LNCTE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Data Structure CS-303

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY & EXCELLENCE, BHOPAL

Department of Computer Science and Engineering

Lab Manual

Subject Name: Data Structure

Course Code: CS-303

Course: B.Tech

Session:2023-24

Prepared By

TABLE OF CONTENT

Vision and Mission of the Institute

VISION OF THE INSTITUTE

To become a pioneer institute in technical education and innovations to build competent technocrats and leaders for the nation.

MISSION OF THE INSTITUTE

M1. To enhance the academic environment with innovative teaching learning processes and modern tools.

M2. To Practice and nurture high standards of human values, transparency and accountability.

M3. To collaborate with other academic and research institutes as well as industries in order to strengthen education and research.

M4. To uphold skill development for employability and entrepreneurship for interdisciplinary research and innovations.

Vision and Mission of the Department

VISION OF THE DEPARTMENT

To be a centre of excellence for providing quality technical education to develop future leaders with the aspects of research &computing, Software product development and entrepreneurship.

MISSION OF THE DEPARTMENT

M1: To offer academic programme with state of art curriculum having flexibility for accommodating the latest developments in the areas of Computer science engineering

M2: To conduct research and development activities in contemporary and emerging areas of computer science & engineering.

M3: To inculcate moral values & entrepreneurial skills to produce professionals capable of providing socially relevant and sustainable solutions.

FOREWORD

It is my great pleasure to present this laboratory manual for second year engineering students for the subject of Data Structure to understand the paradigms and approaches used to analyze and design algorithms and to appreciate the impact of algorithm design in practice.

It also ensures that students understand how the worst-case time complexity of an algorithm is computed. This being a core subject, it becomes very essential to have clear theoretical and designing aspects.

This lab manual provides a platform to the students for understanding the basic concepts of designing and analyzing of algorithm. This practical background will help students to gain confidence in qualitative and quantitative approach to synthesize and analyze the algorithms.

H.O.D

LABORATORY MANUAL CONTENTS

This manual is intended for the Second Year students of CSE branch in the subject of Data Structure. This manual typically contains practical/ Lab Sessions related to Design and Analysis of Algorithm covering various aspects related to the subject for enhanced understanding.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Dos and Don'ts in Laboratory :-

1. Turn off the machine once you are done using it.
2. Student should not attempt to repair, open, tamper or interfere with any of the computer, printing, cabling or other equipment in the laboratory.
3. Make entry in the Log Book as soon as you enter in the laboratory.
4. All the students are supposed to enter the terminal number in the log book.
5. Handle equipments with care.
6. Strictly observe the instructions given by the Teacher/ Lab Instructor.
7. Do not change the terminal on which you are working.
8. Do not install/remove any software on system without permission.
9. Do not open any irrelevant internet sites on lab computer.
10. Do not remove anything from computer laboratory without permission.
11. Do not misbehave in the computer laboratory.
12. Do not plug in external devices without scanning them for computer viruses.

Instruction for Laboratory Teachers:-

1. Submission related to whatever lab work has been completed should be done during the next lab session.
2. Students should be instructed to start the computers. After the experiment is over, the students must shut down the Computers and turn off the switches.
3. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

COURSE OUTCOMES for CS-303 Data structure

Students should be able to

CO303.1	Explain operations arrays, Link list along with their merits and demerits
CO303.2	Use primitive operations, stack and queue data structures.
CO303.3	Develop programs to perform operations on Binary Tree,BST.
CO303.4	Utilize Dijkstra's algorithm to find spanning tree for a given graph.
CO303.5	Apply quick and merge sorting methods in problem solving.

Course Articulation Matrix

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO303.1	3	3	1	-	-	-	-	-	2	-	2	3
CO303.2	3	-	3	-	-	2	-	-	2	-	-	3
CO303.3	3	1	3	2	-	-	-	1	2	-	-	3
CO303.4	3	1	1	2	-	1	-	-	2	2	2	3
CO303.5	3	2	1	-	2	-	-	-	2	2	-	3
	3	**2**	**2**	**2**	**2**	**2**	**-**	**1**	**2**	**2**	**2**	**3**

Program Outcomes as defined by NBA (PO)

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSO)

A graduate of Computer Science and Engineering Program will develop

PSO1: An ability to apply technical knowledge of computer science and engineering fundamentals to become employable in industry.

PSO2: An ability to develop programming skills using modern software tools and techniques.

PSO3: An ability to develop real time projects for problem solving of domains such as Machine Learning, Cyber security, block chain and big data.

PSO4: An ability to grab research, higher studies and entrepreneurship opportunities towards society with moral values and ethics.

Program Educational Objectives (PEO):

PEO 1: Evolve as globally competent computer professionals, researchers and entrepreneurs possessing collaborative and leadership skills, for developing innovative solutions in multidisciplinary domains.

PEO 2: Excel as socially committed computer engineers having mutual respect, effective communication skills, high ethical values and empathy for the needs of society.

PEO 3: Involve in lifelong learning to foster the sustainable development in the emerging areas of technology

UNIVERSITY SCHEME

Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal

New Scheme of Examination as per AICTE Flexible Curricula

III Semester **Bachelor of Technology (B.Tech.) [Computer Science and Engineering/ Computer Engineering/Computer Science & Technology]**

For batches admitted in July, 2020 (w.e.f. July, 2021)

S.No.	Subject Code	Category	Subject Name	Maximum Marks Allotted					Total Marks	Contact Hours per week			Total Credits
				Theory			Practical						
				End Sem.	Mid Sem. Exam.	Quiz/ Assignment	End Sem	Term work / Lab Work & Sessional		L	T	P	
1.	ES301	HSMC-3	Energy & Environmental Engineering	70	20	10	-	-	100	3	1	-	4
2.	CS302	DC-1	Discrete Structure	70	20	10	-	-	100	3	1	-	4
3.	CS303	DC-2	Data Structure	70	20	10	30	20	150	3	-	2	4
4.	CS304	DC-3	Digital Systems	70	20	10	30	20	150	3	-	2	4
5.	CS305	DC-4	Object Oriented Programming & Methodology	70	20	10	30	20	150	3	-	2	4
6.	CS306	DLC-3	Computer Workshop	-	-	-	30	20	50	-	-	4	2
7.	BT107	DLC-1	Evaluation of Internship-I completed at I year level	-	-	-	-	50	50			4	2
8.	BT307	DLC-4	90 hrs Internship based on using various softwares –Internship -II	To be completed anytime during Third/ fourth semester. Its evaluation/credit to be added in fifth semester.									
			Total	350	100	50	120	130	750	15	2	14	24
9.	BT308	MC	Indian Constitution	Non-credit course									
	NC301		NSS/NCC										

*Students can earn additional credits from the University recognized MOOC courses.

1 Hr Lecture	1 Hr Tutorial	2 Hr Practical
1 Credit	1 Credit	1 Credit

SYLLABUS

Branch: CSE, III Semester **Course:** BTech

RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

New Scheme Based On AICTE Flexible Curricula

Computer Science and Engineering, III-Semester

CS303 Data Structure

1. Review of C programming language.Introduction to Data Structure: Concepts of Data and Information, Classification of Data structures, Abstract Data Types, Implementation aspects: Memory representation. Data structures operations and its cost estimation. Introduction to linear data structures- Arrays, Linked List: Representation of linked list in memory, different implementation of linked list. Circular linked list, doubly linked list, etc. Application of linked list: polynomial manipulation using linked list, etc.
2. Stacks: Stacks as ADT, Different implementation of stack, multiple stacks. Application of Stack: Conversion of infix to postfix notation using stack, evaluation of postfix expression, Recursion. Queues: Queues as ADT, Different implementation of queue, Circular queue, Concept of Dqueue and Priority Queue, Queue simulation, Application of queues.
3. Tree: Definitions - Height, depth, order, degree etc. Binary Search Tree - Operations, Traversal, Search. AVL Tree, Heap, Applications and comparison of various types of tree; Introduction to forest, multi-way Tree, B tree, B+ tree, B* tree and red-black tree.
4. Graphs: Introduction, Classification of graph: Directed and Undirected graphs, etc, Representation, Graph Traversal: Depth First Search (DFS), Breadth First Search (BFS), Graph algorithm: Minimum Spanning Tree (MST)- Kruskal, Prim's algorithms. Dijkstra's shortest path algorithm; Comparison between different graph algorithms. Application of graphs.
5. Sorting: Introduction, Sort methods like: Bubble Sort, Quick sort. Selection sort, Heap sort, Insertion sort, Shell sort, Merge sort and Radix sort; comparison of various sorting techniques. Searching: Basic Search Techniques: Sequential search, Binary search, Comparison of search methods. Hashing & Indexing. Case Study: Application of various data structures in operating system, DBMS etc.

Text Books

1. AM Tanenbaum, Y Langsam& MJ Augustein, "Data structure using C and C++", Prentice Hall India.
2. Robert Kruse, Bruse Leung, "Data structures & Program Design in C", Pearson Education.

Reference Books

1. Aho, Hopcroft, Ullman, "Data Structures and Algorithms", Pearson Education.
2. N. Wirth, "Algorithms + Data Structure = Programs", Prentice Hall.
3. Jean – Paul Trembly , Paul Sorenson, "An Introduction to Structure with application", TMH.
4. Richard, GilbergBehrouz, Forouzan ,"Data structure – A Pseudocode Approach with C", Thomson press.

LIST OF EXPERIMENTS

S.No.	EXPERIMENTS	
1.	Write a program for create an array.	CO1
2.	Write a program to implement the multiplication of [3*3] matrix using Array.	CO1
3.	Write a program to implement Link List.	CO1
4.	Write a program that implement stack .	CO2
5.	Write a program that implement Queue .	CO2
6.	Write a program to implement traverse a Tree.	CO3
7.	Write a program to Traverse a Graph using BFS or DFS.	CO4
8.	Write a program that implements Bubble sort .	CO5
9.	Write a program to implements Binary Search	CO5
10.	Write a program to implements the Merge sort.	CO5

Arrays are defined as the collection of similar types of data items stored at contiguous memory locations. It is one of the simplest data structures where each data element can be randomly accessed by using its index number.

In C programming, they are the derived data types that can store the primitive type of data such as int, char, double, float, etc. For example, if we want to store the marks of a student in 6 subjects, then we don't need to define a different variable for the marks in different subjects. Instead, we can define an array that can store the marks in each subject at the contiguous memory locations.

- Each element in an array is of the same data type and carries the same size that is 4 bytes.
- Elements in the array are stored at contiguous memory locations from which the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Representation of an array

- Index starts with 0.
- The array's length is 10, which means we can store 10 elements.
- Each element in the array can be accessed via its index.

EXPERIMENT-1

Aim: Write a program for create an array and insert an element into specific position.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a[50],i,n,pos,item=0;
 clrscr();
 printf("\n Enter the Array nSIze");
 scanf("%d", &n);
 printf("\n Enter the array Element");
 for(i=0;i<n;i++)
 {
  scanf("%d",&a[i]);
 }
 printf("\n Dispaly of Array Element");
 for(i=0;i<n;i++)
 {
  printf("\n a[%d]=%d",i,a[i]);
 }
 printf("\n Insert Element ");
 printf("\n======================");
 printf("\n Enter Position ");
 scanf("%d",&pos);
  n++;
 for(i=n;i>=pos;i--)
 {    a[i]=a[i-1];
 }
 printf("\n Insert Item");
 scanf("%d",&item);
 a[--pos]=item;
 printf("\n Dispay arrary after inter new Element");
 for(i=0;i<n;i++)
 {
  printf("\n a[%d]=%d", i,a[i]);
 }
 getch();

 }
```

Output:--

```
 Enter the Array Size=3

 Enter the array Element
1
2
3

 Dispaly of Array Element

 a[0]=1
 a[1]=2
 a[2]=3
 Insert Element
======================
 Enter Position 1

 Insert Item=100

 Dispay arrary after inter new Element
 a[0]=100
 a[1]=1
 a[2]=2
 a[3]=3_
```

Introduction to Matrix Data structure

A matrix is a two-dimensional array that consists of **rows** and **columns**. It is an arrangement of elements in horizontal or vertical lines of entries.

Different types of Matrix :

Here are some common types of matrices:

- **Square matrix:** A square matrix is a matrix in which the number of rows and columns are equal. In other words, it has the same number of rows as columns. For example, a 3×3 matrix is a square matrix.
- **Identity matrix**: An identity matrix is a square matrix in which all the diagonal elements are 1 and all other elements are 0.
- **Diagonal matrix:** A diagonal matrix is a square matrix in which all the non-diagonal elements are 0.
- **Upper triangular matrix:** An upper triangular matrix is a square matrix in which all the elements below the main diagonal (the diagonal from the top left corner to the bottom right corner) are 0.
- **Lower triangular matrix:** A lower triangular matrix is a square matrix in which all the elements above the main diagonal are 0.
- **Sparse matrix:** A sparse matrix is a matrix in which the majority of the elements are 0. Sparse matrices are often represented using compressed sparse row (CSR) or compressed sparse column (CSC) formats to save space and optimize performance in algorithms that work with such matrices.

Applications of Matrix:

- **Graph theory:** Matrices can be used to represent graphs and networks. For example, the adjacency matrix of a graph is a matrix that describes the connections between nodes in the graph.
- **Image processing:** Matrices are commonly used in image processing to represent images as matrices of pixel values.
- **Optimization**: Matrices are used in optimization problems, such as linear programming and quadratic programming, to represent the objective function and constraints of the problem.
- **Machine learning:** Matrices are used to represent training data and model parameters, and matrix operations are used to perform model

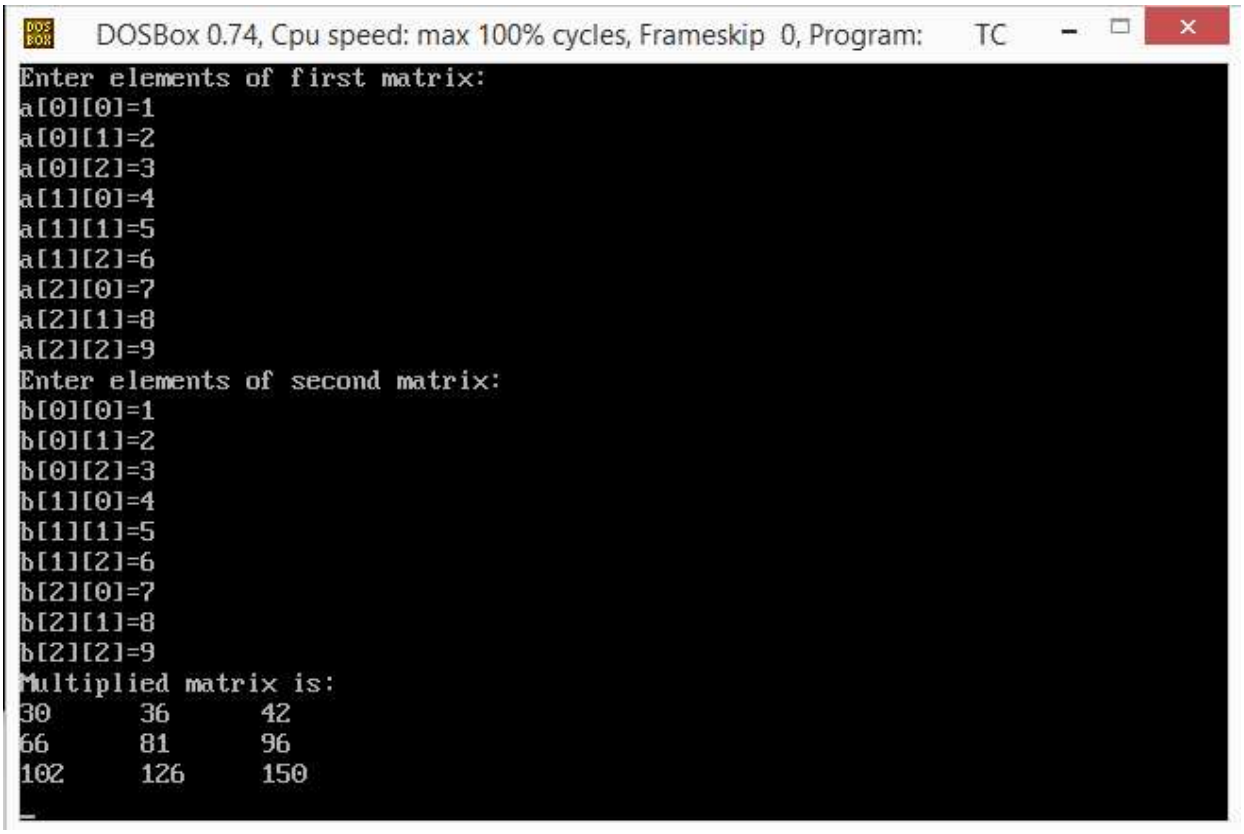
EXPERIMENT-2

Aim: **Write a program in to implement the multiplication of [3*3] matrix using Array.**

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,j,k;
 int a[3][3], b[3][3], mul[3][3];

 printf("Enter elements of first matrix:\n");
 for(i=0;i< 3;i++)
 {
  for(j=0;j< 3;j++)
  {
   printf("a[%d][%d]=",i,j);
   scanf("%d", &a[i][j]);
  }
 }

 printf("Enter elements of second matrix:\n");
 for(i=0;i< 3;i++)
 {
  for(j=0;j< 3;j++)
  {
   printf("b[%d][%d]=",i,j);
   scanf("%d", &b[i][j]);
  }
 }
 for(i=0;i< 3;i++)
 {
  for(j=0;j< 3;j++)
  {
   mul[i][j] = 0;
   for(k=0;k< 3;k++)
   {
    mul[i][j] = mul[i][j] + a[i][k]*b[k][j];
   }
  }
 }
 printf("Multiplied matrix is:\n");
 for(i=0;i< 3;i++)
 {
  for(j=0;j< 3;j++)
  {
   printf("%d\t", mul[i][j]);
  }
  printf("\n");
```

```
 }
 getch();
}
```

Output:-



Link LIST

Linked List is a very commonly used linear data structure which consists of group of **nodes** in a sequence.

Each node holds its own **data** and the **address of the next node** hence forming a chain like structure.

Advantages of Linked Lists

- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.

Disadvantages of Linked Lists

- The memory is wasted as pointers require extra memory for storage.
- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in linked list.

EXPERIMENT-3

Aim: **Write a program to implement Link List.**

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
struct node
{
 int data;
 struct node *next;
};
typedef struct node NODE;
void main()
{
 NODE *startnode,*newnode,*temp;
 int i, n;
 clrscr();
 printf("\n Enter Size of LINK IIST=");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  newnode=(NODE *)malloc(sizeof(NODE));
  printf("\n Enter the LList element %d=", (i+1));
  scanf("%d",&newnode->data);
  if(i==0)
  {
   startnode=temp=newnode;
  }
  else
  {
   temp->next=newnode;
   temp=newnode;
  }

 }
temp->next=NULL;
temp=startnode;
printf("\nDisplay of Link List \n ");
printf("\n=========================\n");
while(temp!=NULL)
{
 printf("|%d|%u|-->" ,temp->data,temp->next);
 temp=temp->next;
}
getch();
}
```

Output:

```
 Enter Size of LINK IIST=4

 Enter the LList element 1=1

 Enter the LList element 2=2

 Enter the LList element 3=3

 Enter the LList element 4=4

Display of Link List

========================
¦1¦1960¦-->¦2¦1968¦-->¦3¦1976¦-->¦4¦0¦-->_
```

What is Stack?

A stack is a linear data structure in which the insertion of a new element and removal of an existing element takes place at the same end represented as the top of the stack. To implement the stack, it is required to maintain the **pointer to the top of the stack**, which is the last element to be inserted because **we can access the elements only on the top of the stack.**

LIFO(Last In First Out):

This strategy states that the element that is inserted last will come out first. You can take a pile of plates kept on top of each other as a real-life example. The plate which we put last is on the top and since we remove the plate that is at the top, we can say that the plate that was put last comes out first.

Basic Operations on Stack

In order to make manipulations in a stack, there are certain operations provided to us.

- **push()** to insert an element into the stack
- **pop()** to remove an element from the stack
- **top()** Returns the top element of the stack.
- **isEmpty()** returns true if stack is empty else false.
- **size()** returns the size of stack.

EXPERIMENT-4

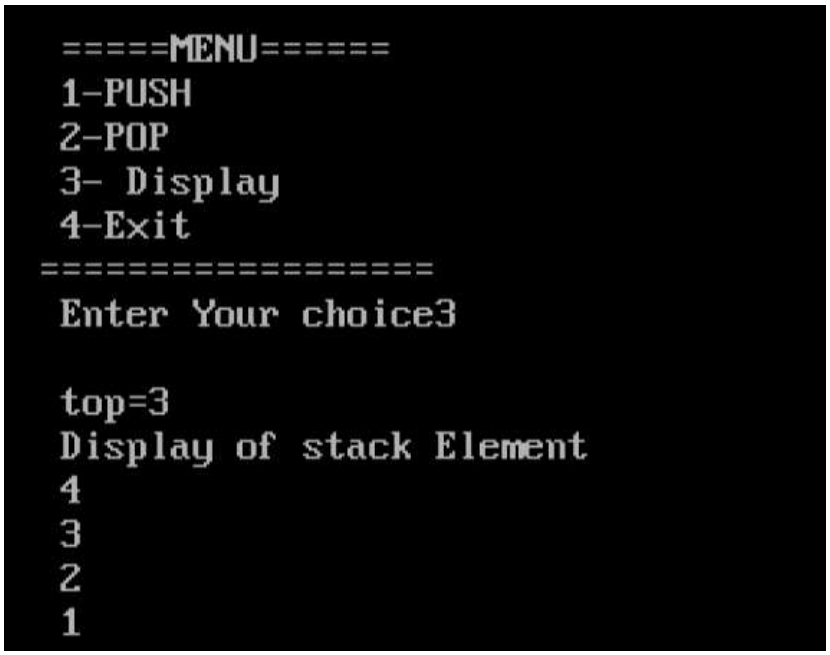
Aim: **Write a program that implement stack using.**

i) Arrays

```
/* Write a programe for STACk using Array */
#include<stdio.h>
#include<conio.h>
#define N 5
void push();
int pop();
void display();
int stack[N];
int i,n;
int top=-1;
void main()
{
 int choice;
 do
 {
  clrscr();
  printf("\n =====MENU======");
  printf("\n 1-PUSH");
  printf("\n 2-POP");
  printf("\n 3- Display");
  printf("\n 4-Exit");
  printf("\n==================");
  printf("\n Enter Your choice");
  scanf("%d",&choice);
  switch(choice)
  {
   case 1: push();
            break;
   case 2:printf("\n Deleted element is %d",pop());
            display();
            getch();
            break;
   case 3:display();
            getch();
            break;
   case 4: exit(0);

  }
 }while(choice!=4);
}
//Insert Element from stack Or PUSH operation
void push()
```

```
{
 int item;
 if(top+1==N)
 {
  printf("\n stack is Full");
 }
 else
 {
  printf("\n Enter the element  ");
  scanf("%d",&item);
  top=top+1;
  stack[top]=item;
 }
 }
 //Delete th element from stack
 int pop()
 {
  int item;
  if (top<0)
  {
   printf("\n stack is empty");
  }
  else
  {
   item=stack[top];
   top=top-1;
  }
  return(item);
 }
//Display
 void display()
{
printf("\n top=%d",top);
 if (top>0)
 {
  printf("\n Display of stack Element");
  for(i=top;i>=0;i--)
  {
   printf("\n %d",stack[i]);
  }
 }
 else
 {
  printf("\n stack is empty");
 }
}
```

Output:-


=====MENU======
1-PUSH
2-POP
3- Display
4-Exit
==================
Enter Your choice3
top=3
Display of stack Element
4
3
2
1

Queue *is a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order.*
We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. The element which is first pushed into the order, the delete operation is first performed on that.

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

Characteristics of Queue:

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

Queue Representation:

1. Array Representation of Queue:

Like stacks, Queues can also be represented in an array: In this representation, the Queue is implemented using the array. Variables used in this case are

- **Queue:** the name of the array storing queue elements.
- **Front**: the index where the first element is stored in the array representing the queue.
- **Rear:** the index where the last element is stored in an array representing the queue.

EXPERIMENT-5

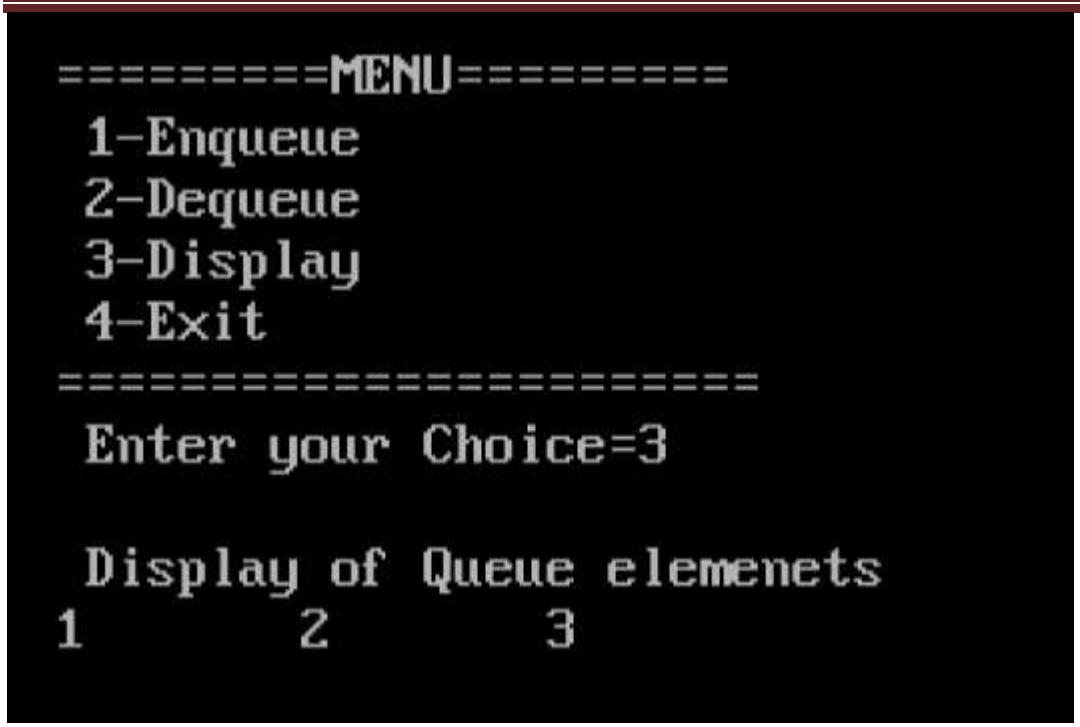
Aim: **Write a program that implement Queue using**
i) Arrays

```
//Queue implementation using Array
#include<stdio.h>
#include<conio.h>
#define Max 5
int Q[Max];
void Enqueue();
void Dequeue();
void Display();
int front=-1;
int rear=-1;
void main()
{
 int choice;
 do
 {
  clrscr();
  printf("\n=========MENU=========");
  printf("\n 1-Enqueue");
  printf("\n 2-Dequeue");
  printf("\n 3-Display");
  printf("\n 4-Exit");
  printf("\n======================");
  printf("\n Enter your Choice=");
  scanf("%d",&choice);
  switch(choice)
  {
   case 1: Enqueue();
           getch();
           break;
   case 2: Dequeue();
           Display();
           getch();
           break;
   case 3: Display();
           getch();
           break;
   case 4: exit(0);


  }


}while(choice !=4);
```

```
}
void Enqueue()
{
 int iteam;
 printf("\n Enter element of Queue=");
 scanf("%d",&iteam);
 printf("\n Front=%d",front);
 printf("\n Rear=%d",rear);


 if(rear+1==Max)
 {
  printf("\n Queue is Overflow");
 }
 else
 {
  if(front==-1 && rear==-1)
  {
   front=front+1;
   rear=rear+1;
   Q[rear]=iteam;
  }
  else
  {
   rear=rear+1;
   Q[rear]=iteam;
  }
 }
}


//Dequeue Function
void Dequeue()
{
 int iteam;
 if(front==-1 && rear==-1)
 {
  printf("\n Queue is Underflow");
 }
 else
 {
    if(front==rear)
    {
         iteam=Q[front];
         front=rear=-1;
    }
    else
    {
```

```
            iteam=Q[front];
            front=front+1;
       }
      printf("\n Deleted iteam is =%d",iteam);
  }
}



void Display()
{
 int i;
 if(rear>=0)
 {
   printf("\n Display of Queue elemenets \n");
   for(i=front; i<=rear; i++)
   {
    printf("%d\t",Q[i]);
   }
 }
 else
 {
    printf("Queue is empty");
 }
}
```

Output:-



*A **tree data structure** is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.*
The topmost node of the tree is called the root, and the nodes below it are called the child nodes. Each node can have multiple child nodes, and these child nodes can also have their own child nodes, forming a recursive structure

.

Basic Operation Of Tree Data Structure:

- **Create** – create a tree in the data structure.
- **Insert** − Inserts data in a tree.
- **Search** − Searches specific data in a tree to check whether it is present or not.
- **Traversal**:
  - **Preorder Traversal** – perform Traveling a tree in a pre-order manner in the data structure.
  - **In order Traversal** – perform Traveling a tree in an in-order manner.
  - **Post-order Traversal** –perform Traveling a tree in a post-order manner.

EXPERIMENT-6

Aim: Write a program to implement traverse a Tree.

```
#include<stdio.h>
#include<conio.h>
struct node
{
 int data;
 struct node *left;
 struct node *right;
};
typedef struct node NODE;

void insert(NODE *,NODE *);
void preorder(NODE *tree);
void inorder(NODE *tree);
void postorder(NODE *tree);

NODE *newnode, *root;
NODE *getnode();
void main()
{
int choice;
char ans='n';
root=NULL;
clrscr();
do
{
printf("\n Programe for Implement Simple Binary Tree");
printf("\n 1. Create");
printf("\n 3. Inorder");
printf("\n 2. Preorder");
printf("\n 4. PostOrder");
printf("\n 5. Exit");
printf("\n Enter Your choice");
scanf("%d",&choice);
 switch(choice)
 {
   case 1:root=NULL;
   do
   {

    newnode=getnode();

   printf("\n Enter The Element=");
   scanf("%d",&newnode->data);
```

```
  if(root==NULL)
  {
  root=newnode;
  }
  else
  {
  insert(root,newnode);
  }
  printf("\n Do you want to Enter more element?(y/n)");
  ans=getche();

 }while(ans=='y'||ans=='Y');
  clrscr();
  break;

  case 2: puts("\n Preorder traversing Tree");

  preorder(root);
  break;
  case 3:
        if(root==NULL)
        {
        printf("Tree is Not Created");
        }
        else
        {
         printf("\n Inorder traversing Tree \n");
          inorder(root);
          //break;
        }
        break;
  case 4: puts("\n postorder Traversing Tree \n");
          postorder(root);
          break;
  case 5: puts("END");
          exit (0);
 }
}while(choice!=5);

}
void inorder(NODE *p)
{
 if(p!=NULL)
 {
  inorder(p->left);
  printf("%d\t",p->data);
  inorder(p->right);
 }
}
```

```
//Preorder--------------
void preorder(NODE *p)
{
 if(p!=NULL)
 {
  //inordr(p->left);
  printf("%d\t",p->data);
  preorder(p->left);
  preorder(p->right);
 }
}
//Postorder--------------
void postorder(NODE *p)
{
 if(p!=NULL)
 {
   postorder(p->left);
   postorder(p->right);
   printf("%d\t",p->data);

 }
}

void insert(NODE *root, NODE *newnode)
{
 char ch;
 printf("\n Where to insert left(l)/right(r) %d", root->data);
 ch=getche();
 if((ch=='r')||(ch=='R'))
 {
  if (root->right==NULL)
  {
          root->right=newnode;
  }
  else
  {
   insert(root->right,newnode);
  }
 }
 else
 {
  if(root->left==NULL)
  {
   root->left=newnode;
  }
  else
  {
   insert(root->left,newnode);
  }
```

```
 }
}
NODE *getnode()
{
  NODE *temp;
  temp=(NODE *) malloc(sizeof(NODE));
   temp->left=NULL;
   temp->right=NULL;

  return temp;
}
```

Output:-

```
Programe for Implement Simple Binary Tree
1. Create
3. Inorder
2. Preorder
4. PostOrder
5. Exit
Enter Your choice1

Enter The Element=1

Do you want to Enter more element?(y/n)y
Enter The Element=2

Where to insert left(l)/right(r) 1l
Do you want to Enter more element?(y/n)y
Enter The Element=3

Where to insert left(l)/right(r) 1r
Do you want to Enter more element?(y/n)_
```

```
 Programe for Implement Simple Binary Tree
 1. Create
 3. Inorder
 2. Preorder
 4. PostOrder
 5. Exit
 Enter Your choice3

 Inorder traversing Tree
2       1       3
 Programe for Implement Simple Binary Tree
 1. Create
 3. Inorder
 2. Preorder
 4. PostOrder
 5. Exit
 Enter Your choice_
```

A graph is a non-linear data structure composed of vertices and edges. Edges are lines or arcs that connect any two nodes in the network. Vertices are also known as nodes. A graph, in more technical terms, is made up of vertices (V) and edges (E). The representation of a graph is G(E, V). So, in this article, we will look at some Graph Traversal Techniques.

Graph Traversal in Data Structure

We can traverse a graph in two ways :

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

BFS Graph Traversal in Data Structure

Breadth-first search (BFS) traversal is a technique for visiting all nodes in a given network. This traversal algorithm selects a node and visits all nearby nodes in order. After checking all nearby vertices, examine another set of vertices, then recheck adjacent vertices. This algorithm uses a queue as a data structure as an additional data structure to store nodes for further processing. Queue size is the maximum total number of vertices in the graph.

DFS Graph Traversal in Data Structure

When traversing a graph, the DFS method goes as far as it can before turning around. This algorithm explores the graph in depth-first order, starting with a given source node and then recursively visiting all of its surrounding vertices before backtracking. DFS will analyze the deepest vertices in a branch of the graph before moving on to other branches. To implement DFS, either recursion or an explicit stack might be utilized.

EXPERIMENT-7

Aim: Write a program to Traverse a Graph using BFS.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size 20
#define TRUE 1
#define FALSE 0
int g[size][size];
int visit[size];
int Q[size];
int front , rear;
int n;
void main()
{
 int v1,v2;
 char ans='y';
 void create(),bfs();
 clrscr();
 create();
 clrscr();
 printf("\n The Adjacency Matrix for the graph \n");
 for(v1=0;v1<n; v1++)
 {
  for(v2=0;v2<n;v2++)
  {
   printf("%d",g[v1][v2]);
  }
   printf("\n");

 }
 getch();
 do
 {
  for(v1=0;v1<n;v1++)
  {
   visit[v1]=FALSE;
   }
   clrscr();
   printf("\n Enter the Vertex from which you want to traverse");
   scanf("%d",&v1);
   if(v1>=n)
   {
    printf("\n Invalid Vertex");
   }
   else
   {
```

```
    printf("\n The BFS of The Graph is \n");
    bfs(v1);
    getch();
   }
   printf("\n Do you want to traverse any other vertix ?(y/n)");
   ans=getche();
  }while(ans=='y');
 exit(0);
}
void create()
{
 int v1, v2;
 char ans='y';
 printf("\n This is a Program to Create a Graph");
 printf("\n The display is in BFS manner");
 printf("\n Enter no. of nodes=");
 scanf("%d",&n);
 for(v1=0;v1<n;v1++)
 {
  for(v2=0;v2<n;v2++)
  {
   g[v1][v2]=FALSE;
  }
 }
  printf("\n Enter the vertex no. starting from 0");
  do
  {
   printf("\n Enter the vertex v1 & v2 \n");
   scanf("%d%d",&v1,&v2);
   if(v1>=n || v2>=n)
   {
    printf("Invalid Verex Value \n");
   }
   else
   {
    g[v1][v2]=TRUE;
    g[v2][v1]=TRUE;
   }
   printf("\n \n Add more edges ??(y/n)");
   ans=getche();
  } while(ans=='y');
}
```

```
void bfs(int v1)
{
int v2;
visit[v1]=TRUE;
front=rear=-1;
Q[++rear]=v1;
while(front!=rear)
{
v1=Q[++front];
printf("%d\n",v1);
for(v2=0;v2<n;v2++)
{
 if(g[v1][v2]==TRUE && visit[v2]==FALSE)
 {
  Q[++rear]=v2;
  visit[v2]=TRUE;
  }
 }
}
```

Output:-

```
   This is a Program to Create a Graph
   The display is in BFS manner
   Enter no. of nodes=4

   Enter the vertex no. starting from 0
   Enter the vertex v1 & v2
  0 1


   Add more edges ??(y/n)y
   Enter the vertex v1 & v2
  0 2


   Add more edges ??(y/n)y
   Enter the vertex v1 & v2
  1 3


   Add more edges ??(y/n)y
   Enter the vertex v1 & v2
  2 3_
}
```

```
Enter the Vertex from which you want to traverse0

The BFS of The Graph is
0
1
2
3
```

Sorting refers to rearrangement of a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure.

Why Sorting Algorithms are Important

The sorting algorithm is important in Computer Science because it reduces the complexity of a problem. There is a wide range of applications for these algorithms, including searching algorithms, database algorithms, divide and conquer methods, and data structure algorithms

S.No.	Searching Algorithm	Sorting Algorithm
1.	Searching Algorithms are designed to retrieve an element from any data structure where it is used.	A Sorting Algorithm is used to arranging the data of list or array into some specific order.
2.	These algorithms are generally classified into two categories i.e. Sequential Search and Interval Search.	There are two different categories in sorting. These are Internal and External Sorting.
3.	The worst-case time complexity of searching algorithm is O(N).	The worst-case time complexity of many sorting algorithms like Bubble Sort, Insertion Sort, Selection Sort, and Quick Sort is $O(N^2)$.
4.	There is no stable and unstable searching algorithms.	Bubble Sort, Insertion Sort, Merge Sort etc are the stable sorting algorithms whereas Quick Sort, Heap Sort etc are the unstable sorting algorithms.
5.	The Linear Search and the Binary Search are the examples of Searching Algorithms.	The Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort etc are the examples of Sorting Algorithms.

EXPERIMENT-8

Aim: Write a program that implements Bubble sort..

i) Bubble sort

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a[20],i,j,n,temp;
 // clrscr();
 printf("\n Enter insert element");
 scanf("%d",&n) ;
//----------------Arry----------------------
 printf("\n Enter Array element");
 for(i=0;i<n;i++)
 {
    scanf("%d",&a[i]);
 }
 printf("\n Dispaly Array Element");
 for(i=0;i<n; i++)
 {
    printf("\n %d",a[i]);
 }
//---------------Bubble Short Logic------------------

 for(i=0;i<n-1; i++)
 {
     for(j=0;j<n-1; j++)
     {
        if(a[j]>a[j+1])
        {
          temp=a[j];
          a[j]=a[j+1];
          a[j+1]=temp;
        }
     }
 }
//-------------Dispaly Array after shorting

 printf("\n Dispaly Array after short");
 for(i=0;i<n; i++)
 {

    printf("\n %d",a[i]);
 }
 getch();
```

}

Output:-

```
 Enter No. of element
5

 Enter Array element
8
1
7
2
9

 Dispaly Array Element
 8
 1
 7
 2
 9
 Dispaly Array after short
 1
 2
 7
 8
 9_
```

EXPERIMENT-9

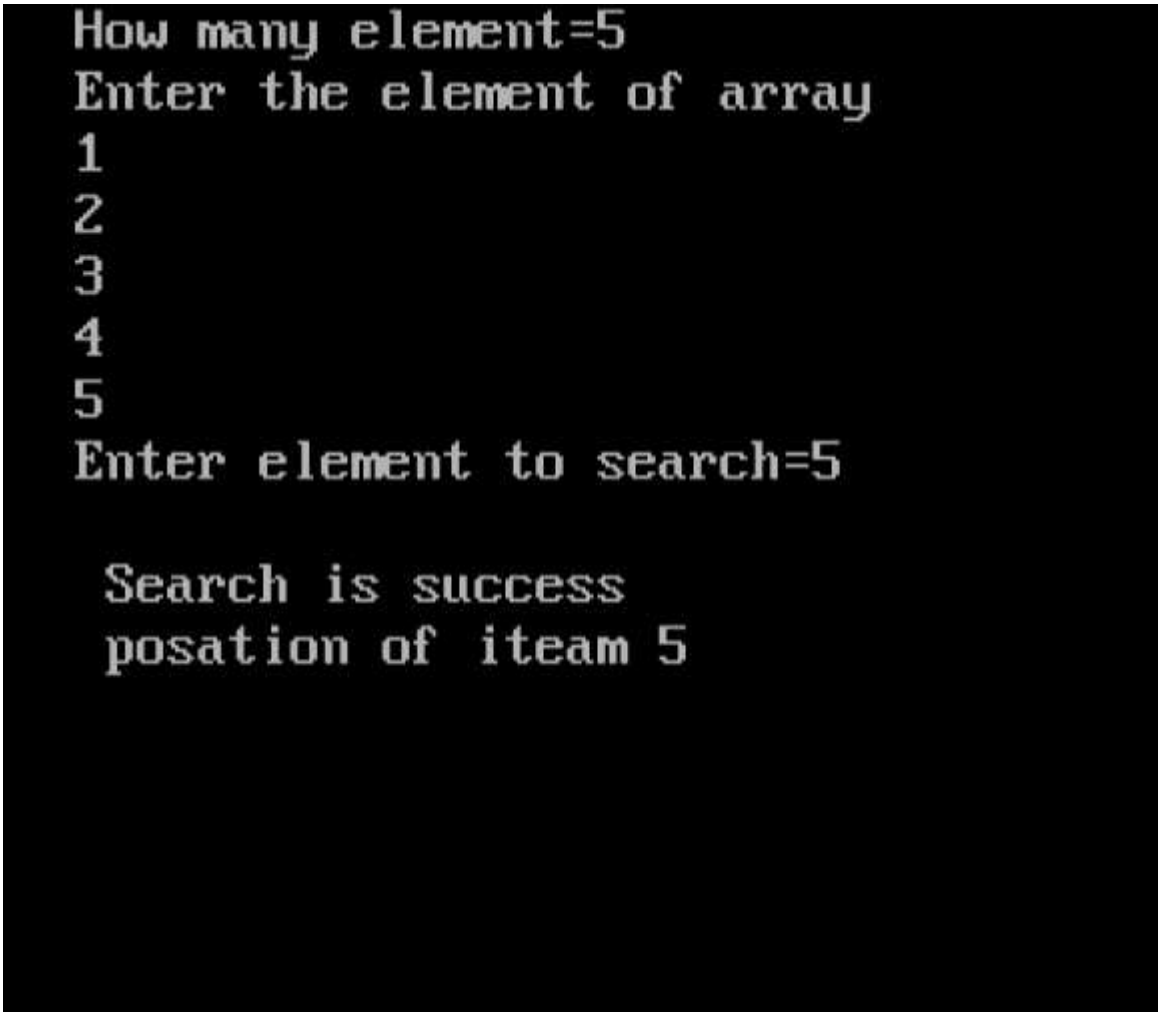
Aim: Write a program to implements the following i) Binary Search ii) Merge sort.

i) Binary Search

```
//----------Binary Search :-----------
// Array element should be in shorted order

#include<stdio.h>
#include<conio.h>
void main()
{
 int a[50],i, loc=0, beg, end,mid, item,n, flag=0;
 clrscr();
 printf("How many element");
 scanf("%d",&n);
//-------------------Store element in Array------------
 printf("Enter the element of array");
 for(i=0;i<= n-1;i++)
 {
  scanf("%d",&a[i]);
 }
//-------------------------------------------------
//----------------Search Item--------------------
 printf("Enter element to search");
 scanf("%d", &item);
//-----------------------------------------------
 beg=0;
 end=n-1;

//---------------Binary Search --Logic-----------------
 while((beg<=end) && (item!=a[mid]) )
 {
    mid= ((beg+end)/2);
    if(item==a[mid])
    {
     printf("\n Search is success");
     loc=mid;
     printf("\n posation of iteam %d \n", loc+1);
     flag=flag+1;
    }
    else if(item< a[mid])
    {
      end=mid-1;
    }
    else
    {
     beg=mid+1;
    }
```

```
 }
//-------------------------------------------------
 if(flag==0)
 {
  printf("\n Iteam is not found");
 }
 getch();
}
```

Output:-



EXPERIMENT-10

Aim: Write a program to implements Merge sort.

```
//--------Merge Sort-------------//

#include<stdio.h>
#include<conio.h>
int n;
void main()
{
  int i,low,high;
  int A[10];
  void MSort(int A[10],int low, int high);
  void Display(int A[10]);
  clrscr();
  printf("\n Merge Sort");
  printf("\n Enter lenght of List");
  scanf("%d",&n);
  printf("\n Enter list element");
  for(i=0;i<n;i++)
  {
   scanf("%d",&A[i]);
  }
  low=0;
  high=n-1;
  MSort(A,low,high);
  Display(A);
  getch();
}
//----------------MergeSort--------------------------//
void MSort(int A[10],int low, int high)
{
  int mid;
  void combine(int A[10],int low,int mid, int high);
  if(low<high)
  {
   mid=(low+high)/2;// split the list at mid
   MSort(A,low,mid); //first sublist
   MSort(A,mid+1,high); //Second sub list
   combine(A,low,mid,high);// merge of two sublists
  }
}
```

```
//----------------combine--------------------------//
void combine(int A[10],int low, int mid, int high)
{
 int i,j ,k;
 int temp[10];
 i=low;
 k=low;
 j=mid+1;
 while(i<=mid && j<=high)
 {
  if (A[i]<=A[j])
  {
    temp[k]=A[i];
     i++;
     k++;
  }
  else
  {
   temp[k]=A[j];
   j++;
   k++;
  }
 }
 while(i<=mid)
 {
  temp[k]=A[i];
  i++;
  k++;
 }
 while(j<=high)
 {
  temp[k]=A[j];
  j++;
  k++;
 }
 //copy the element from temp array to A
 for(k=low; k<=high; k++)
 {
   A[k]=temp[k];
 }

}
//----------------Display Sorted Array--------------------------//
void Display(int A[10])
{
  int i;
  printf("\n The Sorted Array is");
  for(i=0;i<n;i++)
  {
```

```
    printf("\n %d", A[i]);
   }
 }
```

Output:-

```
Merge Sort
Enter lenght of List8

Enter list element4
1
3
10
2
6
5
12

The Sorted Array is
1
2
3
4
5
6
10
12_
```

EXPECTED VIVA VOICE QUESTIONS

1. What is Data Structure?

Data structure refers to the way data is organized and manipulated. It seeks to find ways to make data access more efficient. When dealing with the data structure, we not only focus on one piece of data but the different set of data and how they can relate to one another in an organized manner.

2. What is Link List?

A linked list is a sequence of nodes in which each node is connected to the node following it. This forms a chain-like link for data storage.

3. What is Queue?

A queue is a data structure that can simulate a list or stream of data. In this structure, new elements are inserted at one end, and existing elements are removed from the other end.

4. What are Binary Tree?

A binary tree is one type of data structure that has two nodes, a left node, and a right node. In programming, binary trees are an extension of the linked list structures.

5. What is Stack?

A stack is a data structure in which only the top element can be accessed. As data is stored in the stack, each data is pushed downward, leaving the most recently added data on top.

6. What is Merge Sort?

Merge sort, is a divide-and-conquer approach for sorting the data. In a sequence of data, adjacent ones are merged and sorted to create bigger sorted lists. These sorted lists are then merged again to form an even bigger sorted list, which continues until you have one single sorted list.

7. What is the difference between PUSH and POP?

Pushing and popping applies to the way data is stored and retrieved in a stack. A push denotes data being added to it, meaning data is being "pushed" into the stack. On the other hand, a pop denotes data retrieval, and in particular, refers to the topmost data being accessed.

8. What is a linear search?

A linear search refers to the way a target key is being searched in a sequential data structure. In this method, each element in the list is checked and compared against the target key. The process is repeated until found or if the end of the file has been reached.

9. What is a bubble sort and how do you perform it?

A bubble sort is one sorting technique that can be applied to data structures such as an array. It works by comparing adjacent elements and exchanges their values if they are out of order. This method lets the smaller values "bubble" to the top of the list, while the larger value sinks to the bottom.

10. What is Graph?

A graph is one type of data structure that contains a set of ordered pairs. These ordered pairs are also referred to as edges or arcs and are used to connect nodes where data can be stored and retrieved.

11. What is an AVL Tree?

An AVL tree is a type of binary search tree that is always in a state of partially balanced. The balance is measured as a difference between the heights of the subtrees from the root. This self-balancing tree was known to be the first data structure to be designed as such.

12. What are double linked list?

Doubly linked lists are a special type of linked list wherein traversal across the data elements can be done in both directions. This is made possible by having two links in every node, one that links to the next node and another one that connects to the previous node.

VISION OF THE DEPARTMENT

To be a centre of excellence for providing quality technical education to develop future leaders with the aspects of research &computing, Software product development and entrepreneurship.

MISSION OF THE DEPARTMENT

M1: To offer academic programme with state of art curriculum having flexibility for accommodating the latest developments in the areas of Computer science engineering

M2: To conduct research and development activities in contemporary and emerging areas of computer science & engineering.

M3: To inculcate moral values & entrepreneurial skills to produce professionals capable of providing socially relevant and sustainable solutions.