algorithm is slower than binary & interpolation, which in turn are slower than hashing.

Linear search :

- The linear search is one of the simplest searching techniques.
- In this technique, the array is traversed sequentially from the first element until the value is found or the end of the array.
- During traversing, each element of the array is compared with the value to be searched, and if the value is found the search is said to be successful.
- This technique is suitable for performing a search in a small array or in an unsorted array.
- Time complexity of linear search :
  - ✓ Best case - O(1)
  - ✓ Avearge case - O(n)
  - ✓ Worst case - O(n)

Binary search :

- The binary search technique is used to search for a particular data item in a sorted (ascending or descending order) array.
- In this technique, the item to be searched is compared middle element of the array.
- If the item is equal to the middle element, then the search is successful.
- If item is smaller than the middle value, then item is searched before the middle element.
- If item is greater than the middle value, then item is searched after the middle element.
- This process is repeated until the value is found or the array is reduced to a single element that is not equal to item.
- The binary search technique is used for larger and sorted array.
- It is faster as compared to linear search.
- Time complexity of binary search is $O(\log_2 n)$.
- The middle of the array is determined using formula MID.

$$MID = \frac{Low+High}{2}$$

Example :

1, 2, 3, 4, 5, 6, 7

Interpolation search :

- Interpolation search is similar to binary search because both are applied on sorted array.
- It is based on the assumption that the elements in the list are uniformly distributes.
- It determines the location of the item to be searched according to the magnitude to the first & last elements of the array.
- Time complexity of interpolation search is $O(\log_2 n(\log_2 n))$.
- It uses the following formula for calculating the mid :

$$\text{Mid} = \text{Low} + (\text{high} - \text{low}) \times \frac{(\text{item}-\text{arr[low]})}{\text{arr[high]}-\text{arr[low]}}$$

Example :

1, 7, 13, 19, 25, 31, 36

Sorting :

- The process of arranging data in some logical order is known as sorting.
- Sorting is a operation that are performed on a list, which is maintained either in array or in the linked list.
- All the sort algorithm take a list as input and produce a sorted list as output.
- To sort a given list, various algorithms are available.
  - ✓ Insertion sort
  - ✓ Selection sort
  - ✓ Bubble sort
  - ✓ Quick sort
  - ✓ Merge sort
  - ✓ Heap sort
  - ✓ Shell sort
  - ✓ Radix sort
- The simple sort algorithms are insertion, selection & bubble sorting algorithm doesn't requires additional data structure to sort a given list.
- The merge, quick & heap use additional data structure.
- Merge & quick sort algorithms use a stack whereas heap uses heap data structure.
- Insertion, selection & bubble sorting algorithms are slower than others.

Insertion sort :

- This algorithm is one of the simplest algorithm with simple implementation.
- Basically, Insertion sort is efficient for small data values
- The insertion sort algorithm selects elements and insert it at its proper position in the list.
- Insertion sort works similar to the sorting of playing cards in hands.
- It is not appropriate for large data because the time complexity of insertion sort in the average case and worst case is **O(n²)**.

Example : Sort the values stored in array in ascending order using insertion sort.

7, 33, 20, 11, 6

Selection sort :

- In selection sort, first smallest elements in the list is searched and is swapped with the first position element.
- Then the second smallest element is searched in the list and swapped with the second position element and so on.
- The selection sort requires (n-1) passes to sort an array containing n elements.
- The average and worst-case complexity of selection sort is **O(n²)**, where **n** is the number of items.
- Due to this, it is not suitable for large data sets.

Example : Sort the values stored in array in ascending order using selection sort.

8, 33, 6, 21, 4

Bubble sort :

- It is the simplest sorting algorithm that repeatedly swapping the adjacent elements if they are in the wrong order.
- The bubble sort algorithm requires n-1 passes to sort an array.
- In the first pass each element(except last) in the list is compared with next element and swapped it is greater.
- After the first pass , the largest elementin the list is placed at the last position.
- Similarly, in the second pass the second largest element placed at its appropriate position.

Example : Sort the values stored in array in ascending order using bubble sort.

8, 7, 65, 5, 43

Quick sort :

- Quick sort algorithm is based on the fact that it is easier and faster to sort two smaller array than one larger array.
- It follows the principle of divide-and-conquer.
- It first picks up a portioning element, called pivot.
- Pivot devides the list into two sub lists such that the element in the left sub list is smaller and elements in the right sub list are greater than pivot.
- These two sub lists are sorted separately.
- The same process applied to sort the elements of left and right sub lists.
- This process is repeated recursively until element are sorted.

Example : Sort the values stored in array in ascending order using quick sort.

8, 33, 6, 21, 4

Merge sort :

- The merge sort algorithm also follows the principle of divide-and-conquer.
- In this sorting, the list is first divided into two halves.
- The left and right sub list again recursively divided into two sub list until each sub list contains not more than one element.
- The sub list containing only one element do not require any sorting.
- Therefore, we start merging the sub lists recursively until we get the final sorted array.

Example : Sort the values stored in array in ascending order using merge sort.

18, 33, 5, 20, 55, 89, 4, 14

Heap sort :

- Heap sort are much more efficient version of selection sort.
- To sort an array of size n in ascending order using heap sort the following steps are performed :
  - ➢ The max-heap is build from the given array.
  - ➢ The root element is swapped with the last element in the array.
  - ➢ The heap of remaining elements is restored.
  - ➢ The step 2 and 3 are repeated until there are no more elements.

Example : Sort the values stored in array in ascending order using heap sort.

9, 11, 6, 45, 22, 10, 12, 90, 67, 17

Shell sort :

- The shell sort algorithm is invented by Donald L Shell in 1959.
- It is the most efficient sorting algorithm among all the algorithms.
- In this algorithm, the elements are separated by a specific distance d.
- Once all the elements are in order with distance d then, the value of d is reduced.
- Each time value of d is reduced by 1 and process will continue until elements are sorted.

$$d = \frac{\text{Size}}{2}$$

Example : Concider an unsorted array with size=8, sort the array in ascending order.

11, 33, 99, 22, 88, 77, 55, 44

Radix/Bucket sort :

- The radix or bucket sort algorithm sorts the numbers considering individual digits starting from right to left.
- In the first pass, the numbers are sorted according to the digits at units place.
- In the second pass, the numbers are sorted according to the digits at tens place and so on.
- The radix sort requires 10 buckets, numbered 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.
- The number of passes in the algorithm is equal to number of digits in the largest number.

Example : Concider an unsorted array with size=8, sort the array in ascending order.

318, 233, 56, 899, 912, 674, 555, 110, 21, 746

Unit – 03 : Stacks & Queues

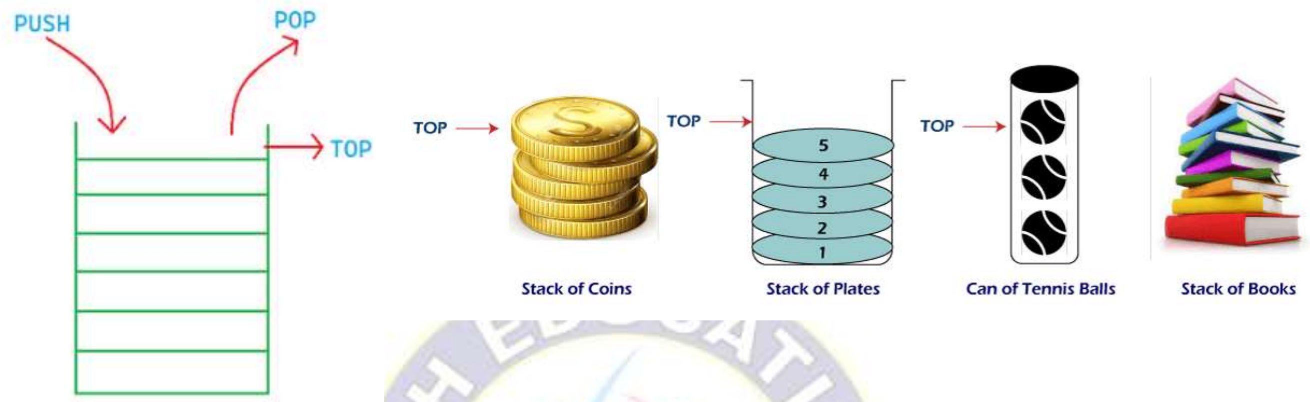
- Stack :
  - Introduction to Stack
  - Stack Operation Conditions
  - Application of Stack : Infix- to-Postfix Transformation Evaluation Postfix.
- Introduction to Queue,
  - Dequeue:
  - Array Representation of Queue;
  - Operation on Queue:
  - Types of Queues: Linear Queue, Circular Queue, Priority Queue,
  - Application of Queue.

Questions to discussed :

1) What is stack ? What are the application of stack ?
2) Explain operations performed on stack with example.
3) Convert the following expressions :
4) Define queue ? Write the application of queue.
5) Explain in details array repersentation of queue.
6) Explain operations performed on queue with example.
7) Differentiate between stack & queue.
8) Write short notes on :
   a) Dequeue
   b) Circular queue
   c) Priority queue

What is stack ?

- A stack is a linear data structure.
- In stack insertions and deletions of elements can be take place at one end only called top.
- Insert and delete operations are known as push and pop operations respectively.
- The last element added in the stack is the first element to be removed.
- A stack works on the principle of LIFO or FILO techniques.
- A stack can be implemented using an array or a linked list.

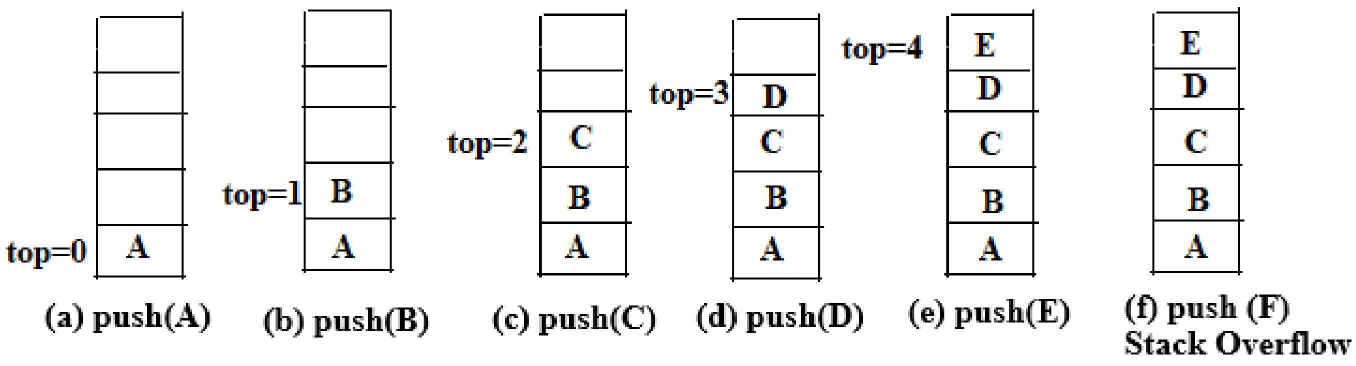


Operations on Stack :

The two basic operations that can be performed on the stack :

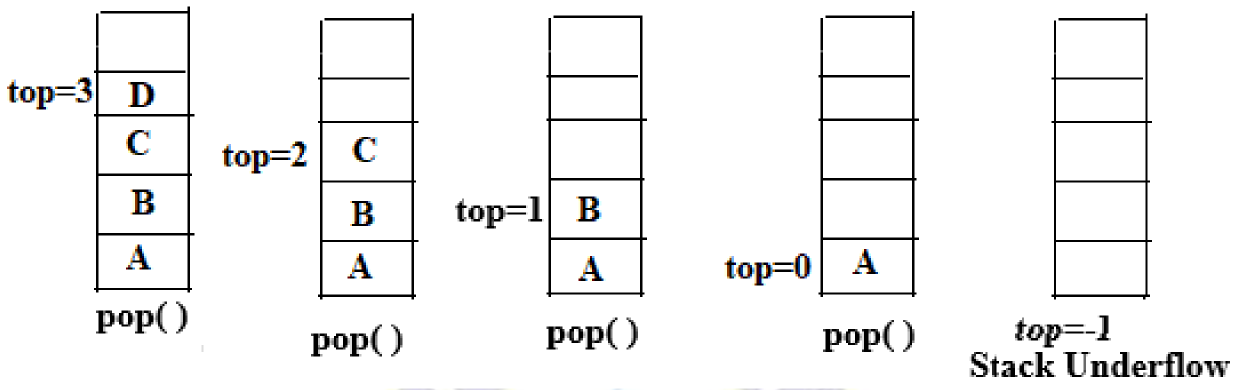
Push :

- When we insert an element in a stack then the operation is known as a push.
- Before inserting a new element onto the stack, it is necessary to test the condition of overflow.
- If the stack is full(Top = Max-1) then the overflow condition occurs.
- If the stack is not full, then push operation can be performed successfully.



Pop :

- When we delete an element from the stack, the operation is known as a pop.
- Before removing the element from the stack, it is necessary to ckeck the condition of underflow.
- If the stack is empty(Top = -1) then the underflow condition occurs.
- If the stack is not empty, then pop operation can be performed successfully.



Applications of Stack :

The applications of stacks are in :

- Reversing string
- Checking whether the arithmetic expression is properly parenthesized
- Expression convertion :
    - ✓ infix to postfix
    - ✓ infix to prefix
    - ✓ prefix to infix
    - ✓ prefix to postfix
    - ✓ postfix to infix
    - ✓ postfi to prefix.
- Implementing recursion and function call etc.

Expression convertion :

- Another important application of stack is the conversion of expressions from infix notation to postfix and prefix notations.
- There are three types of notations :
  1. Infix notation
  2. Prefix notation
  3. Postfix notation

Infix notation :

- The general way of writing arithmetic expressions is known as infix notation, where the binary operator is placed between two operands.
- Example :
  - a + b
  - (a - c) * d
  - ((a + b) * (d/f) - f) are in the infix notation.

Prefix notation :

- The notation in which an operator occurs before its operands is knowns as the prefix notation.
- It is also known as Polish notation.
- Example :
  - +ab
  - *+acd are in the prefix notation.

Postfix notation :

- The notation in which an operator occurs after its operands is knowns as the postfix notation.
- It is also known as Reverse Polish or suffix notation.
- Example :
  - ab+
  - ac–d* are in the postfix notation.

x + y = Infix
+xy = Prefix
xy+ = Postix

Conversion of Infix to Postfix notation :

- To convert an arithmetic expression from infix to postfix, the precedence & associativity rules of operator is apply.
- The operators of the same precedence are evaluated from left to right.
- The order of precedence of these operators is as follows:
  - Higher priority *, /
  - Lower priority +, –

Example :

1. a + b * c [Ans : abc*+]
2. a – (b + c) * d/f [Ans : abc+d*f/-]
3. Evaluate a postfix expression

   4 3 2 + 2 ^ * 5 - [Ans : 95]

Conversion of Infix to Prefix notation :

- The conversion of infix to prefix, is similar to conversion of infix to postfix notation.
- The only difference is that the expression in the infix notation is scanned in the reverse order, that is from right to left.

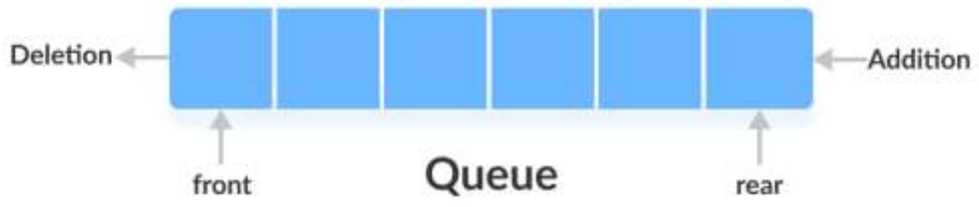
Convert infix to prefix :

1. a – (b + c) * d/f [Ans : -a/*+bcdf]
2. Evaluate a prefix expression.

   - * 4 ^ + 3 2 2 5 [Ans : 95]

What is queue ?

- A queue is a linear data structure.
- In queue new element is inserted at one end and deleted from other end.
- The end of the queue at which element is added is known as rear.
- The end of the queue from which the element is deleted is known as front.
- Queue works on the principle of FIFO or LILO.



Operations on queue :

The following are the basic operations that can be performed on the queue :

Insert operation :

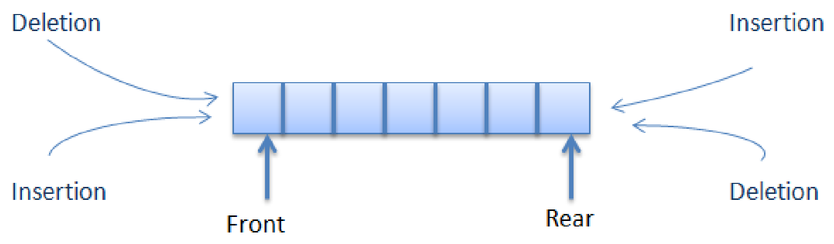
- To insert an element at the rear of the queue.
- If the queue is full then the overflow condition occurs.
- Before inserting a new element onto the queue, it is necessary to test the condition of overflow.
- If the queue is not full, then insert operation can be performed successfully.

Delete operation :

- To delete an element from the front of the queue.
- If the queue is empty then the underflow condition occurs.
- Before deleting the element from the queue, it is necessary to ckeck the condition of underflow.
- If the queue is not empty, then delete operation can be performed successfully.

What is a Dequeue ?

- Deque is a linear data structure.
- The dequeue stands for Double Ended Queue.
- In dequeue the insertion and deletion operations are performed from both ends.
- We can say that dequeue is a generalized version of the queue.
- It does not follow the FIFO rule.

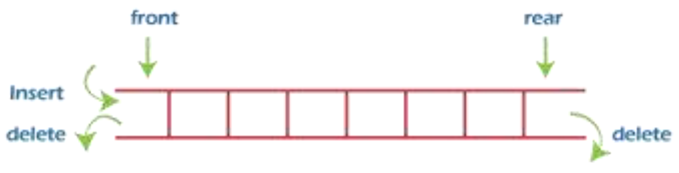


There are two types of dequeue :

1. Input restricted queue
2. Output restricted queue

Input restricted Queue

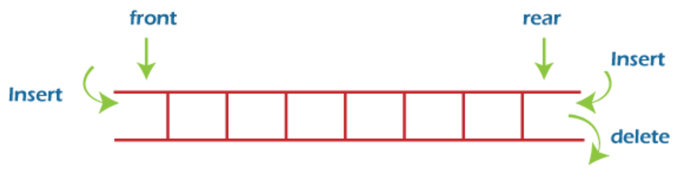
- In input restricted queue, insertion operation can be performed at only one end, while deletion can be performed from both ends.



input restricted double ended queue

Output restricted Queue

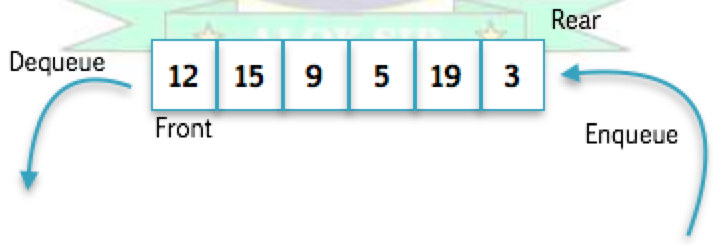
- In output restricted queue, deletion operation can be performed at only one end, while insertion can be performed from both ends.



Output restricted double ended queue

Linear Queue :

- A Linear Queue is a linear data structure.
- It consists of data elements which are connected in a linear fashion.
- It follows the FIFO (First In First Out) principle.
- A real-life example of a queue is any queue of customers waiting to buy a product from a shop where the customer that came first is served first.



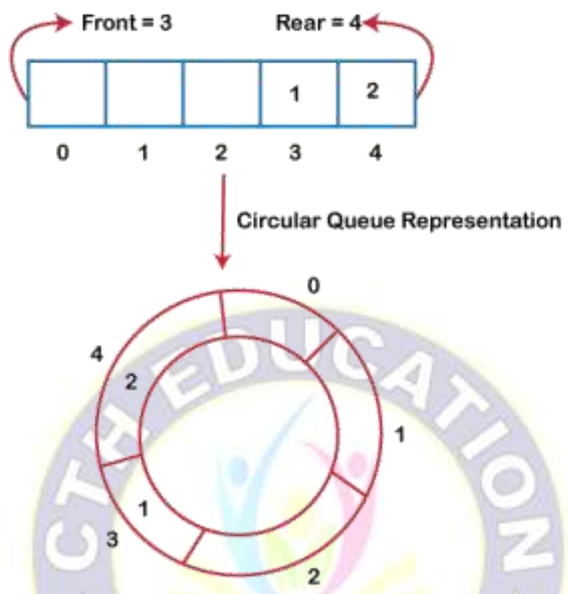
Operations on Linear Queue :

There are two operations that can be performed on a linear queue:

- **Enqueue :** The enqueue operation inserts the new element from the rear end.
- **Dequeue:** It is used to delete the existing element from the front end of the queue.

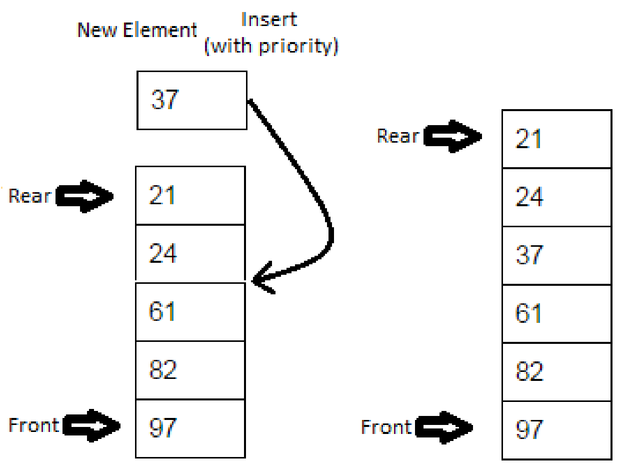
Circular queue :

- A circular queue is similar to a linear queue.
- It is also based on the FIFO (First In First Out) principle.
- In a circular queue the last position is connected to the first position in a circular queue that forms a circle.
- It is also known as a Ring Buffer.



Priority queue :

- A priority queue is a type of queue in which each element is associated with priority.
- The elements are added and deleted according to that priority.
- In priority queue the following two rules are applied :
  - ➢ The element with higher priority is processed first.
  - ➢ The element with the same priority are processed according to the order which are added.



Array repersentation of queue :

- The representation of a queue as an array needs an array to hold the elements of the queue and two variables – Rear & Front.
- Initially the value of rear and front is set to -1 to indicate n empty queue.

Insert operation :

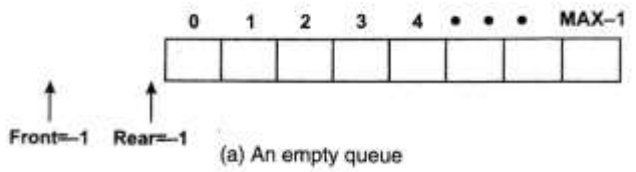
- Before inserting a new element onto the queue, it is necessary to test the condition of overflow.
- The queue is in a condition of overflow(full) then the rear is equal to MAX-1.
- If the queue is not full, then insert operation can be performed successfully and the Rear is incremented by one.

Delete operation :

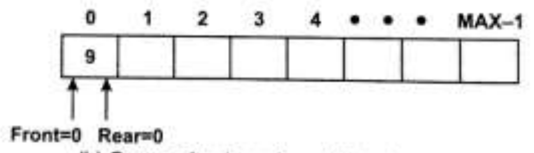
- Before deleting the element from the queue, it is necessary to ckeck the condition of underflow.
- The queue is in a condition of underflow(empty) then the Front is equal to -1.
- If the queue is not empty, then delete operation can be performed successfully and the Front is incremented by one.

The total number of elements in a queue at a given point of time can be calculated from the value of Rear and Front.

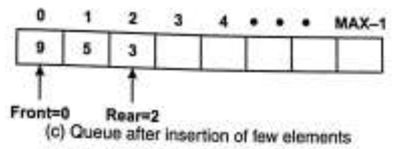
Number of elements = Rear – Front + 1



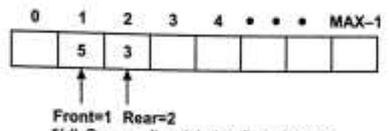
(a) An empty queue



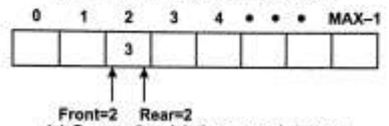
(b) Queue after insertion of first element



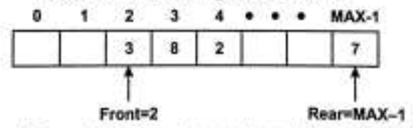
(c) Queue after insertion of few elements



(d) Queue after deleting first element



(e) Queue after deleting second element



(f) Queue having vacant space though Rear = MAX-1

Application of queues :

There are many applications of queue in computer science :

- Railway ticket reservation.
- In Banking system also implemented.
- Simulation(Process of handlling a real life situation through a computer).
- In operating system.
- CPU Schedulling.
- Device management(like printer or disk).
- Level order traversal of binary tree etc.

Difference between stack & queue :

Stacks	Queues
Stacks are based on the LIFO principle.	Queues are based on the FIFO principle.
Insertion and deletion in stacks takes place only from one end of the list called the top.	Insertion and deletion in queues takes place from the opposite ends called rear front resp.
Insert operation is called push operation.	Insert operation is called enqueue operation.
Delete operation is called pop operation.	Delete operation is called dequeue operation.
In stacks we maintain only one pointer to access the list, called the top.	In queues we maintain two pointers to access the list, called front and rear.
Stack is used in solving problems works on recursion.	Queue is used in solving problems having sequential processing.
Stack does not have any types.	Queue is of three types – 1. Circular Queue 2. Priority queue 3. double-ended queue.

Unit – 04 : Linked List

- Introduction to Linked List Terminology :
  - Node,
  - Address,
  - Pointer,
  - Data field and Next Pointer,
  - Empty List.
- Types of Lists : Singly Linked List, Doubly Link list, Circular Linked List.
- Operation on Linked List :

Questions to be discussed :

1. Define linked list. What are the applications of linked list ?
2. Explain different types of linked list in data structure.
3. What are the advantage & disadvantage of circular linked list.
4. What are the advantage & disadvantage of doubly linked list.
5. Explain different operations performed on linked list.
6. Differentiate between array and linked list.

- **Node :** In linked list a node is an elements that consist of data and address of next node.
- **Address :** It is a field in node that have address of next node.
- **Pointer :** Pointer is a special type of variable that stores the address of ordinary variable.
- **Data field & next pointer :** A node having both data field & next pointer.
- **Empty linked list :** A linked list having no element is called empty linked list.

What is list ?

- A list is a collection of data items of similar type arranged in a sequence.
- For example :

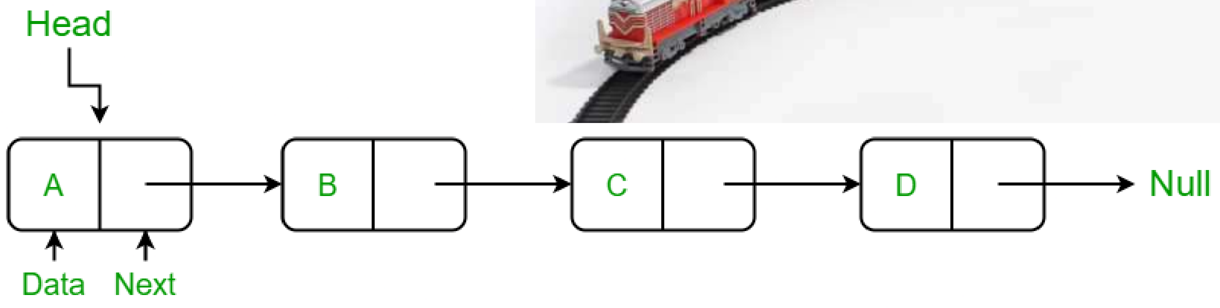
  List of names, list of marks, List of employees, and so on....
- Lista are once created and then can be modified during their lifetime.
- We need to add a new element, search and delete an existing elements from the lists.
- We can also combine two or more list or split a list as per the requirements.

Why Linked List ?

- Arrays can be used to store linear data of similar types, but arrays have the following limitations :
  - The size of the arrays is fixed
  - Insertion of a new element/Deletion of a existing element in an array is expensive.
  - To remove above limitation we introduce the concept of linked list.

What is Linked List ?

- It is a collection of nodes contains data and pointer.
- Linked List is a linear and dynamic data structure.
- In linked list elements are stored in non-contiguous location.
- It is defined as linear collection of homogeneous elements called node.
- Each node consists of two or more parts.
- One part essentially stores an element of list and other part stores pointer.
- Time complexity of insertion is O(1) and for searching O(n).
- A linked list can be classified into various types such as :
  1. Singly linked list
  2. Doubly linked list
  3. Circular linked list.



Advantages Of Linked List :

- **Dynamic data structure :** So there is no need to give the initial size of the linked list.
- **No memory wastage:** Since the size of the linked list increase or decrease at run time so there is no memory wastage of memory.
- **Implementation:** Linear data structures like stacks and queues are often easily implemented using a linked list.
- **Insertion and Deletion Operations :** There is no need to shift elements after the insertion or deletion of an element only the address present in the next pointer needs to be updated.

Disadvantages Of Linked List :

- **Memory usage :** More memory is required in the linked list as compared to an array.
- **Traversal:** In a Linked list traversal is more time-consuming as compared to an array.
- **Reverse Traversing:** In a singly linked list reverse traversing is not possible.
- **Random Access:** Random access is not possible in a linked list due to its dynamic memory allocation.

Difference between array and linked list.

Array	Linked List
It stores data in contiguous manner.	It stores data in non-contiguous manner.
Memory is allocated at compile time.	Memory is allocated at run time.
Array is a static data structure.	Linked list is a dynamic data structure.
Fixed in size.	Dynamic in size.
It uses less memory space.	It uses more memory space.
Insertions & deletion operation is slower.	Insertions & deletion operation is faster.
Time complexity : ▪ Insertion – O(n) ▪ Searching – O(1)	Time complexity : ▪ Insertion – O(1) ▪ Searching – O(n)

Basic Operations on Linked List :

- There are various operations that allow us to perform different actions on linked lists.
  - **Traversal** : To traverse all the nodes one after another.
  - **Insertion** : To add a node at the given position.
  - **Deletion** : To delete a node.
  - **Searching** : To search an element(s) by value.
  - **Updating** : To update a node.
  - **Sorting :** To arrange nodes in a linked list in a specific order.
  - **Merging :** To merge two linked lists into one.

Note : Only linear search can be performed on linked list.

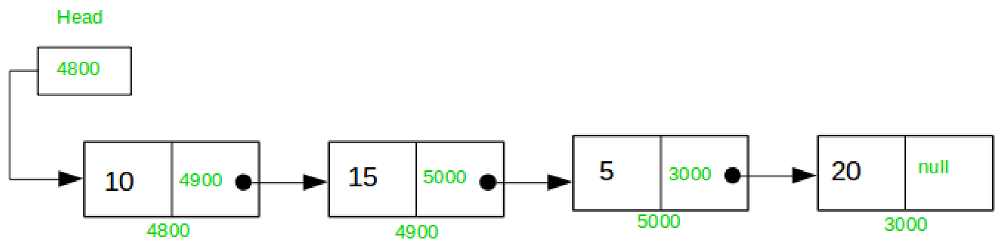
Types of linked list :

1. Singly linked list
2. Doubly linked list
3. Circular linked list.

Single Linked List ?

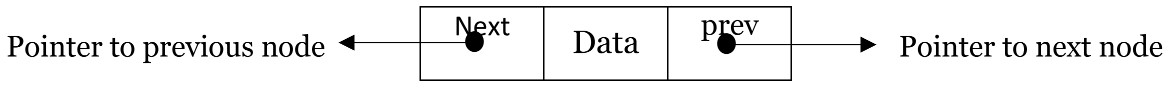
- It is a sequence of elements in which every element has link to its next element in the sequence.
- In any single linked list, the individual element is called as "Node".
- Every "Node" contains two fields, data field, and the next field.
- The data field is used to store actual value of the node and next field is used to store the address of next node in the sequence.
- In a single linked list always next part of the last node must be NULL.
- **Head :** Always points to the first node of the linked list.
- **Next : A** pointer of the last node is NULL, so if the next current node is NULL, then we have reached the end of the linked list.

Singly Linked list

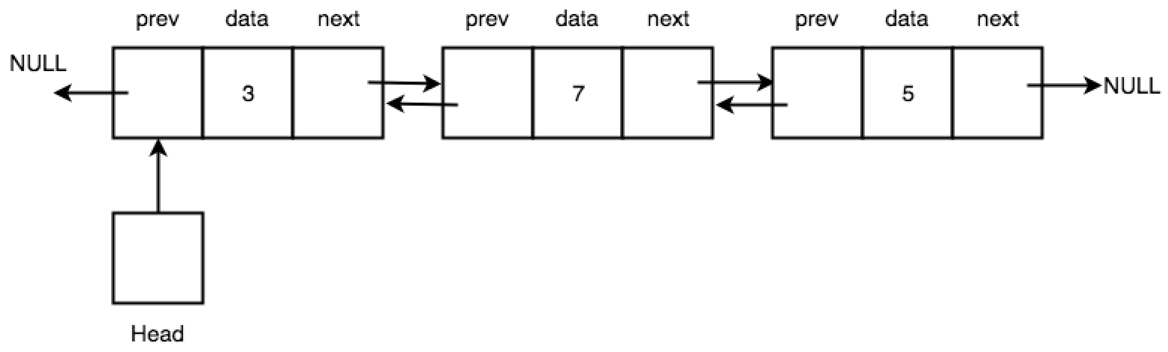


Doubly Linked List ?

- In a sinly linked list, each node contains a pointer to the next node and it has no information about its previous node.
- That means singly linked list traverse in only one direction fron beginning to end.
- However sometimes it is required to traverse in the backward direction from end to beginning.
- This can be maintained by additional pointer in each node of the list to the previous node and such type of linked list is called doubly linked list.
- Each node of doubly linked list consists of three fields : prev, info and next.



- Info field contains the data.
- Prev field contains the address of the previous node.
- Next field contains the address of the next node.



Advantages over Singly Linked List-

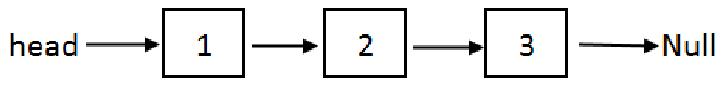
- It can be traversed both forward and backward direction.
- The delete operation is more efficient if the node to be deleted is given.
- The insert operation is more efficient if the node is given before which insertion should take place.

Disadvantages over Singly Linked List-

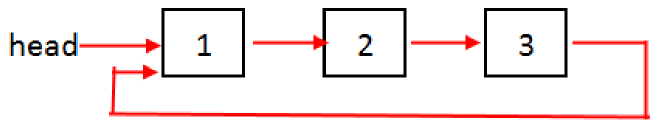
- It will require more space as each node has an extra memory to store the address of the previous node.
- The number of modification increase while doing various operations like insertion, deletion, etc.

Circular linked list :

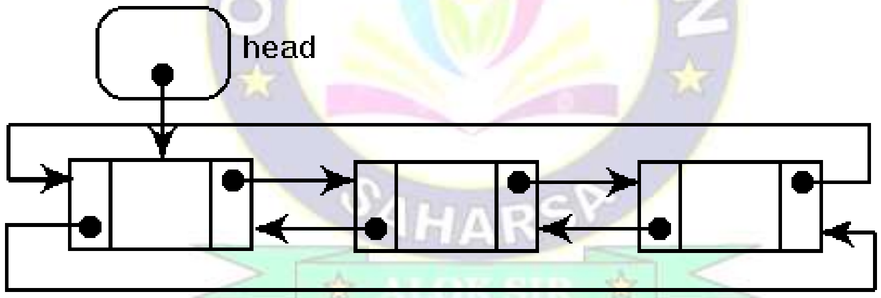
- A circular linked list is either a singly or doubly linked list in which there are no **NULL** value.
- A linear linked list in which the next field of the last node points back to the first node instead of containing NULL, is called circular linked list.
- The main advantage of circular linked list over a linear linked list is that by starting with any node in the list, we can reach any of its predecessor nodes.



Singly Linked List



Circular Linked List



Doubly Linked Circular list

Advantages of a Circular linked list

- The list can be traversed from any node.
- Circular lists are the required data structure when we want a list to be accessed in a loop.
- We can easily traverse to its previous node in a circular linked list, which is not possible in a singly linked list.

Disadvantages of Circular linked list

- If not traversed carefully, then we could end up in an infinite loop because here we don't have any **NULL** value to stop the traversal.
- Operations in a circular linked list are complex as compared to a singly linked list and doubly linked list like reversing a circular linked list, etc.

Difference between singly linked list and doubly linked list :

Singly linked list (SLL)	Doubly linked list (DLL)
SLL nodes contains 2 field-data field and next link field.	DLL nodes contains 3 fields -data field, a previous link field and a next link field.
Pointers contains the address of next node in the list.	Pointers contains the address of next node as well as previous node in the list.
One pointers (Only next).	Two pointers (previous and next).
Each node is connected to the next node.	Next node also know about the previous node.
Traversal is possible in one direction only.	Traversal is possible in both directions.
The SLL occupies less memory because it has only 2 fields.	The DLL occupies more memory because it has 3 fields.

Difference between doubly linked list and circular linked list :

Doubly linked list	Circular linked list
Pointers contains the address of next node as well as previous node in the DLL.	Pointers can or cannot contains the address of previous node, it depends on the type of CLL.
Last element is linked to NULL.	Last element is linked to the first element.
Two pointers.	Can have one or two pointers.
Next node also know about the previous node.	The last node know about the first node.
Insertion in between two address to be updated.	Insertion in between four address to be updated.

Unit – 05 : Trees and Graphs

- Introduction to Trees : Basic Terminology :
  - Tree,
  - Degree of a Node
  - Level of Node,
  - Leaf Node,
  - Depth & Height of a Tree ;
  - Type of Tree.
- Introduction to Binary Tree (BT):
  - Operation on BT : Insertion, Deletion, Searching, and traversing the Tree (Pre-order, Post order, In order);
  - Application of BT.
- Introduction to Binary Search Tree (BST) :
  - Operation on BST : Insertion, Deletion, Finding Min-Max Element, Sorting Element;
- Introduction to AVL Tree : Insertion, Deletion;
- Introductions to Graph basic terminology :
- Adjacency List, Adjacency Matrix.

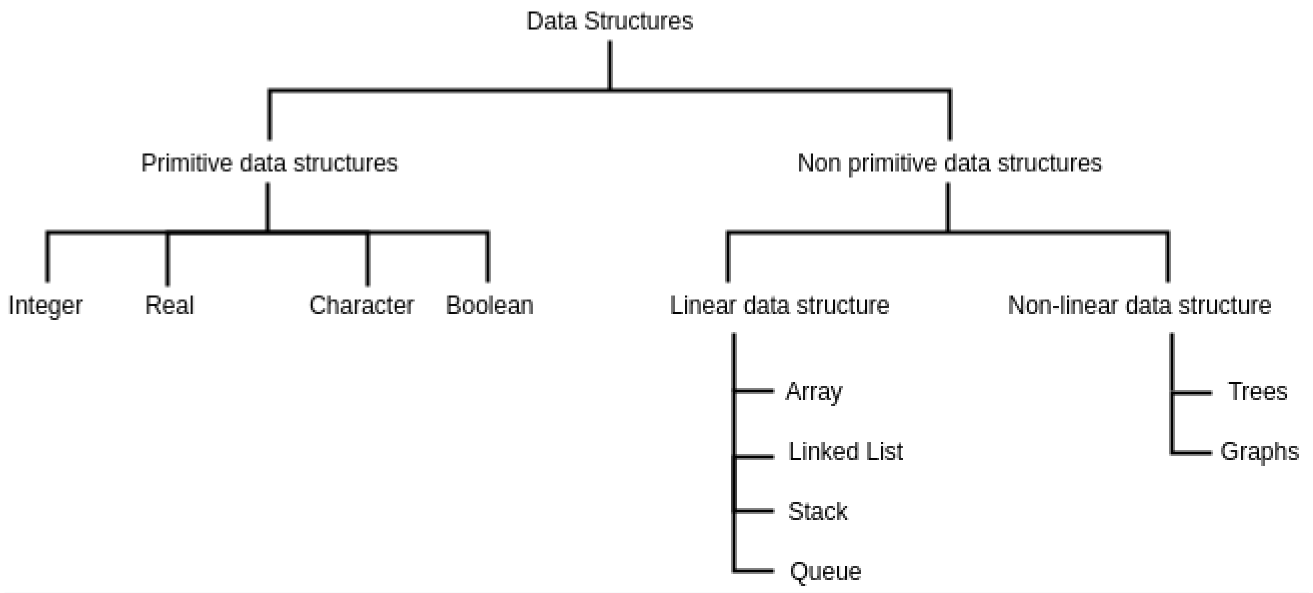
Questions to be discussed :

1. Define tree in data structure. Why Tree is considered to be a non-linear data structure ?
2. Explain binary tree and its type with examples.
3. What is preorder, inorder and postorder traversal ? Explain with examples.
4. Create a binary tree using inorder and postorder traversal :

   Inorder : D B H E A I F J C G

   Postorder : D H E B I J F G C A
5. Explain application of binary tree and create a binary tree from the following sequence :
   a. 14, 34, 22, 44, 11, 24, 33
6. Draw the binary tree to represent the following expression :
   a. (5 + 4 x (6 - 7)/(5 + 8))
7. Differentiate between binary tree and AVL tree.

Types of data structures :

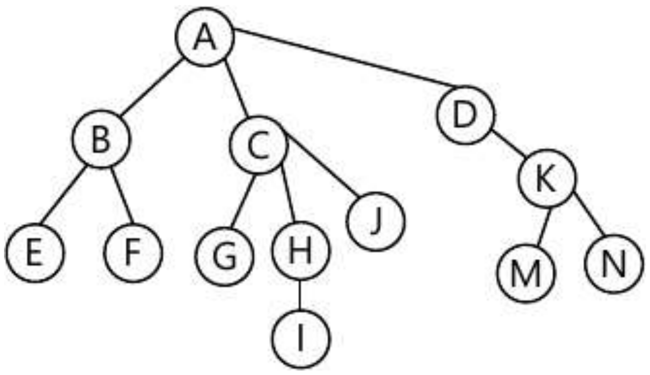


Difference between linear and non-linear data structure:

Linear Data Structure	Non-linear Data Structure
Here, data elements are arranged in a linear order.	Here, data elements are attached in hierarchical manner.
In linear data structure, single level is involved.	In non-linear data structure, multiple levels are involved.
Its implementation is easy.	While its implementation is complex.
In a linear data structure, memory is not utilized in an efficient way.	In non-linear data structure, memory is utilized in an efficient way.
Examples: array, stack, queue, linked list, etc.	Examples: trees and graphs.
Application: Application software development.	Application: Artificial Intelligence and image processing.

What is a Tree data structure ?

- A tree is used to represent the hierarchical structure of one or more elements called node.
- Each node of a tree stores data value & has zero or more pointers pointing to the other nodes.
- Each node in a tree can have zero or more child nodes which is at one level below it.
- Each child node can have only one parent node which is at one level above it.
- The node at the top of the tree is known as root node of the tree.
- While the nodes at the lowest level is known as leaf nodes of the tree.
- The root node is a special node having no parent & leaf nodes are node having no child nodes.
- Any node having child node as well as parent node is known as internal node.

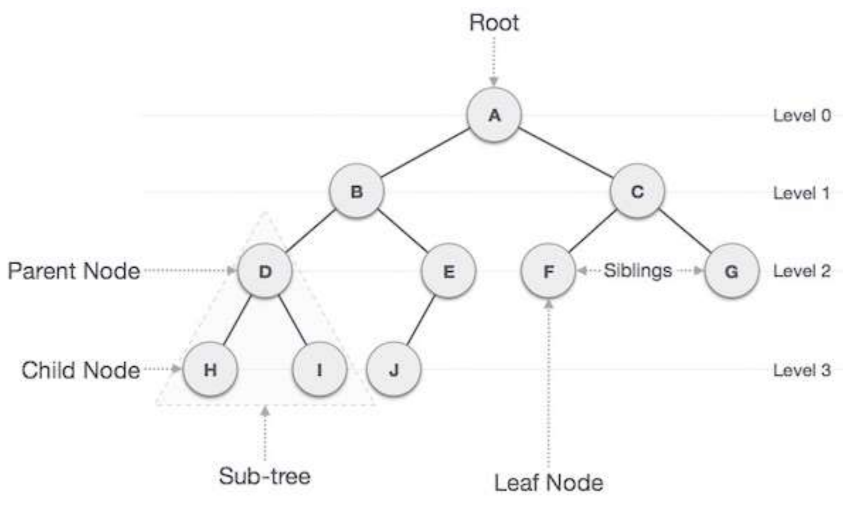


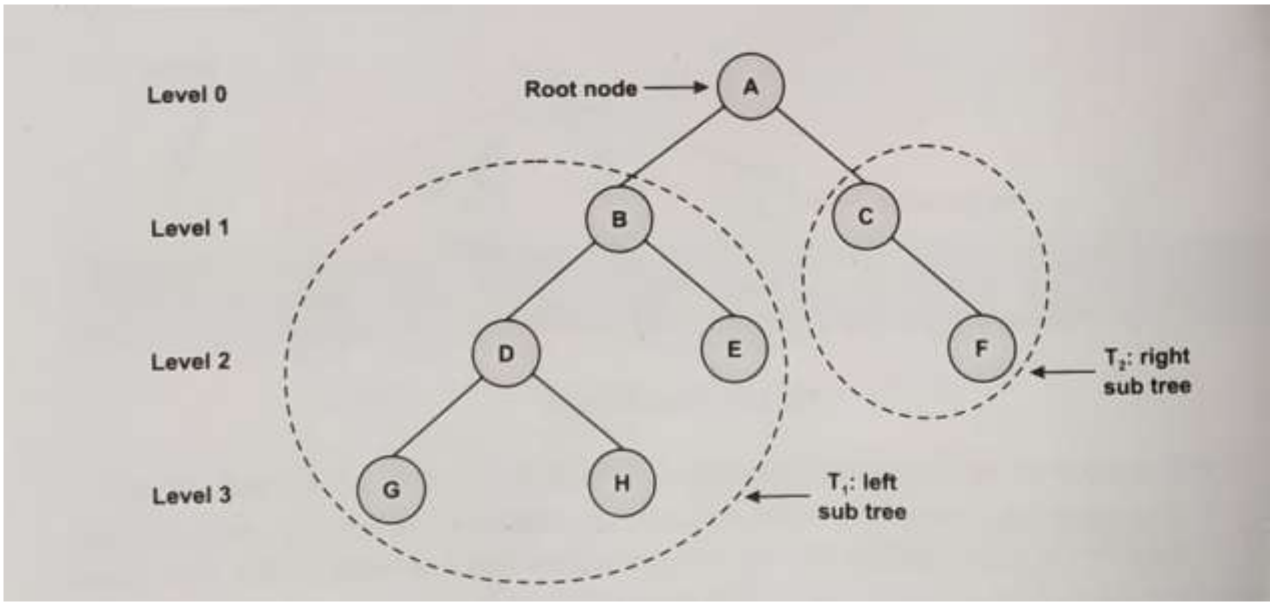
Why Tree is considered a non-linear data structure ?

- The data in a tree are not stored in a sequential manner.
- That means they are not stored linearly.
- Instead, they are arranged on multiple levels or we can say it is a hierarchical structure.
- For this reason, the tree is considered to be a non-linear data structure.

Some basic terminology :

- Root node
- Leaf node
- Internal node
- Degree of a Node
- Level of Node,
- Depth & Height of a Tree ;
- Siblings
- Ancestor and descendant





Root Node :

- The node at the top of the tree is known as root node of the tree.

Leaf Node :

- While the nodes at the lowest level is known as leaf nodes of the tree.

Internal Node :

- Any node having child node as well as parent node is known as internal node.

Degree of a Node :

- Degree of a node is equal to the number of child nodes.
- In the above figure, the nodes A, B & D have degree 2, node C has degree 1 and nodes G, H, E and F have degree 0.

Level of Node :

- In the above figure, the root node A belong to level 0, its child nodes belongs to level 1.
- Child nodes of nodes B and C belong to level 2, and so on.

Depth & Height of a Tree :

- The height of the tree is defined as the longest path from the root node to the leaf node.
- In the above figure, nodes G and H are nodes with highest level number 3, so depth of binary tree is 3.

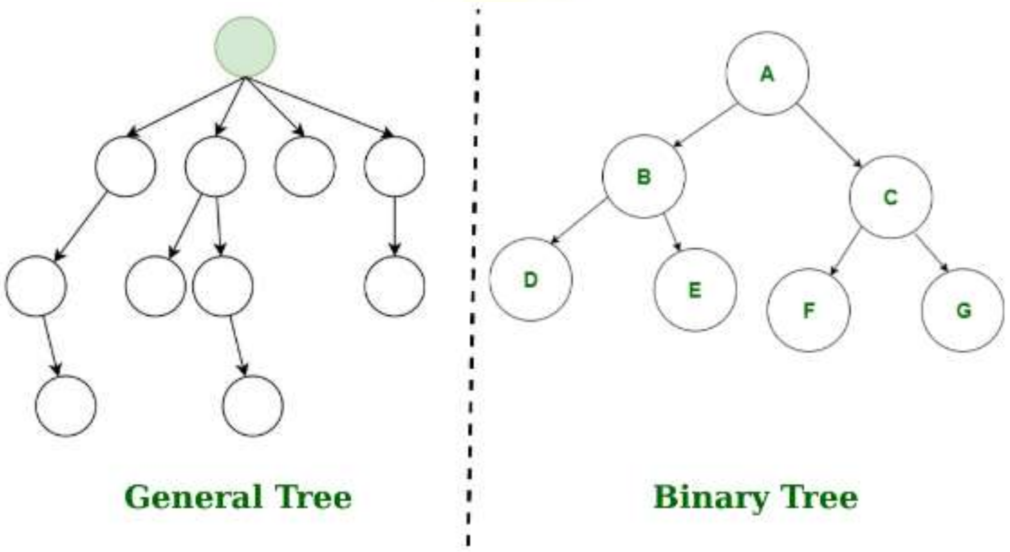
Siblings :

- The nodes belonging to the same parent node are known as sibling nodes.
- In the above figure, nodes B and C are sibling nodes as they have same parent node.
- Similarly, the nodes D and E are also sibling nodes.

Ancestor and Descendant of a Node :

- A node N1 is said to be ancestor of node N2 if N1 is parent node of N2 where as N2 is said to be descendant of nodeN1.
- In the above figure, node A is the ancestor of node H and node H is the descendant of node A.

Differentiate between general tree and binary tree :



General tree	Binary tree
It has no limitation for child nodes.	It can have at most two child nodes.
A general tree can not be empty.	A binary tree can be empty.
In general tree, there is no limitation on the degree of a node.	In binary tree, there is limitation on the degree of a node.
In general tree, there is either zero subtree or many subtree.	While in binary tree, there are mainly two subtree: Left-subtree and Right-subtree.

Types of Tree in data structures :

1. General tree
2. Binary tree
3. Binary search tree
4. AVL tree(Balanced tree)

General tree :

- A general tree data structure has no restriction on the number of nodes.
- It means that a parent node can have any number of child nodes.

Binary tree :

- A node of a binary tree can have a maximum of two child nodes.

Binary search tree :

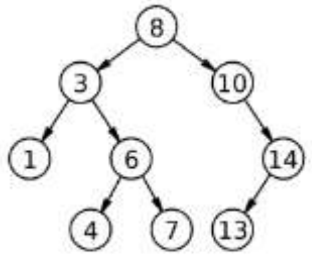
- It shows that the value of the left node is less than its parent, while the value of the right node is greater than its parent.

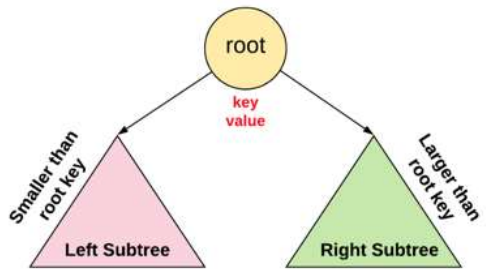
Balanced tree :

- If the height of the left sub-tree and the right sub-tree is equal or differs at most by 1, the tree is known as a balanced tree or AVL tree.

Binary tree :

- A binary tree is a special type of tree, which can be either empty or has finite set of nodes.
- Each node in binary tree is restricted to have at most two child node only.
- It is designed in such a way that one node as a root node and remaining nodes as partitioned into two sub tree of root node known as left subtree and right subtree.
- The non-empty left sub tree and right sub tree are also a binary trees.
- If the tree is empty, then the value of the root node is NULL.
- The worst case time complexity for search, insert and delete operations in BST is O(n).





Properties of Binary Tree :

- Each node in binary tree can have at most two child node only.
- At each level of i, the maximum number of nodes is 2^i.
- In general, the maximum number of nodes possible at height h = $\mathbf{2^{h+1} - 1}$.
- The minimum number of nodes possible at height h is equal to **h+1**.

Application of Binary Trees:

- Huffman coding
- In compilers, Expression
- Represent hierarchical data.
- Used in editing software like Microsoft Excel and spreadsheets.
- For implementing priority queues.
- Used to find elements in less time
- Used to enable fast memory allocation in computers.
- To perform encoding and decoding operations.

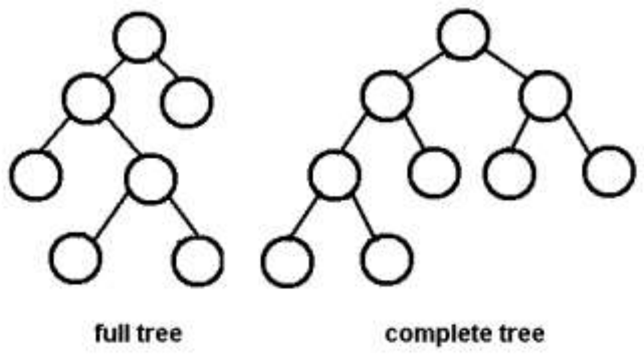
Types of Binary Tree :

There are four types of Binary tree :

1. Full/ proper/ strict Binary tree
2. Complete Binary tree
3. Perfect Binary tree
4. Degenerate Binary tree

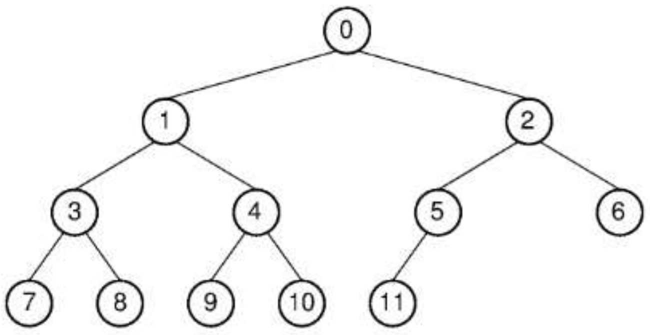
Full binary tree :

- The full binary tree is also known as a strict binary tree.
- In full binary tree each node must contain 2 children except the leaf nodes.



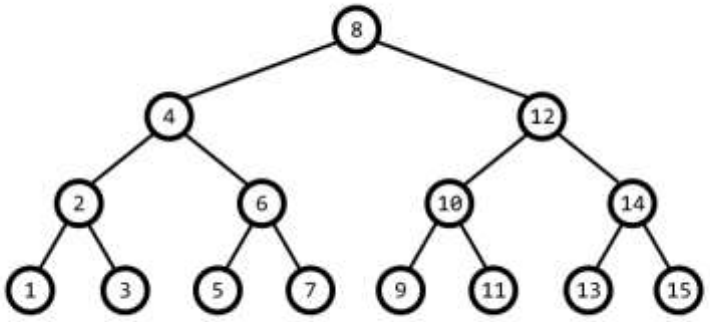
Complete Binary tree :

- The complete binary tree is a tree in which all the nodes are completely filled except the last level.
- In the last level, all the nodes must be as left as possible.
- In a complete binary tree, the nodes should be added from the left.



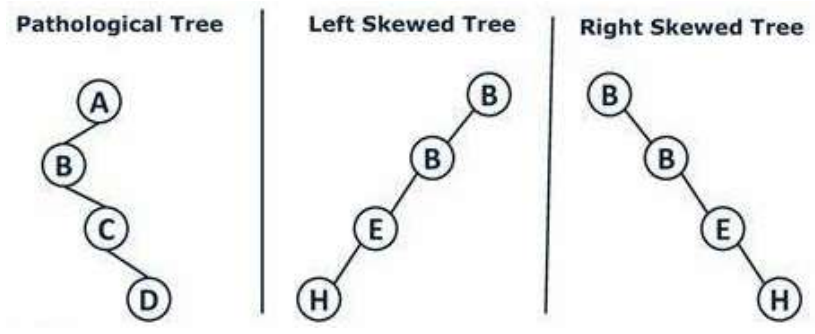
Perfect Binary Tree :

- A binary tree is a perfect binary tree if all the internal nodes have 2 children, and all the leaf nodes are at the same level.
- All the perfect binary tree is always full as well as complete binary tree.



Degenerate Binary Tree :

- The degenerate binary tree is a tree in which all the internal nodes have only one children.



Operations on Binary Tree :

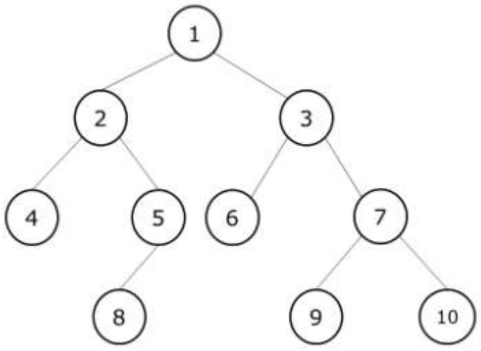
- Create : create a binary tree in data structure.
- Insert : Inserts data in a binary tree.
- Search : Searches specific data in a binary tree to check it is present or not.
- Traversal : The process of visiting each and every node of a binary tree exactly once.
- There are five types of traversal :
  1. Preorder Traversal
  2. In order Traversal
  3. Post order Traversal
  4. Depth first traversal(DFS)
  5. Breath first traversal(BFS) or Level order traversal

Preorder traversal :

- The preorder traversal technique follows the Root Left Right policy.
- The process of preorder traversal can be represented as :
  - ❖ Root → Left → Right
- The steps to perform the preorder traversal are as follows :
  - ➢ First, visit the root node.
  - ➢ Then, visit the left subtree.
  - ➢ At last, visit the right subtree.

Inorder traversal :

- The inorder traversal technique follows the Left Root Right policy.
- The process of inorder traversal can be represented as :
  - ❖ Left → Root → Right
- The steps to perform the inorder traversal are as follows :
  - ➢ First, visit the left subtree.
  - ➢ Then, visit the root node.
  - ➢ At last, visit the right subtree.



Preorder : 1,2,4,5,8,3,6,7,9,10

Inorder : 4,2,8,5,**1**,6,3,9,7,10

Postorder : 4,8,5,2,6,9,10,7,3,**1**

Postorder traversal :

- The postorder traversal technique follows the Left Right Root policy.
- The process of postorder traversal can be represented as :
    - ❖ Left → Right → Root
- The steps to perform the postorder traversal are as follows :
    - ➢ First, visit the left subtree.
    - ➢ Then, visit the right subtree.
    - ➢ At last, visit the root node.

Question : Create a binary tree using inorder and postorder traversal :

Inorder : D B H E A I F J C G

Postorder : D H E B I J F G C A

Binary Search Tree (BST):

- BST is a special type of binary tree in which left child of a node has value less than the parent and right child has value greater than parent.
- This rule is applicable on all the subsequent sub tree in a binary search tree.
- Each and every value in BST is unique, that is no two nodes have identical value.
- There are various operations that can be performed on the binary search trees.
- Some of these are searching, insertion of a new node traversal of tree and deletion of a node.
- The worst case time complexity for search, insert and delete operations in BST is O(n).

Ques : Create a binary tree from the following sequence :

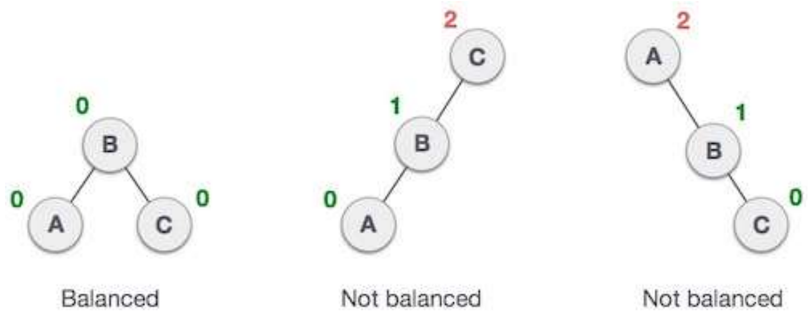
14, 34, 22, 44, 11, 24, 33

AVL/ Height Balanced Tree:

- AVL tree is binary search tree with additional property that difference between height of left sub-tree and right sub-tree of any node can't be more than 1.
- AVL Tree is invented by GM Adelson - Velsky and EM Landis in 1962.
- AVL Tree is also known as height balanced binary search tree in which each node is associated with a balance factor.
- Balance factor is calculated by subtracting the height of right sub-tree from height left sub-tree.
- Tree is said to be balanced if balance factor of each node is in between -1 to 1, otherwise, the tree will be unbalanced and need to be balanced.

Balance Factor = height of left sub tree - height of right sub tree.

- The worst case time complexity of AVL tree for insertion and deletion is $O(\log_2 n)$.



What is graph ?

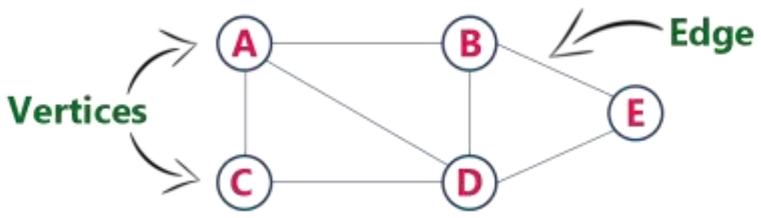
- A graph is a non-linear data structure consisting of vertices and edges.
- The vertices are called nodes and the edges are lines that connect any two nodes in the graph.
- In other words a graph is a set of vertices(**V**) and a set of edges(**E**).
- The graph is denoted by **G(V, E).**

Example :

The following is a graph with 5 vertices and 6 edges.

Where V = {A,B,C,D,E} and

E = {(A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}.



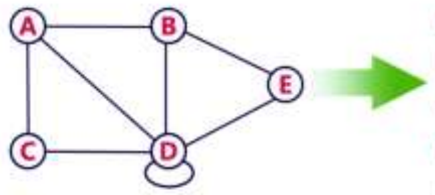
Representation of Graphs :

There are two ways to represents a graph:

1. Adjacency Matrix
2. Adjacency List

Adjacency Matrix :

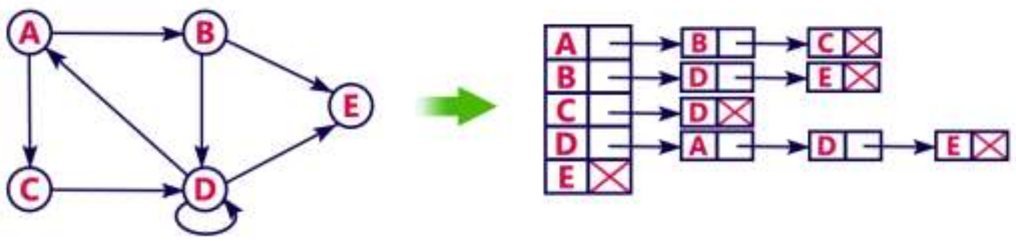
- In this method, the graph is representation in the form of the 2D matrix where rows and columns denote vertices.
- That means a graph with 4 vertices is represented using a matrix of size 4X4.
- In this matrix, both rows and columns represent vertices.
- This matrix is filled with either 1 or 0.
- Here, 1 represents an edge from row vertex to column vertex and 0 represents there is no edge from row vertex to column vertex.



$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 1 & 1 & 0 \\ B & 1 & 0 & 0 & 1 & 1 \\ C & 1 & 0 & 0 & 1 & 0 \\ D & 1 & 1 & 1 & 1 & 1 \\ E & 0 & 1 & 0 & 1 & 0 \end{array}$$

Adjacency List :

- In this method, The graph is represented as a collection of linked lists.
- In this representation, every vertex of a graph contains list of its adjacent vertices.
- There is an array of pointer which points to the edges connected to that vertex.



Action	Adjacency Matrix	Adjacency List
Adding Edge	O(1)	O(1)
Removing and edge	O(1)	O(N)