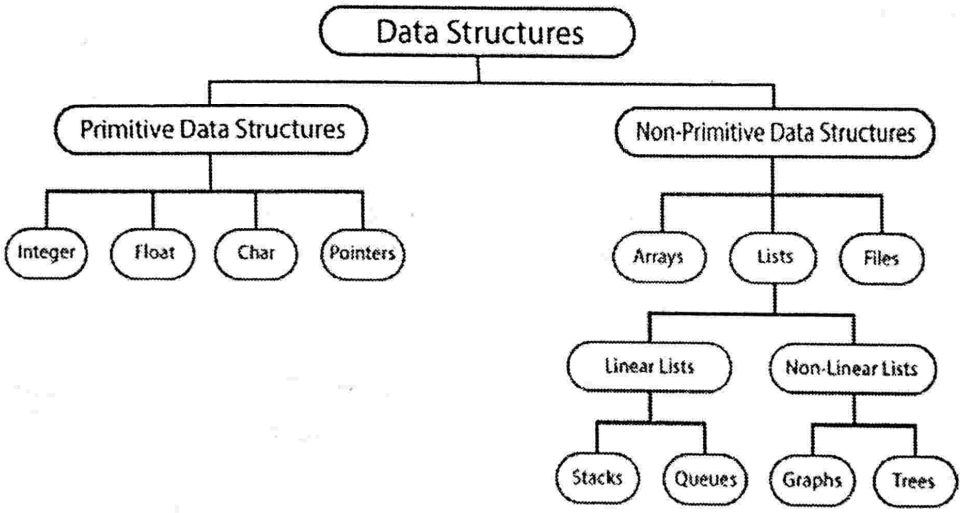
UNIT - 1

1Q) What is data structure? Explain Primitive and Non-Primitive Data Structures.

Ans) Data structure is an organized collection of related information. A data structure uses logical and mathematical techniques to store and retrieve data in an efficient manner. A data structure is the portion of memory allotted for a model, in which the required data can be arranged in a proper fashion.

The following figure shows primitive and non-primitive data structures.



2Q) Define ADT? Explain.

Ans) An ADT is a mathematical model of a data structure that specifies the type of data stored, the operations supported on them, and the types of parameters of the operations. An ADT specifies what each operation does. ADT acts as a useful guideline to implement a data type correctly.

Example:1) int data type in C Language.

The **int** data type provides an implementation of the mathematical concept of an integer number. It can be considered as implementation of Abstract data type, INTEGER-ADT.

2) Specification of Abstrct data type STACK is as follows:

abstract typedef <1,2,3 … n. top> STACK;
condition top=NULL;
abstract STACK PUSH(data);

3Q) Explain Data representation?

Ans)

The basic unit of data representation is a **bit**. It is either 0 or 1. The various combinations of these two values of a bit are used to represent data in different ways. The combination of eight bits is called a **byte**. One byte represents a **character**. The combination of one or more characters is called a **String**.

Examples:

bit : 0 or 1
byte : 00100110
character : 'A', 'B', '1', '2'
string : "STRING"

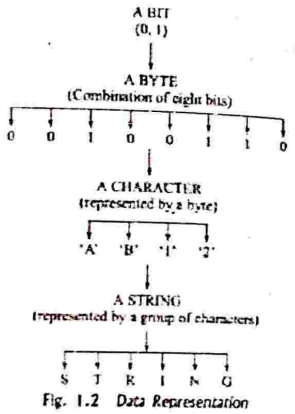


Fig. 1.2 *Data Representation*

4Q) Explain Data types (or) Primitive Data Types.

Ans) The following are different primitive data types.

i. int (integers): These are whole numbers with a range of value supported by a particular machine. 'C' has three classes of integer storage, namely int, short int and long int. in both signed and unsigned form.

Syntax : int var1,var2,var3;
Example : int a,b;

ii. float: By the key word float, which can also carry real parts, with size digits of precision.

Syntax : float var1,var2,var3;

Example : float a,b;

iii. double: When the accuracy provided by a float number is not sufficient, the data types double floating point can used to defined the number. A double data type numbers uses 8 bytes giving precision of 14 digits.

Syntax : double var1,var2,var3;
Example : double a,b;

iv. characters(char): A single character can be defined as a 'char' data type. Characters are usually stored in 1 Byte or internal storage. The qualifier signed or unsigned may be applied to character. While unsigned characters have values between 0 to 255 and signed characters have between –128 to 127.

Syntax : char var1,var2,var3;
Example : char a,b;

5Q) Write about a) Data Structure and Structured Type, b) Atomic Type, c) Difference between Abstract Data Types, Data Types, and Data Structures, d) Refinement Stages

Ans)

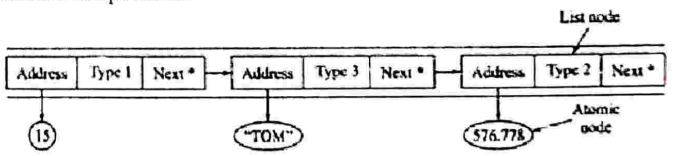
Data Structure and Structured Type:

A Data structure is a structural relationship present within the data set and which should be viewed as two tuple (N,R) where N s finite set of nodes representing the data structure and R is the set of relationship among those nodes.

A Structured type refers to a data structure which is made up of one or more elements known as components. These components are grouped together according to a set of rules.

Atomic Type:

Atomic type data is a data structure that contains only the data items and not the pointers.



In above address is atomic type data structure which contains data items 15, "TOM" and 576.778.

Difference between Abstract Data Types, Data Types, and Data Structures:

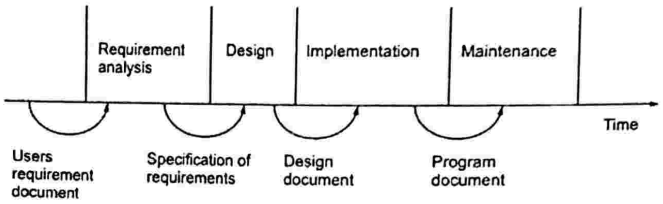
- An Abstract data type is the specification of the data type which specifies the logical and mathematical model of the data type.
- A data type is the implementation of an abstract data type.
- Data structure refers to the collection of computer variables that are connected in some specific manner.

Refinement Stages:

The best way to solve a complex problem is to divide it into smaller parts such that each part becomes an independent module which is easy to manage. Best example for this approach is SDLC (Software Development Life Cycle).

6Q) Write about Software Engineering.

Ans) Software engineering is the theory and practice of methods helpful for the construction and maintenance of large software systems. There are so many stages in the software development cycle. The process is referred to as Software Development Life Cycle (SDLC).



Software Development Life Cycle

The different steps in SDLC are as follows:

1. **Analyze** the problem precisely and completely.
2. **Build** a prototype and **experiment** with it until all specifications are finalized.
3. **Design** the algorithm using the tools of data structures.
4. **Verify** the algorithm, such that its correctness is self-evident.
5. **Analyze** the algorithm to determine its requirements.
6. **Code** the algorithm into an appropriate programming language.
7. **Test** and **evaluate** the program with carefully chosen data.
8. **Refine** and **repeat** the foregoing steps until the software is complete.
9. **Optimize** the code to improve performance.
10. **Maintain** the program so that it meets the changing needs of its users.

Program Design

Program design can be considered as an important phase of the software development life cycle. In this phase algorithms and data structures are proposed to solve the problems. Here the designer need to adopt a design strategy.

7Q) What is an Algorithm? Write Different Approaches to Designing an Algorithm.

Ans) An algorithm is a set of rules to follow for completing a specific task. It is also defined as a set of step-by-step procedures solving a particular problem.

Example: Algorithm to add 3 numbers and print their sum:

Step 1 – START
Step 2 – declare three integers a, b & c
Step 3 – define values of a & b
Step 4 – add values of a & b
Step 5 – store output of step 4 to c
Step 6 – print c
Step 7 – STOP

Different Approaches to Designing an Algorithm:

There are two approaches to designing an algorithm.

- Top-down approach
- Bottom-up approach

Top-down approach:

The top-down approach starts by identifying the major components of the program, decomposing them into their lower-level

components and iterating until the desired level of module complexity is achieved.

Bottom-up approach:

The bottom-up approach starts with designing primitive components and proceeds to higher level components. This method works with layers of abstraction.

8Q) What is Complexity? Explain its types.

Ans) The time or space requirements for solving a problem by a particular algorithm is called complexity. These requirements are expressed in terms of a single parameter that represents the size of the problem. There are two types of complexity.

- Time complexity
- Space complexity

The time complexity of a program/algorithm is the amount of computer time that it needs to run to completion.

The space complexity of a program/ algorithm is the amount of memory that it needs to run to completion.

9Q)Write about Big 'O' Notation

Ans)

If f(n) represents the computing time of some algorithm and g(n) represents a known standard function like n, n^2, nlog n, etc., then to write f(n) is O g(n). i.e. f(n) of n is equal to biggest order of function g(n).

$$|f(n)|<=C|g(n)|$$

From the above statements, we can say that the computing time of an algorithm is O(g(n)).

Big O notation helps to determine the time as well as space complexity of the algorithm. The Big 'O' notation has been extremely useful to classify algorithms by their performances. Developers use this notation to reach to the best solution for the given problem.

The most common computing times of algorithms in the order of performance are as follows. O(1), O(log n),O(n), O(nlog n), $O(n^2)$, $O(n^3)$, $O(2^n)$

10Q) Explain Algorithm Analysis.

Ans) Algorithm analysis provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

The following are different types of analysis:

- **Worst-case** – The maximum number of steps taken on any instance of size a.
- **Best-case** – The minimum number of steps taken on any instance of size a.
- **Average case** – An average number of steps taken on any instance of size a.

11Q) Write about Structured Programming Approch?

Ans)

It is a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction one after the other. It doesn't support the possibility of jumping from one instruction to some other with the help of any statement. Therefore, the instructions in this approach will be executed in a serial and structured manner. The languages that support Structured programming approach are:

- C
- C++
- Java
- C#

Advantages of Structured Programming Approach:

- Easier to read and understand
- User Friendly
- Easier to Maintain
- Mainly problem based instead of being machine based
- Development is easier as it requires less effort and time
- Easier to Debug
- Machine-Independent, mostly.

Disadvantages of Structured Programming Approach:

- Since it is Machine-Independent, So it takes time to convert into machine code.

- The converted machine code is not the same as for assembly language.

12Q) Explain Recursion.

Ans) A recursive routine is one whose design includes a call to itself. In design phase of software development we use various problem solving methods in which recursion can be one of the powerful tools. The C programming language supports recursion, i.e., a function to call itself.

Example 1:

```
void main()
{
    main();   // recursive call
}
```

Example 2:

```
factorial(int a)
{
    int fact=1;
    if(a>1)
    fact=a*factorial (a-1);
    return fact;
}
```

13Q) Write about Tips and Techniques for Writing Programs in 'C'.

Ans) C program can be viewed as a group of built in blocks called functions. A function is a subprogram that may be include of one or more statements designed to perform a specific task. "C" programs make contain one or more sections.

Documentation Section

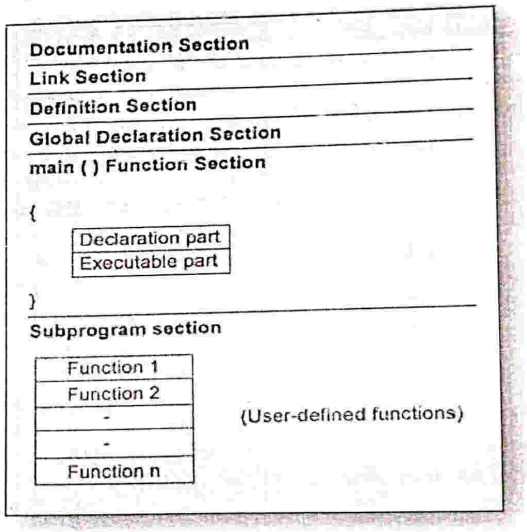
The documentation section consist of a set of comment lines giving the name of program, the author and the other details which the programmer would like to use later. It is optional depend by user.

Example : /* The program name, author name */

Link Section

The link section provides instructions to the compiler to link functions or built in programs from the C library. The C library is a set of header files. A header file is a collection of predefined programs developed for predefined functions.

Example : # include <header file .h>



Define Section

The definition section defines all symbolic constants.

Example :

1. # Define var value; (var = variable)
2. # Define pi 3.141;

External or Global declaration Section

There are some variables that are used in more than one function, such variable are called global variables.

Main function Section

Every C program must here one main function section. This section contains two parts. They are,

1. Declaration part
2. Executable part

The declaration part declares all the variable s used in executable part. These two parts must appear between the opening and the closing braces. The closing of main bracket is logical end of the program. All statements in declaration and executable parts end with a semicolon [:].

User defined function (or) Sub program section

```
function 1 ( )
{
----------
----------
}
```


UNIT – 2

1Q) Explain Types of Data Structures?
(Or) Explain Linear and Non linear Data Structures

Ans)

Types of Data Structures

Generally data structures are classified into two classes:

- Linear Data structures
- Non-linear data structures

Linear Data structures

Linear data structures, in which elements are organized in sequence order. The following are various linear data structures:

- Arrays
- Linked lists
- Stacks
- Queues

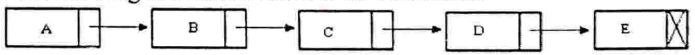
Arrays

An array is a collection of ordered elements of same type. (OR) Array is a collection of subscripted variables of the same type. Array is finite because it contains only limited number of elements.

Linked lists

A linked list is an ordered collection of nodes. Node is a data structures, which contains two parts called **data** and **link.** DATA part is used to store element information and LINK part is used to store address of the next node.

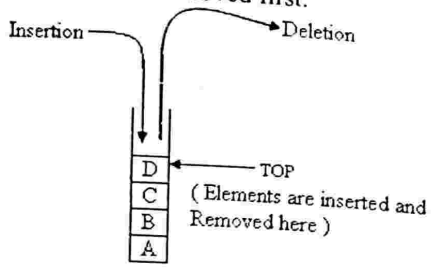
The following is a linked list with five elements:



Stacks

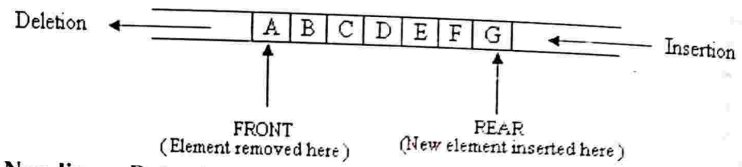
A Stack is a linear data structure, in which elements are inserted and deleted at one end called TOP. TOP is a pointer always pointing to

top of the stack. Stack is also called LIFO (Last-In First-Out) list, because last inserted element removed first.



Queues

A Queue is a linear data structure, in which elements are inserted at one end called REAR and deleted at one end called FRONT. Queue is also called FIFO (First-In First-Out) list, because first inserted element removed first.



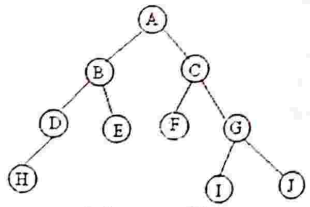
Non-linear Data structures

A Non linear data structure is a data structure, in which elements are organized but not ordered in one particular way. The following are the non-linear data structures:

- Trees
- Graphs

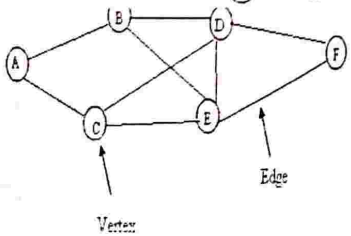
Trees

A tree is a non-linear data structure, in which elements are organized in hierarchical manner.



Graphs

A graph is a set of vertices and edges. Here vertex is a node and edge connects two vertices.



Formally, a graph G = (V, E)
V = Set of vertices and E = Set of edges
For example, consider the following un-directed graph:

2Q) Explain Arrays?

Ans)

An Array is a collection of values of the same data type. Always, Contiguous (adjacent) memory locations are used to store array elements in memory. It is a best practice to initialize an array to zero or null while declaring, if we don't assign any values to array.

Types of arrays:

There are 2 types of C arrays. They are,

1. One dimensional array
2. Two dimensional array (Multi dimensional array)

1. One dimensional array

A one dimensional array contains only on subscript(size of array) . The subscript is also called index. It starts from zero (0) and ends with size-1;

Syntax : data-type array_name[array_size];
Here Array size must be a constant value.

Example: int a[10];

In the above example variable 'a' can stores 10 integer values.

Array initialization

Syntax: data_type arr_name [arr_size]={value1, value2, value3,....};

Example: int a[5]={1,2,3,4,5};
char str[10]={'A','T','P'}

Accessing array

Syntax: arr_name[index];

Example: int a[3];
a[0]=1;
a[1]=2;
a[2]=3;

2. Two dimensional array

Two dimensional arrays is nothing but array which contains number of rows and number of columns.

Syntax : data_type array_name[row size][column size];

Example: int a[2][2];

In the above example variable 'a' can stores 4 integer values.

Array initialization

Syntax: data_type arr_name[row size][column size] = {val1,val2,val3,val4……..};

Example: int a[2][2] – {1,2, 3, 4};

Accessing array

Syntax: arr_name[row index][column index]=value;

In the above syntax, row index starts from 0 and ends with row size-1. Column index starts from 0 and ends with column size-1.

Example: int a[2][2];

a[0] [0] = 1;

a[0]]1] = 2;

a[1][0] = 3;

a[1] [1] = 4;

Basic Operations

Following are the basic operations supported by an array.

Traverse – print all the array elements one by one.

Insertion – Adds an element at the given index.

Deletion – Deletes an element at the given index.

Search – Searches an element using the given index or by the value.

Update – Updates an element at the given index.

3Q) Explain Pointers (or) Write about Pointer and Arrays.

Ans) Pointer is a variable that stores or points the address of another variable. Pointer is used to allocate memory dynamically i.e. at run time. The variable might be any of the data type such as int, float, char, double, short etc.

Syntax : data_type *var_name;

Example : int *p;

char *p;

Here, * is used to denote that "p" is pointer variable and not a normal variable.

Pointer and Arrays

Arrays are closely related to pointers in C programming but the important difference between them is that, a pointer variable can take different addresses as value whereas; in case of array it is fixed.

Example : char *x[20];

Program : Lab Program-1

4Q)Explain Dynamic memory allocation in C? (or) Explain Memory allocation in C?

Ans) Sometimes the size of the array declared in a program may be insufficient. To solve this problem, we can allocate memory manually at run-time. This is known as dynamic memory allocation in C programming.

Pointers play an important role in dynamic memory allocation in C programming because we can access the dynamically allocated memory only through pointers.

There are 4 library functions provided by C defined under <stdlib.h> header file to facilitate dynamic memory allocation in C programming. They are: malloc(), calloc(), free() and realloc().

1. malloc()

The name "malloc" stands for memory allocation. It allocates a single large block of memory with the specified size.

Syntax : ptr = (cast_type *) malloc (byte_size);

Example: ptr = (float*) malloc(100 * sizeof(float));

Here it allocates 400 bytes of memory because the size of float is 4 bytes. And, the pointer ptr holds the address of the first byte in the allocated memory.

2. calloc()

The name "calloc" stands for contiguous allocation. The malloc() function allocates memory and leaves the memory uninitialized, whereas the calloc() function allocates memory and initializes all bits to zero.

Syntax: ptr = (cast-type*)calloc(n, element-size);

Example: ptr – (float*) calloc(25, sizeof(float));

Here it allocates contiguous space in memory for 25 elements of type float.

3. free()

It is used to de-allocate the memory dynamically. The memory allocated using functions malloc() and calloc() is not de-allocated on their own. Hence the free() method is used.

Syntax: free(ptr);

4. realloc()

If the dynamically allocated memory is insufficient or more than is required, we can change the size of previously allocated memory using the realloc() function.

Syntax: ptr = realloc(ptr, newSize);

Example: ptr = (int*) malloc(10 * sizeof(int));
ptr = realloc(ptr, 25 * sizeof(int));

5Q) Explain about Linked List? And Types of Linked List?

Ans) Linked lists

A linked list is an ordered collection of nodes. Node is a data structures, which contains two parts called **data** and **link.**

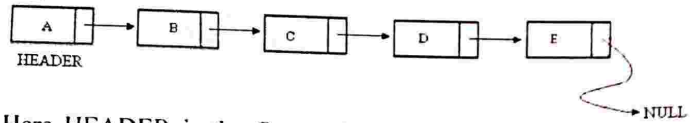
DATA part is used to store element information and LINK part is used to store address of the next node.

Generally, there are three types linked lists:

- Single linked lists
- Circular linked lists
- Doubly linked lists

Single linked lists

A single linked list is a linked list, in which each node contains one data part and one link part. The following figure shows single linked list:



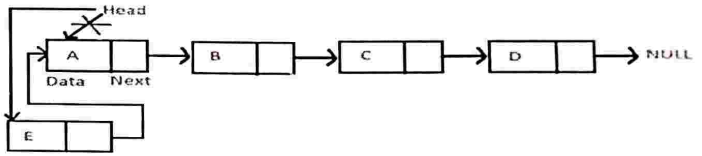
Here HEADER is the first node in the linked list and last node is pointing to NULL. Without HEADER it is not possible to identify the

beginning of the linked list and the same way without NULL it is not possible to identify the end of the linked list

Linked list ADT

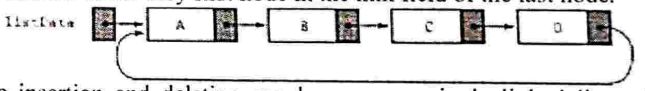
```
struct LinkedList
{
    int data;
    struct LinkedList *next;
};
```

Add a node at the front



Circular Linked List:

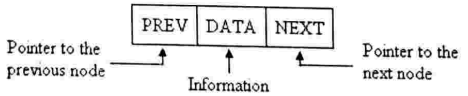
A **Circular Linked list** is one which has no beginning and no end. A Singly Linked list can be made a circular linked list by simply storing the address of the very first node in the link field of the last node.



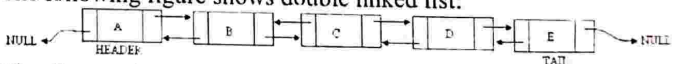
The insertion and deletion can be same as single linked list, with appropriate changes made in the program code.

Doubly linked lists

A double linked list is a linked list, in which each node contains one data part and two link parts, one link pointing to next node and other pointing to previous node. The following figure shows double linked list node:



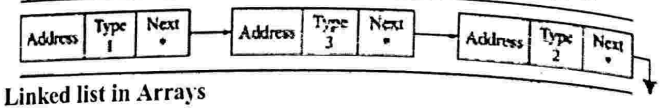
The following figure shows double linked list:



The first node PREV pointer and last node NEXT pointer always pointing to NULL, indicated no previous node and no next node.

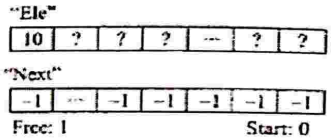
Atomic Node Linked List

An atomic data type contains only the data items and not the pointers. For a list of data items several atomic type nodes may exist, each with a single data item corresponding to one of the legal data types.



Linked list in Arrays

Linked list can be implemented without using pointers. For example, let an array Ele=(10,2,3,6); this list can be stored in and array "Ele". The link can be implemented by using another array "Next". The ith elements of array Ele is guaranteed to be stored in the ith index of an array "Next".

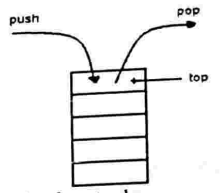


Linked List versus Arrays

Array	Linked List
Array is a collection of elements having same data type with common name.	Linked list is an ordered collection of elements which are connected by links.
Elements can be accessed randomly.	Elements cannot be accessed randomly. It can be accessed only sequentially.
Array elements can be stored in consecutive manner in memory.	Linked list elements can be stored at any available place as address of node is stored in previous node.
Insert and delete operation takes more time in array.	Insert and delete operation cannot take more time. It performs operation in fast and in easy way.
Memory is allocated at compile time.	Memory is allocated at run time.
It can be single dimensional, two dimensional or multidimensional.	It can be singly, doubly or circular linked list.
Each array element is independent and does not have a connection with previous element or with its location.	Location or address of element is stored in the link part of previous element or node.
Array elements cannot be added, deleted once it is declared.	The nodes in the linked list can be added and deleted from the list.

UNIT – 3

1Q) Define Stack? Explain.



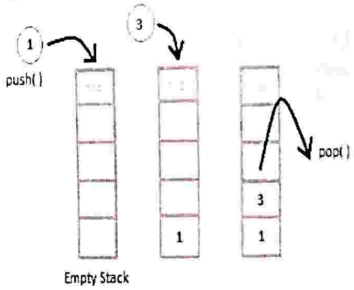
Ans) A stack is a container of elements that are inserted and removed according to the last-in first-out (LIFO) principle. A stack is a limited access data structure - elements can be added and removed from the stack only at the top

The following two basic operations are performed in the stack:

- **Push:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
- **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.

Stack ADT

```
struct Stack
{
   static final int MAX – 1000;
   int top;
   int s[MAX];
};
```

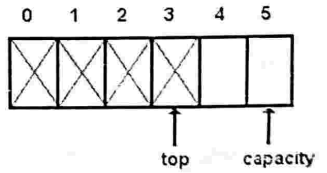


Implementation of Stack

Stack can be easily implemented using an Array or a Linked List.

Array-based representation

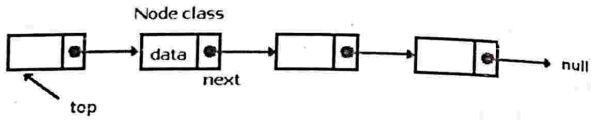
In an array-based implementation we maintain the following fields: an array A of a default size (≥ 1), the variable *top* that refers to the top element in the stack and the



capacity that refers to the array size. The variable *top* changes from -1 to capacity - 1. We say that a stack is empty when top = -1, and the stack is full when top = capacity-1.

Linked List-based representation

Linked List-based implementation provides the best (from the efficiency point of view) dynamic stack implementation.



Applications of stack:

- Balancing of symbols
- Infix to Postfix /Prefix conversion
- Redo-undo features at many places like editors, Photoshop.
- Forward and backward feature in web browsers
- Used in many algorithms like Tower of Hanoi, tree traversals, histogram problem, etc.

2Q) Explain Array Implementation of Stack. (or) Explain Array based representation of Stack

Ans) Array based representation: see Question 1.
Program : Lab Program-2

3Q) Explain Linked List Implementation of Stack. (or) Explain Linked List based representation of Stack

Ans) Linked List based representation: see Question 1.
Program : Lab Program-3.

4Q) Explain about Queues.

Ans) A queue is a container of objects (a linear collection) that are inserted and removed according to the first-in first-out (FIFO) principle. In Queue the first element is inserted from one end called **REAR**(also called tail), and the deletion of existing element takes place from the other end called as **FRONT**(also called head).
The following are the Queue operations:

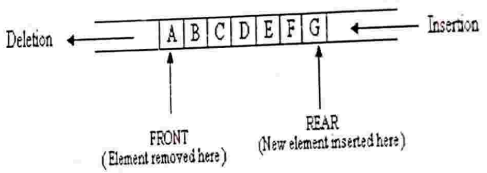
- Insertion
- Deletion

Insertion

Inserting a new element into the queue at REAR position is called insertion. It is also called enqueue. New item inserted into the queue only when REAR is not equal to MAXSIZE. If REAR equal with MAXSIZE then insertion not possible, because queue is full.

Deletion

Removing the existing element from the queue at FRONT position is called deletion. It is also called dequeue. Item removed from the queue only when FRONT is not equal to 0. If FRONT is equal to 0, then queue is empty. So, removing the existing item from the queue is not possible.



Queue ADT

```
struct Queue
{
 int q[100];
 int rear, int front;
}
```

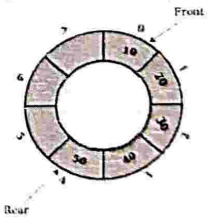
Applications of Queue Data Structure

1) When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.

2) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

Circular Queue :

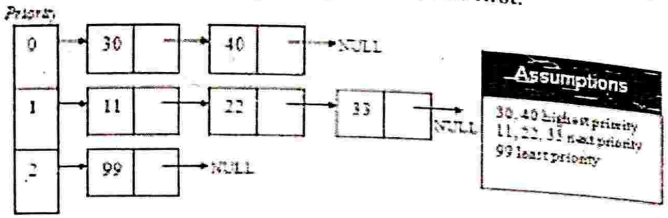
Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position



to make a circle. It is also called **'Ring Buffer'**

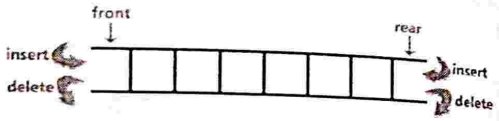
Priority Queue:

A priority queue is a queue that contains items that have some preset priority. When an element has to be removed from a priority queue, the item with the highest priority is removed first.



Dequeue (Double Ended queue):

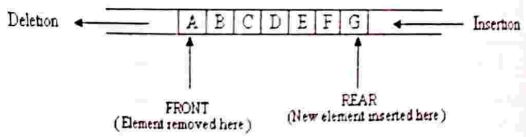
Double Ended Queue is also a Queue data structure in which the insertion and deletion operations are performed at both the ends (**front** and **rear**). That means, we can insert at both front and rear positions and can delete from both front and rear positions.



5Q) Explain Array based representation of Queue. (or) Explain Array based implementation of Queue.

Ans)

The implementation of queue data structure using array is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using **FIFO (First In First Out) principle** with the help of variables **'front'** and **'rear'**. Initially both **'front'** and **'rear'** are set to -1. Whenever, we want to insert a new value into the queue, increment **'rear'** value by one and then insert at that position. Whenever we want to delete a value from the

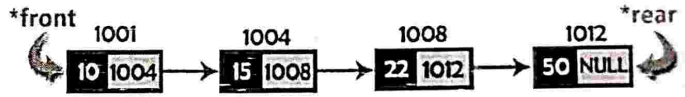


queue, then increment 'front' value by one and then display the value at **'front'** position as deleted element.

Example : Lab Program-4

6Q) Explain Linked List Implementation of Queue. (or) Explain Linked List based representation of Queue.

Ans) In linked list implementation of a queue, the last inserted node is always pointed by **'rear'** and the first node is always pointed by **'front'**.



Example : Lab Program-5

UNIT – 4

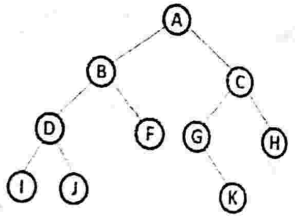
1Q) Explain Trees.(or) Explain Basic Definition of Binary Trees (or) Explain Properties of Binary Trees

Ans) Tree is a non-linear data structure which organizes data in hierarchical structure.

Tree Terminology

Root: A root is the special node, which is used to identify the tree. There is only path from root node to every node. Example : A

Node: Node is used to store tree element.



Parent: Immediate predecessor of a node. Every node in the tree has a parent except to the root node. Example : A, B, C, D, F, G, H.

Child: Immediate successor of a node.

Example : B,C,D,E,F,G,H,I,J,K

Siblings: Child nodes with same parent.

Example : BC,DF,GH,IJ

Leaf node: A node which has no child nodes.

Example : F,H,I,J,K

Degree: Maximum number of child nodes to a node. Example : 2

Path: A path is the sequence of nodes which connected with edge.

Edge: Edge connects two nodes. Edge is also called link.

Level: Length of the path from root node to the node. Root has level 0.

Height: Maximum number of nodes possible from root node to leaf node.

2Q) Explain Binary Tree.

Ans) In a normal tree, every node can have any number of children. Binary tree is a special type of tree data structure in which every node can have a **maximum of 2 children**. One is known as left child and

the other is known as right child. In a binary tree, every node can have either 0 children or 1 child or 2 children but not more than 2 children.

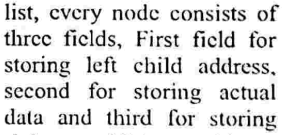
Array Representation

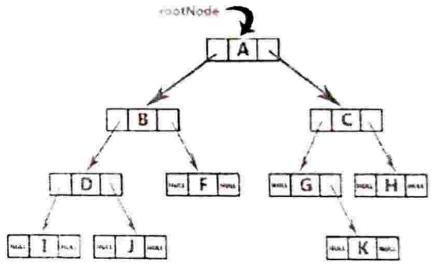
In array representation of binary tree, we use a one dimensional array (1-D Array) to represent a binary tree. Consider the above example of binary tree and it is represented as follows...

A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

Linked List Representation

We use double linked list to represent a binary tree. In a double linked list, every node consists of three fields, First field for storing left child address, second for storing actual data and third for storing right child address. In this linked list representation, a node has the following structure...



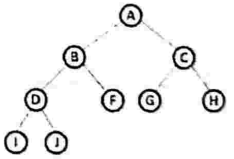


The above example of binary tree represented using Linked list representation is shown as follows...

There are different types of binary trees and they are.

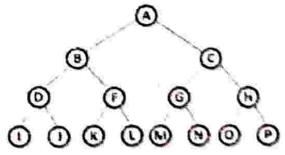
Strictly Binary Tree

A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree. Strictly binary tree is also called as **Full Binary Tree** or **Proper Binary Tree** or **2-Tree**



Complete Binary Tree

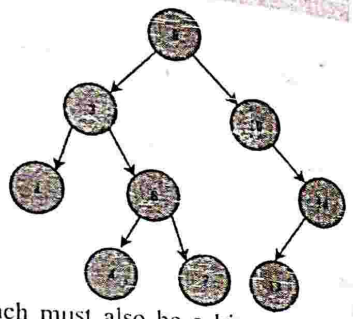
A binary tree in which every internal node has exactly two children and all leaf nodes are at same level is called



Complete Binary Tree. Complete binary tree is also called as **Perfect Binary Tree.**

3Q) Explain BST. (or) Explain Binary Search Trees.(or) Explain operations on Binary Search Trees.

Ans) Binary Search Tree, is a node-based binary tree data structure which has the following properties:



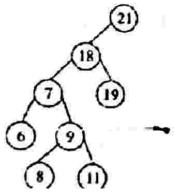
- The left sub tree of a node contains only nodes with keys less than the node's key.
- The right sub tree of a node contains only nodes with keys greater than the node's key.
- The left and right sub tree each must also be a binary search tree.

There must be no duplicate nodes.

The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search a given key.

Searching a key

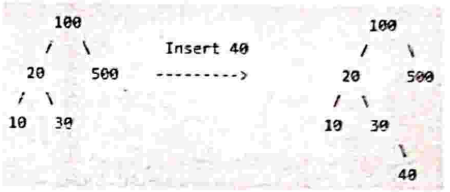
To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root's key, we proceed for right sub tree of root node. Otherwise we proceed for left sub tree. Same procedure is repeated for left and right sub trees until the data is found.



While searching the data if the leaf node of tree is reached and data is not found then it is concluded that the data is not present in the tree.

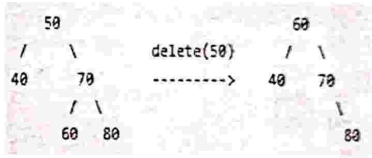
Insertion of a key

A new key is always inserted at leaf. We start searching a key from root till we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.



Deletion of a key

Find in-order successor of the node. Copy contents of the in-order successor to the node and delete the in-order successor. Note that in-order predecessor can also be used.



4Q) Explain Binary tree traversal.

Ans) Binary tree traversal means traversing or visiting each node of a binary tree. There are three types of binary tree traversals.

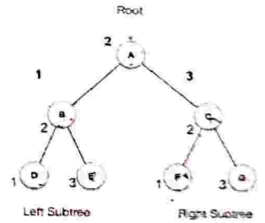
- In - Order Traversal
- Pre - Order Traversal
- Post - Order Traversal

In-order Traversal

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree.

Algorithm In-order

1. Traverse the left subtree
2. Visit the root.
3. Traverse the right subtree



$D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$

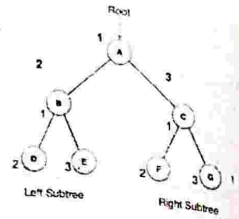
Pre-order Traversal

In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.

Algorithm Pre-order

1. Visit the root.
2. Traverse the left subtree.
3. Traverse the right subtree.

$$A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$$



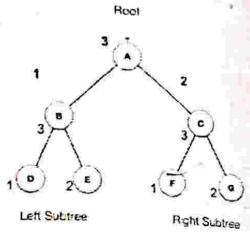
Post-order Traversal

In this traversal method, the root node is visited last. First we traverse the left subtree, then the right subtree and finally the root node.

Algorithm Post-order

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the root.

$$D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$$



5Q) Write Applications of Binary Tree.

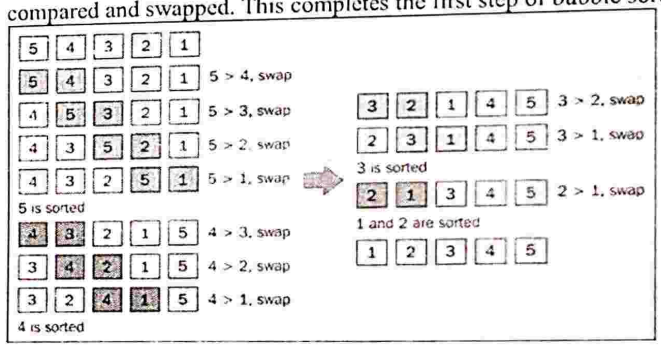
Ans) Following are the Applications of Binary Tree:

- Binary Tree is used to as the basic data structure in Microsoft Excel and spreadsheets in usual.
- Binary Tree is used to implement indexing of Segmented Database.
- Binary Tree is used to compute arithmetic expressions in compilers like GCC, AOCL and others.
- Binary Tree is used to implement Priority Queue efficiently which in turn is used in Heap Sort Algorithm.
- Binary Search Tree is used to search elements efficiently and used as a collision handling technique in Hash Map implementations.
- Balanced Binary Search Tree is used to represent memory to enable fast memory allocation.

UNIT – 5

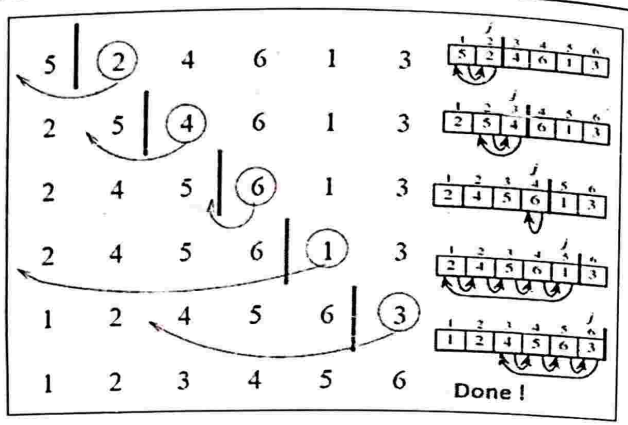
1Q) Explain Bubble Sort.

Ans) Bubble sort algorithm starts by comparing the first two elements of an array and swapping if necessary, i.e., if you want to sort the elements of array in ascending order and if the first element is greater than second then, you need to swap the elements but, if the first element is smaller than second, you mustn't swap the element. Then, again second and third elements are compared and swapped if it is necessary and this process go on until last and second last element is compared and swapped. This completes the first step of bubble sort.



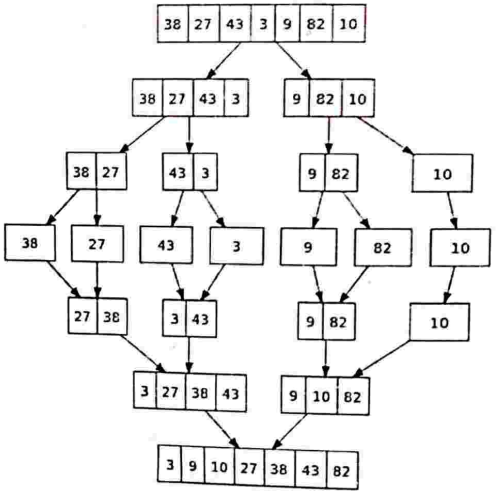
2Q) Explain Insertion Sort.

Ans) Sorting is the process of arranging a list of elements in a particular order (Ascending or Descending). Insertion sort algorithm arranges a list of elements in a particular order. In insertion sort algorithm, every iteration moves an element from unsorted portion to sorted portion until all the elements are sorted in the list.



3Q) Explain Merge Sort.

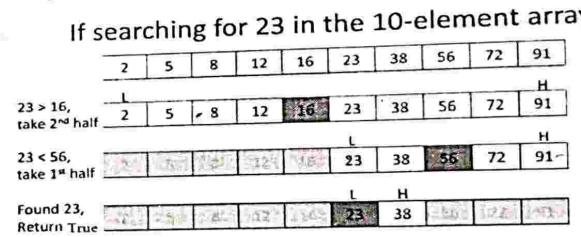
Ans) Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves



4Q) Explain Binary Searching.

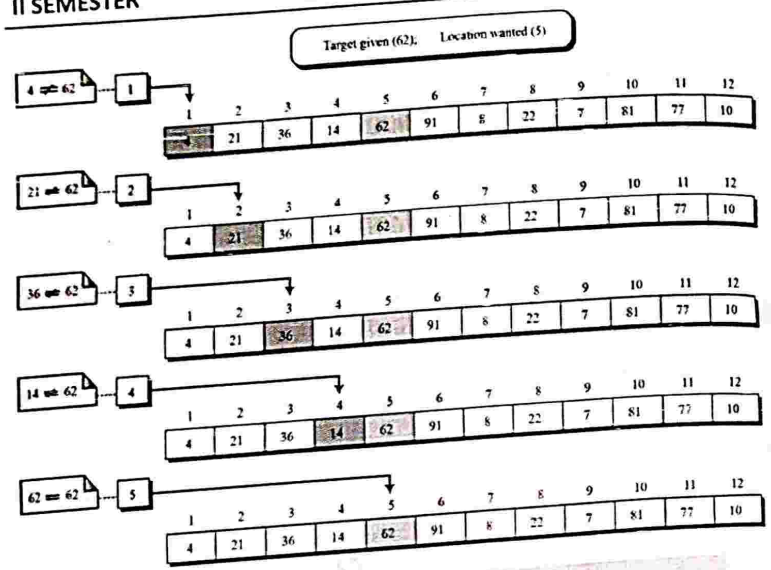
Ans) Binary search algorithm finds given element in a list of elements with **O(log n)** time complexity where **n** is total number of elements in the list. The binary search algorithm can be used with only sorted list of element. This search process starts comparing of the search element with the middle element in the list. If both are matched, then the result is "element found". Otherwise, we check whether the search element is smaller or larger than the middle element in the list. If the search element is smaller, then we repeat the same process for left sub list of the middle element. If the search element is larger, then we repeat the same process for right sub list of the middle element. We repeat this process until we find the search element in the list or until we left with a sub list of only one element. And if that element also doesn't match with the search element, then the result is "Element not found in the list".

Example :



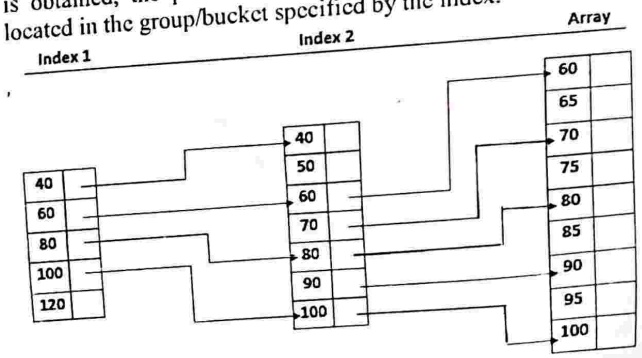
5Q) Explain Sequential Searching.

Ans) Search is a process of finding a value in a list of values. Linear search algorithm finds given element in a list of elements with **O(n)** time complexity where **n** is total number of elements in the list. This search process starts comparing of search element with the first element in the list.



6Q) Explain Index sequential searching.

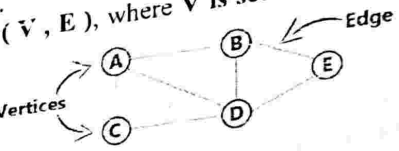
Ans) In Index sequential searching technique, first of all an index file is created that contains references to a group of records, once an index is obtained, the partial searching takes less time since it is to be located in the group/bucket specified by the index.



7Q) Define Graph? Explain its representation.

Ans) Graph is a non linear data structure; it contains a set of points known as nodes (or vertices) and set of links known as edges (or Arcs) which connects the vertices.

A graph **G** is represented as **G = (V , E)**, where **V is set of vertices and E is set of edges.**

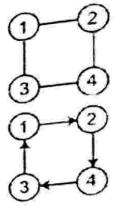


Example

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

This graph G can be defined as G = (V , E)

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



Undirected Graph

A graph with only undirected edges is said to be undirected graph.

Directed Graph

A graph with only directed edges is said to be directed graph.

Graph Terminology

Vertex

An individual data element of a graph is called as Vertex. **Vertex** is also known as **node**. In above example graph, A, B, C, D & E are known as vertices.

Edge

An edge is a connecting link between two vertices. **Edge** is also known as **Arc**. An edge is represented as (startingVertex, endingVertex). For example, in above graph, the link between vertices A and B is represented as (A,B). In above example graph, there are 7 edges (i.e., (A,B), (A,C), (A,D), (B,D), (B,E), (C,D), (D,E)).

Digraph: A digraph is the directed graph.

Path: A path is the sequence of edges from one vertex to another.

Degree of vertex: The number of vertices connected to this vertex.

Weighted graph: A weight(or Value) is assigned to the each edge of a graph.

8Q) Explain Graph Traversal. (or) Explain basic searching techniques of Graph. (or) Write about Spanning Tree(or) Write about Shortest path

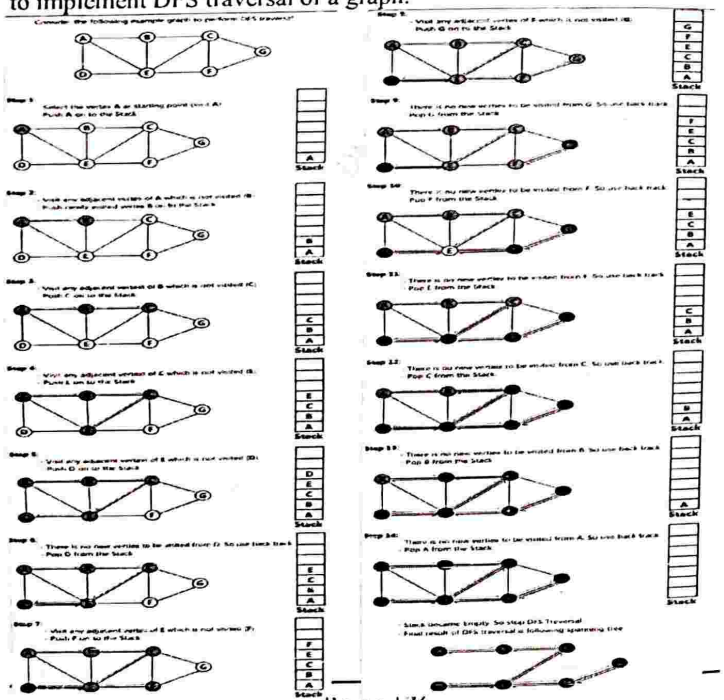
Ans) Graph traversal is technique used for searching a vertex in a graph. The graph traversal is also used to decide the order of vertices to be visit in the search process.

here are two graph traversal techniques and they are as follows...

1. **DFS (Depth First Search)**
2. **BFS (Breadth First Search)**

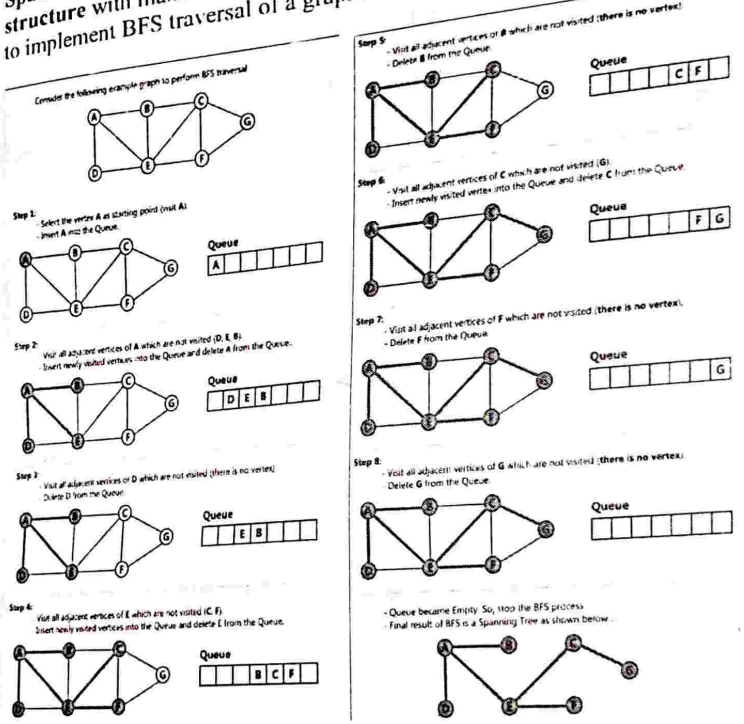
DFS (Depth First Search)

DFS traversal of a graph, produces a **spanning tree** as final result. **Spanning Tree** is a graph without any loops. We use **Stack data structure** with maximum size of total number of vertices in the graph to implement DFS traversal of a graph.



BFS (Breadth First Search)

BFS traversal of a graph, produces a **spanning tree** as final result. **Spanning Tree** is a graph without any loops. We use **Queue data structure** with maximum size of total number of vertices in the graph to implement BFS traversal of a graph.



LAB PROGRAM - 1

Q) Write a C-Program for sorting strings using pointers.

See Lab Manual : page 51-Lab Program -14

LAB PROGRAM - 2

Q) Write a C program to implement stack operations using Array.

Aim : To implement Stack using Array.

Program :

```
#include<stdio.h>
#include<conio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
  clrscr();
  top=-1;
  printf("\n Enter the size of STACK[MAX=100]:");
  scanf("%d",&n);
  printf("\n\t STACK OPERATIONS USING ARRAY");
  printf("\n\t----------------------------");
  printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
  do
  {
    printf("\n Enter the Choice:");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
      {
        push();
        break;
      }
      case 2:
```



```
      {
        pop();
        break;
      }
      case 3:
      {
        display();
        break;
      }
      case 4:
      {
        printf("\n\t EXIT POINT ");
        break;
      }
      default:
      {
        printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
      }

    }
  }
  while(choice!=4);
  return 0;
}
void push()
{
  if(top>=n-1)
  {
    printf("\n\tSTACK is over flow");

  }
  else
  {
    printf(" Enter a value to be pushed:");
    scanf("%d",&x);
    top++;
    stack[top]=x;
```

```
    }
}
void pop()
{
   if(top<=-1)
   {
      printf("\n\t Stack is under flow");
   }
   else
   {
      printf("\n\t The popped elements is %d",stack[top]);
      top--;
   }
}
void display()
{
   if(top>=0)
   {
      printf("\n The elements in STACK \n");
      for(i=top; i>=0; i--)
         printf("\n%d",stack[i]);
      printf("\n Press Next Choice");
   }
   else
   {
      printf("\n The STACK is empty");
   }

}
```

Test data and Output

```
Enter the size of STACK[MAX=100]:10

      STACK OPERATIONS USING ARRAY
      ----------------------------------------------
      1.PUSH
      2.POP
```



```
      3.DISPLAY
      4.EXIT
Enter the Choice:1
Enter a value to be pushed:12

Enter the Choice:1
Enter a value to be pushed:24

Enter the Choice:1
Enter a value to be pushed:98

Enter the Choice:3

The elements in STACK

98
24
12
Press Next Choice
Enter the Choice:2

      The popped elements is 98
Enter the Choice:3

The elements in STACK

24
12
Press Next Choice
Enter the Choice:4

      EXIT POINT
```

LAB PROGRAM - 3

Q) Write a C program to implement stack operations using Linked list.

Aim : To implement Stack using Linked List.

Program :

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head = NULL;
void push(int val)
{
   struct node *newNode = malloc(sizeof(struct node));
   newNode->data = val;
   newNode->next = head;
   head = newNode;
}
void pop()
{
   struct node *temp;

   if(head == NULL)
      printf("Stack is Empty\n");
   else
   {
      printf("Poped element = %d\n", head->data);
      temp = head;
      head = head->next;
      free(temp);
   }
}
void printList()
{
```



```
   struct node *temp = head;
   while(temp != NULL)
   {
      printf("%d->", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}

int main()
{
   push(10);
   push(20);
   push(30);
   printf("Linked List\n");
   printList();
   pop();
   printf("After the pop, the new linked list\n");
   printList();
   pop();
   printf("After the pop, the new linked list\n");
   printList();
   return 0;
}
```

Test data and Output

```
Linked List
10 -> 20 -> 30
Poped element =30
After the pop, the new linked list
10 -> 20
Poped element =20
After the pop, the new linked list
10
```

LAB PROGRAM - 4

Q) Write a C program to implement Queue operations using Arrays.

Aim: To implement Queue operations using Arrays.

Program:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10
void enQueue(int);
void deQueue();
void display();
int queue[SIZE], front = -1, rear = -1;
void main()
{
  int value, choice;
  clrscr();
  while(1){
    printf("\n\n***** MENU *****\n");
    printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice){
      case 1: printf("Enter the value to be insert: ");
              scanf("%d",&value);
              enQueue(value);
              break;
      case 2: deQueue();
              break;
      case 3: display();
              break;
      case 4: exit(0);
      default: printf("\nWrong selection!!! Try again!!!");
    }
  }
}
void enQueue(int value){
  if(rear == SIZE-1)
```



```
    printf("\nQueue is Full!!! Insertion is not possible!!!");
  else{
    if(front == -1)
      front = 0;
    rear++;
    queue[rear] = value;
    printf("\nInsertion success!!!");
  }
}
void deQueue(){
  if(front == rear)
    printf("\nQueue is Empty!!! Deletion is not possible!!!");
  else{
    printf("\nDeleted : %d", queue[front]);
    front++;
    if(front == rear)
      front = rear = -1;
  }
}
void display(){
  if(rear == -1)
    printf("\nQueue is Empty!!!");
  else{
    int i;
    printf("\nQueue elements are:\n");
    for(i=front; i<=rear; i++)
      printf("%d\t",queue[i]);
  }
}
```

Test data and Output

```
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
```

```
Enter your choice: 1
Enter the value to be insert:10
Insertion success!!!

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20
Insertion success!!!
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3
Queue elements are:
10   20
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2
Deleted : 10
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3
Queue elements are:
20
```


LAB PROGRAM - 5

Q) Write a C program to implement Queue operations using Linked List.

Aim: To implement Queue operations using Linked List.

Program:

```
#include<stdio.h>
#include<conio.h>
struct Node
{
  int data;
  struct Node *next;
}*front = NULL,*rear = NULL;
void insert(int);
void delete();
void display();
void main()
{
  int choice, value;
  clrscr();
  printf("\n:: Queue Implementation using Linked List ::\n");
  while(1){
    printf("\n****** MENU ******\n");
    printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice){
      case 1: printf("Enter the value to be insert: ");
              scanf("%d", &value);
              insert(value);
              break;
      case 2: delete(); break;
      case 3: display(); break;
      case 4: exit(0);
      default: printf("\nWrong selection!!! Please try again!!!\n");
    }
}
}
```

```
void insert(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  newNode -> next = NULL;
  if(front == NULL)
    front = rear = newNode;
  else{
    rear -> next = newNode;
    rear = newNode;
  }
  printf("\nInsertion is Success!!!\n");
}
void delete()
{
  if(front == NULL)
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;
    front = front -> next;
    printf("\nDeleted element: %d\n", temp->data);
    free(temp);
  }
}
void display()
{
  if(front == NULL)
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;
    while(temp->next != NULL){
      printf("%d--->",temp->data);
      temp = temp -> next;
    }
    printf("%d--->NULL\n",temp->data);
  } }
```


Test data and Output

```
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert:10
Insertion success!!!
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20
Insertion success!!!
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3
Queue elements are:
10 ---> 20 ---> NULL
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2
Deleted element:  10
```

LAB PROGRAM - 6

Q) Write a C program to implement binary tree traversal.

Aim: To implement binary tree traversal.

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
        int data;
        struct node* left;
        struct node* right;
};
struct node* newNode(int data)
{
        struct node* node
                = (struct node*)malloc(sizeof(struct node));
        node->data = data;
        node->left = NULL;
        node->right = NULL;

        return (node);
}
void printPostorder(struct node* node)
{
        if (node — NULL)
                return;
        printPostorder(node->left);
        printPostorder(node->right);
        printf("%d ", node->data);
}

void printInorder(struct node* node)
{
        if (node == NULL)
                return;

        printInorder(node->left);
        printf("%d ", node->data);
```



```
        printInorder(node->right);

}
void printPreorder(struct node* node)
{
        if (node == NULL)
                return;
        printf("%d ", node->data);
        printPreorder(node->right);

}
int main()
{
        struct node* root = newNode(1);
        root->left = newNode(2);
        root->right = newNode(3);
        root->left->left = newNode(4);
        root->left->right = newNode(5);
        printf("\nPreorder traversal of binary tree is \n");
        printPreorder(root);
        printf("\nInorder traversal of binary tree is \n");
        printInorder(root);
        printf("\nPostorder traversal of binary tree is \n");
        printPostorder(root);
        getchar();
        return 0;
}
```

Test data and Output

```
Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Postorder traversal of binary tree is
4 5 2 3 1
```

LAB PROGRAM - 7

Q) Write a C program to implement bubble sort.

Aim: To implement bubble sort.

Program:

```
#include <stdio.h>
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
void bubbleSort(int arr[], int n)
{
int i, j;
for (i = 0; i < n-1; i++)

    for (j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1])
            swap(&arr[j], &arr[j+1]);
}
void printArray(int arr[], int size)
{   int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main()
{   int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Test data and Output

Sorted array:
11 12 22 25 34 64 90

LAB PROGRAM - 8

Q) Write a C program to implement insertion sort.

Aim: To implement insertion sort.

Program:

```
#include <stdio.h>
void printArray(int array[], int size) {
 for (int i = 0; i < size; i++) {
  printf("%d ", array[i]);
 }
 printf("\n");
}
void insertionSort(int array[], int size) {
 for (int step = 1, step < size; step++) {
  int key = array[step];
  int j = step - 1;
  while (key < array[j] && j >= 0) {
   array[j + 1] = array[j];
   --j;
  }
  array[j + 1] = key;
 }
}
int main()
{
 int data[] = {9, 5, 1, 4, 3};
 int size = sizeof(data) / sizeof(data[0]);
 insertionSort(data, size);
 printf("Sorted array in ascending order:\n");
 printArray(data, size);
}
```

Test data and Output

Sorted array in ascending order:
1 3 4 5 9

LAB PROGRAM - 9

Q) Write a C program to implement Linear search.

Aim: To implement Linear search.

Program:

```
#include <stdio.h>
int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int n = sizeof(arr)
    int result = search(arr, n, x);
    if(result == -1)
        printf("Element is not present in array");
    else
        printf("Element is present at index %d", result);
    return 0;
}
```

Test data and Output

Element is present at index 3

LAB PROGRAM - 10

Q) Write a C program to implement Binary search.

Aim: To implement Binary search.

Program:

```
#include <stdio.h>
void binary_search();
int a[50], n, item, loc, beg, mid, end, i;
main()
{
  printf("\nEnter size of an array: ");
  scanf("%d", &n);
  printf("\nEnter elements of an array in sorted form:\n");
  for(i=0; i<n; i++)
    scanf("%d", &a[i]);
  printf("\nEnter ITEM to be searched: ");
  scanf("%d", &item);
  binary_search();
  getch();
}
void binary_search()  //called Function - Dont Return Anything.
{
  beg = 0;
  end = n-1;
  mid = (beg + end) / 2;
  while ((beg<=end) && (a[mid]!=item))
  {
    if (item < a[mid])
      end = mid - 1;
    else
      beg = mid + 1;
    mid = (beg + end) / 2;
  }
  if (a[mid] == item)
    printf("\n\nITEM found at location %d", mid+1);
  else
    printf("\n\nITEM doesn't exist");
}
```

Test data and Output

Enter size of an array: 6
Enter elements of an array in sorted form:
10 20 30 40 50 60
Enter ITEM to be searched: 30
ITEM found at location 3

Question Bank

UNIT - I

1. Write about Abstract Data Types.
2. Explain Primitive Data Types.
3. Explain Software Engineering.
4. Write about Tips and Techniques for Writing Programs in 'C'.

UNIT - II

5. Define an Array? Explain.
6. Write about Pointers and Arrays.
7. Explain Dynamic Memory Allocation.
8. Explain Linked List Operations.

UNIT - III

9. Write about Stack operations.
10. Write about Representation of Stacks through Linked Lists.
11. Define a Queue? Explain.
12. Explain Circular Queue.

UNIT -IV

13. Write about Basic Definition of Binary Trees.
14. Explain Representation of Binary Trees.
15. Explain Binary Tree Traversal.

UNIT - V

16. Explain Bubble Sort.
17. Write about Merge Sort.
18. Write about Binary Search.
19. Explain Terms Associated with Graphs.
20. Explain Spanning Trees.