MALLA REDDY COLLEGE OF ENGINEERING
(Formerly CM Engineering College)
Approved by AICTE. & Permanently Affiliated to JNTUH; ISO 9001:2008 Certified Institution
Maisammaguda, Dhulapally (post via kompally) secunderabad-500100. Tel:040-64632248,Mobile:9348161222,9346162620
E-mail: principal@mrce.in ,cmecprincipal@gmail.com; web: www.mrce.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B. TECH
II-I SEMESTER

DATA STRUCTURES

LAB MANUAL

Subject Code : CS306PC

Regulation : R22/JNTUH

Academic Year : 2023-2024

INSTITUTE VISION

To emerge as a Center of Excellence for producing professionals who shall be the leaders in technology innovation, entrepreneurship, management and in turn contribute for advancement of society and human kind**.**

INSTITUTE MISSION

- To provide an environment of learning in emerging technologies.
- To nurture a state of art teaching-learning process and R&D culture.
- To foster networking with alumni, Industry, Institutes of repute and other stake holders for effective interaction.
- To practice and promote high standards of ethical values through societal commitment.

DEPARTMENT VISION

To impact futuristic knowledge in computer science to produce highly skilled, imaginative and socially mindful experts who can contribute to industry and architect research fit for working in worldwide condition.

DEPARTMENT MISSION

- To promote strong academic growth by providing fundamental domain knowledge and offering state of art technology for having an excellence in Research & Development.
- To create an environment for learning analytical skills, advanced programming languages using modern tools and to equip for higher studies.
- To undertake collaborative projects for understanding need of a team work in real time environment and to improve communication and interpersonal skills for better employability.
- To promote high standards of ethical values through societal commitment.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO-1: To make the students understand and implement the engineering concepts in multiple domains.

PEO-2: To provide knowledge based services so as to meet the needs of the society and industry by usage of modern tools.

PEO-3: To understand engineering processes for design and development of software components and products efficiently for improving employability.

PEO-4: To educate students in disseminating the research findings to create interest for higher studies.

PEO-5: To inculcate knowledge with due consideration for ethical and economic issues.

PROGRAM OUTCOMES (POs)

PO-01 **ENGINEERING KNOWLEDGE**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO-02 **PROBLEM ANALYSIS**: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO-03 **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.

PO-04 **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.

PO-05 **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.

PO-06 **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.

PO-07 **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.

PO-08 **ETHICS**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO-09 **INDIVIDUAL AND TEAM WORK**: Function effectively as an individual,and as a member or leader in diverse teams, and in multidisciplinary settings.

PO-10 **COMMUNICATION**: Communicate effectively on complex engineeringactivities 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, give and receive clear instructions.

PO-11 **PROJECT MANAGEMENT AND FINANCE**: Demonstrate knowledge and understanding of the engineering and management principles and applythese to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO-12 **LIFE-LONG LEARNING**: Recognize the need for, and have the preparationand ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSO's)

PSO-1: PROFESSIONAL SKILLS: The ability to understand, analyze and develop computer programs in the areas related to algorithms and System Software.

PSO-2: PROBLEM SOLVING SKILLS: The ability to apply standard practices and strategies in software project development to deliver a quality and defect free product.

PSO-3: EMPLOYABILITY SKILLS: The ability to employ modern computer languages and technologies, so as to be industry ready and for better employability and research.

SYLLABUS

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, HYDERABAD

II Year B.Tech. CSE I-Sem

L	T	P	C
0	0	3	1.5

PREREQUISITES:

A course on "Programming for Problem Solving"

COURSE OBJECTIVES:

- It covers various concepts of C programming language
- It introduces searching and sorting algorithms
- It provides an understanding of data structures such as stacks and queues.

COURSE OUTCOME

- Ability to develop C programs for computing and real-life applications using basic elements like control statements, arrays, functions, pointers and strings, and data structures like stacks, queues and linked lists.
- Ability to Implement searching and sorting algorithms

List of Experiments:

1. Write a program that uses functions to perform the following operations on singly linkedlist.:
   i) Creation ii) Insertion iii) Deletion iv) Traversal
2. Write a program that uses functions to perform the following operations on doubly linkedlist.:
   i) Creation ii) Insertion iii) Deletion iv) Traversal
3. Write a program that uses functions to perform the following operations on circular linkedlist.:
   i) Creation ii) Insertion iii) Deletion iv) Traversal
4. Write a program that implement stack (its operations) using
   i) Arrays ii) Pointers
5. Write a program that implement Queue (its operations) using
   i) Arrays ii) Pointers
6. Write a program that implements the following sorting methods to sort a given list of integersin ascending order
   i) Quick sort ii) Heap sort iii) Merge sort
7. Write a program to implement the tree traversal methods (Recursive and Non Recursive).
8. Write a program to implement
   i) Binary Search tree ii) B Trees iii) B+ Trees iv) AVLtrees
   v) Red - Black trees
9. Write a program to implement the graph traversal methods.
10. Implement a Pattern matching algorithms using Boyer- Moore, Knuth-Morris-Pratt

TEXT BOOKS:

1. Fundamentals of Data Structures in C, 2nd Edition, E. Horowitz, S. Sahni and Susan AndersonFreed, Universities Press.
2. Data Structures using C – A. S. Tanenbaum, Y. Langsam, and M. J. Augenstein, PHI/PearsonEducation.

REFERENCE BOOK:

1. Data Structures: A Pseudocode Approach with C, 2nd Edition, R. F. Gilberg and B. A. Forouzan,Cengage Learning.

Course objectives

1.	It covers various concepts of C programming language.
2.	It introduces searching and sorting algorithms.
3.	It provides an understanding of data structures such as stacks and queues.

Course Outcomes

Course Outcome	Course Outcome Statement	Bloom's Taxonomy level
CO-1	Ability to develop C programs for computing and real-life applications using basic elements like control statements, arrays, functions, pointers and strings, and data structures like stacks, queues and linked lists.	Apply
CO-2	Ability to Implement searching and sorting algorithms.	Create

COURSE: DATA STRUCTURES LAB(CS306PC)

CLASS / SEM: II B.TECH. CSE /I SEM **A.Y:2023-24**

CO-PO MAPPING

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO1	PSO2	PSO3
CO-1	1	2	2		3								1	2	
CO-2	1	1	2		3								1	1	
Avg.	**1**	**1.5**	**2**		**3**								**1**	**1.5**	

***(Indicators**: 1-Strong, 2-Average, 3-Low.)*

CO-PO MAPPING FOR OPEN ENDED PROGRAM

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO1	PSO2	PSO3
CO-1	1	2	2		3								1	2	
CO-2	1	3	2		3								1	2	
Avg.	**1**	**2.5**	**2**		**3**								**1**	**2**	

***(Indicators**: 1-Strong, 2-Average, 3-Low.)*

DATA STRUCTURES LAB INDEX

EXPERIMENT- 1

Write a program that uses functions to perform the following operations on singly linked list: i) Creation ii) Insertion iii) Deletion iv) Traversal

```
#include<stdio.h>
#include<stdlib.h>
struct node
 {
int data;
struct node *next;
 };
struct node *head;
void beginsert  ();
void  lastinsert  ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
 {
int choice =0;
while(choice != 9)
 {
printf("\n\n*********Main Menu*********\n");
printf("\nChoose one option from the following list ...\n");
printf("\n===============================================\n");
printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete fro m Beginning\n 5.Delete from last\n6.Delete node after specified location\n7.Search for an el ement\n8.Show\n9.Exit\n");
printf("\nEnter your choice?\n");
scanf("\n%d",&choice); switch(choice)
 {
case 1:
beginsert();
break;
case 2:
lastinsert();
break;
case 3:
randominsert();
break;
case 4:
begin_delete(); break;
case 5:
last_delete();
```

```
break;
case 6:
random_delete();
break;
case 7:
search();
break;
case 8:
display();
break;
case 9:
exit(0);
break;
default:
printf("Please enter valid choice..");
  }
  }
  }
void beginsert()
  {
struct node *ptr;
int item;
ptr = (struct node *) malloc(sizeof(struct node *));
if(ptr == NULL)
  {
printf("\nOVERFLOW");
  }
else
  {
printf("\nEnter value\n");
scanf("%d",&item);
ptr->data  =  item;
ptr->next  =  head;
head = ptr;
printf("\nNode inserted");
  }
  }
void lastinsert()
  {
struct node *ptr,*temp; int
item;
ptr = (struct node*)malloc(sizeof(struct node));
if(ptr == NULL)
  {
printf("\nOVERFLOW");
  }
else
  {
printf("\nEnter value?\n");
scanf("%d",&item);
```

```
ptr->data = item;
if(head == NULL)
  {
ptr -> next = NULL;  head
= ptr;  printf("\nNode
inserted");
  }
else
  {
temp = head;
while (temp -> next != NULL)
  {
temp = temp -> next;
  }
temp->next = ptr;
ptr->next = NULL;
printf("\nNode inserted");
  }
  }
  }
void randominsert()
  {
int i,loc,item;
struct node *ptr, *temp;
ptr = (struct node *) malloc (sizeof(struct node));
if(ptr == NULL)
  {
printf("\nOVERFLOW");
  }
else
  {
printf("\nEnter element value");
scanf("%d",&item);
ptr->data = item;
printf("\nEnter the location after which you want to insert ");
scanf("\n%d",&loc);
temp=head;
for(i=0;i<loc;i++)
  {
temp = temp->next;
if(temp == NULL)
  {
printf("\ncan't insert\n");
return;
  }
  }
ptr ->next = temp ->next;
temp ->next = ptr;
printf("\nNode inserted");
  }
```

```
 }
void begin_delete()
 {
struct node *ptr;
if(head == NULL)
 {
printf("\nList is empty\n");
 }
else
 {
ptr = head;
head = ptr->next;
free(ptr);
printf("\nNode deleted from the begining ...\n");
 }
 }
void last_delete()
 {
struct node *ptr,*ptr1;
if(head == NULL)
 {
printf("\nlist is empty");
 }
else if(head -> next == NULL)
 {
head = NULL;
free(head);
printf("\nOnly node of the list deleted ...\n");
 }
else
 {
ptr = head;
while(ptr->next != NULL)
 {
ptr1 = ptr;
ptr = ptr ->next;
 }
ptr1->next = NULL;
free(ptr);
printf("\nDeleted Node from the last ...\n");
 }
 }
void random_delete()
 {
struct node *ptr,*ptr1;
int loc,i;
printf("\n Enter the location of the node after which you want to perform deletion \n");
scanf("%d",&loc);
ptr=head;
for(i=0;i<loc;i++)
```

```
  {
ptr1 = ptr;
ptr = ptr->next;
if(ptr == NULL)
  {
printf("\nCan't delete");
return;
  }
  }
ptr1 ->next = ptr ->next;
free(ptr);
printf("\nDeleted node %d ",loc+1);
  }
void search()
  {
struct  node  *ptr;
int   item,i=0,flag;
ptr = head;
if(ptr == NULL)
  {
printf("\nEmpty List\n");
  }
else
  {
printf("\nEnter item which you want to search?\n");
scanf("%d",&item);
while (ptr!=NULL)
  {
if(ptr->data == item)
  {
printf("item found at location %d ",i+1);
flag=0;
  }
else
  {
flag=1;
}
i++;
ptr = ptr -> next;
  }
if(flag==1)
  {
printf("Item not found\n");
  }
  }
  }
void display()
  {
struct node *ptr;
ptr = head;
```

```
if(ptr == NULL)
  {
printf("Nothing to print");
  }
else
  {
printf("\nprinting values..... \n");
while (ptr!=NULL)
  {
printf("\n%d",ptr->data);
ptr = ptr -> next;
  }
  }
  }
```

Output:

```
*********Main Menu*********
Choose one option from the following list ...
  ================================================
  1. Insert in begining
  2. Insert at last
  3. Insert at any random location
  4. Delete from Beginning
  5. Delete from last
  6. Delete node after specified location
  7. Search for an element
  8. Show
  9. Exit
Enter your choice? 1
Enter value
1
Node inserted
*********Main Menu*********
Choose one option from the following list ...
  ================================================
  1. Insert in begining
  2. Insert at last
  3. Insert at any random location
  4. Delete from Beginning
  5. Delete from last
  6. Delete node after specified location
  7. Search for an element
  8. Show
  9. Exit
Enter your choice? 2
Enter value?
2
Node inserted
```

*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete node after specified location
7. Search for an element
8. Show
9. Exit
Enter your choice? 3
Enter element value1
Enter the location after which you want to insert 1
Node inserted
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete node after specified location
7. Search for an element
8. Show
9. Exit
Enter your choice? 8
printing values . . . . .
1
2
1
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete node after specified location
7. Search for an element
8. Show
9. Exit
Enter your choice? 2
Enter value?

123
Node inserted
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete node after specified location
7. Search for an element
8. Show
9. Exit
Enter your choice? 1
Enter value
1234
Node inserted
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete node after specified location
7. Search for an element
8. Show
9. Exit
Enter your choice? 4
Node deleted from the begining ...
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete node after specified location
7. Search for an element
8. Show
9. Exit
Enter your choice? 5
Deleted Node from the last ...
*********Main Menu*********

Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete node after specified location
7. Search for an element
8. Show
9. Exit
Enter your choice? 6
Enter the location of the node after which you want to perform deletion 1
Deleted node 2
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete node after specified location
7. Search for an element
8. Show
9. Exit
Enter your choice? 8
printing values . . . . .
1
1
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete node after specified location
7. Search for an element
8. Show
9. Exit
Enter your choice? 7
Enter item which you want to search?
1
item found at location 1

item found at location 2
*********Main Menu*********
Choose one option from the following list ...
==========

1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete node after specified location
7. Search for an element
8. Show
9. Exit

Enter your choice? 9

EXPERIMENT- 2

Write a program that uses functions to perform the following operations on doubly linked list.: Creation ii) Insertion iii) Deletion iv) Traversal

```
#include<stdio.h>
#include<stdlib.h>
struct node
 {
struct  node  *prev;
struct  node  *next;
int data;
 };
struct node *head;
void  insertion_beginning();
void insertion_last();
void   insertion_specified();
void   deletion_beginning();
void deletion_last();
void deletion_specified();
void display();
void search();
void main ()
 {
int choice =0;
while(choice != 9)
 {
printf("\n*********Main Menu*********\n");
printf("\nChoose one option from the following list ...\n");
printf("\n==========================================\n");
printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete
from Beginning\n
```

```
5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Show\n9.Exit\n"
 );
printf("\nEnter your choice?\n");
scanf("\n%d",&choice);
switch(choice)
 {
case 1:
insertion_beginning();
break;
case 2:
insertion_last();
break;
case 3:
insertion_specified();
break;
case 4:
deletion_beginning();
break;
case 5:
deletion_last(); break;
case 6:
deletion_specified();
break;
case 7:
search();
break;
case 8:
display();
break;
case 9:
exit(0);
break;
default:
printf("Please enter valid choice..");
 }
 }
 }
void insertion_beginning()
 {
struct node *ptr;
int item;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
 {
printf("\nOVERFLOW");
 }
else
 {
printf("\nEnter Item value");
scanf("%d",&item);
if(head==NULL)
```

```
 {
ptr->next = NULL;
ptr->prev=NULL;
ptr->data=item;
head=ptr;
 }
else
 {
ptr->data=item;
ptr->prev=NULL;
ptr->next = head;
head->prev=ptr;
head=ptr;
 }
printf("\nNode inserted\n");
 }
 }
void insertion_last()
 {
struct node *ptr,*temp; int
item;
ptr = (struct node *) malloc(sizeof(struct node));
if(ptr == NULL)
 {
printf("\nOVERFLOW");
 }
else
 {
printf("\nEnter value");
scanf("%d",&item); ptr-
>data=item;  if(head ==
NULL)
 {
ptr->next  =  NULL;
ptr->prev  =  NULL;
head = ptr;
 }
else
 {
temp = head;
while(temp->next!=NULL)
 {
temp = temp->next;
 }
temp->next = ptr;
ptr ->prev=temp;
ptr->next = NULL;
 }
 }
printf("\nnode inserted\n");
 }
```

```
void insertion_specified()
 {
struct node *ptr,*temp; int
item,loc,i;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
 {
printf("\n OVERFLOW");
 }
else
 {
temp=head;
printf("Enter the location");
scanf("%d",&loc);
for(i=0;i<loc;i++)
 {
temp = temp->next;
if(temp == NULL)
 {
printf("\n There are less than %d elements", loc);
return;
 }
 }
printf("Enter value");
scanf("%d",&item);
ptr->data = item;
ptr->next = temp->next;
ptr -> prev = temp;
temp->next = ptr;
temp->next->prev=ptr;
printf("\nnode inserted\n");
 }
 }
void deletion_beginning()
 {
struct node *ptr;
if(head == NULL)
 {
printf("\n UNDERFLOW");
 }
else if(head->next == NULL)
 {
head = NULL;
free(head);
printf("\nnode deleted\n");
 }
else
 {
ptr = head;
head = head -> next;
head -> prev = NULL;
```

```
free(ptr);
printf("\nnode deleted\n");
  }
  }
void deletion_last()
  {
struct node *ptr;
if(head == NULL)
  {
printf("\n UNDERFLOW");
  }
else if(head->next == NULL)
  {
head = NULL;
free(head);
printf("\nnode deleted\n");
  }
else
  {
ptr = head;
if(ptr->next != NULL)
  {
ptr = ptr -> next;
  }
ptr -> prev -> next = NULL;
free(ptr);
printf("\nnode deleted\n");
  }
  }
void deletion_specified()
  {
struct node *ptr, *temp;
int val;
printf("\n Enter the data after which the node is to be deleted : ");
scanf("%d", &val);
ptr = head;
while(ptr -> data != val)
ptr = ptr -> next;
if(ptr -> next == NULL)
  {
printf("\nCan't delete\n");
  }
else if(ptr -> next -> next == NULL)
  {
ptr ->next = NULL;
  }
else
  {
temp = ptr -> next;
ptr -> next = temp -> next;
```

```
temp -> next -> prev = ptr;
free(temp);
printf("\nnode deleted\n");
  }
  }
void display()
  {
struct node *ptr;
printf("\n printing values...\n");
ptr = head;
while(ptr != NULL)
  {
printf("%d\n",ptr->data);
ptr=ptr->next;
  }
  }
void search()
  {
struct  node  *ptr;
int   item,i=0,flag;
ptr = head;
if(ptr == NULL)
  {
printf("\nEmpty List\n");
  }
else
  {
printf("\nEnter item which you want to search?\n");
scanf("%d",&item);
while (ptr!=NULL)
  {
if(ptr->data == item)
  {
printf("\nitem found at location %d ",i+1);
flag=0;
break;
  }
else
  {
flag=1;
}
i++;
ptr = ptr -> next;
  }
if(flag==1)
  {
printf("\nItem not found\n");
  }
  }
  }
```

Output

```
*********Main Menu*********
Choose one option from the following list ...
 ==============================================
 1. Insert in begining
 2. Insert at last
 3. Insert at any random location
 4. Delete from Beginning
 5. Delete from last
 6. Delete the node after the given data
 7. Search
 8. Show
 9. Exit
Enter your choice? 8
printing values...
*********Main Menu*********
Choose one option from the following list ...
 ==============================================
 1. Insert in begining
 2. Insert at last
 3. Insert at any random location
 4. Delete from Beginning
 5. Delete from last
 6. Delete the node after the given data
 7. Search
 8. Show
 9. Exit
Enter your choice? 1
Enter Item value12
Node inserted
*********Main Menu*********
Choose one option from the following list ...
 ==============================================
 1. Insert in begining
 2. Insert at last
 3. Insert at any random location
 4. Delete from Beginning
 5. Delete from last
 6. Delete the node after the given data
 7. Search
 8. Show
 9. Exit
Enter your choice? 1
Enter Item value123
Node inserted

 *********Main Menu*********
 Choose one option from the following list ...
```

```
==============================================
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit
Enter your choice? 1
Enter Item value1234
Node inserted
*********Main Menu*********
Choose one option from the following list ...
==============================================
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit
Enter your choice? 8
printing values...
1234
123
12
*********Main Menu*********
Choose one option from the following list ...
==============================================
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit
Enter your choice? 2
Enter value89
node inserted


*********Main Menu*********
Choose one option from the following list ...
```

```
==============================================
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit
Enter your choice? 3
Enter the location1
Enter value12345
node inserted
*********Main Menu*********
Choose one option from the following list ...
==============================================
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit
Enter your choice? 8
printing values...
1234
123
12345
12
89
*********Main Menu*********
Choose one option from the following list ...
==============================================
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit
Enter your choice? 4
node deleted
```

*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit
Enter your choice? 5
node deleted
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit
Enter your choice? 8
printing values...
123
12345
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit
Enter your choice? 6
Enter the data after which the node is to be deleted : 123
*********Main Menu*********
Choose one option from the following list ...
==========

1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit

Enter your choice? 8
printing values...
123
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit

Enter your choice? 7
Enter item which you want to search?
123
item found at location 1
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit

Enter your choice? 6
Enter the data after which the node is to be deleted : 123 Can't delete
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining

2. Insert at last
3. Insert at any random location
4. Delete from Beginning
5. Delete from last
6. Delete the node after the given data
7. Search
8. Show
9. Exit

Enter your choice? 9
Exited.

EXPERIMENT- 3

Write a program that uses functions to perform the following operations on circular linked list.: Creation ii) Insertion iii) Deletion iv) Traversal

```
#include<stdio.h>
#include<stdlib.h>
struct node
 {
int data;
struct node *next;
 };
struct node *head;
void beginsert  ();
void  lastinsert  ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
 {
int choice =0;
while(choice != 7)
 {
printf("\n*********Main Menu*********\n");
printf("\nChoose one option from the following list ...\n");
printf("\n===============================================\n");
printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from
last\n5.Search for an element\n6.Show\n7.Exit\n");
printf("\nEnter your choice?\n");
scanf("\n%d",&choice); switch(choice)
 {
case 1:
beginsert();
break;
```

```
case 2:
lastinsert();
break;
case 3:
begin_delete(); break;
case 4:
last_delete();
break;
case 5:
search();
break;
case 6:
display();
break;
case 7:
exit(0);
break;
default:
printf("Please enter valid choice..");
  }
  }
  }
void beginsert()
  {
struct node *ptr,*temp; int
item;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
  {
printf("\nOVERFLOW");
  }
else
  {
printf("\nEnter the node data?");
scanf("%d",&item);
ptr -> data = item;
if(head == NULL)
  {
head = ptr;
ptr -> next = head;
  }
else
  {
temp = head;
while(temp->next != head) temp
= temp->next;
ptr->next = head;
temp -> next = ptr;
head = ptr;
  }
printf("\nnode inserted\n");
```

```
  }
  }
void lastinsert()
  {
struct node *ptr,*temp; int
item;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
  {
printf("\nOVERFLOW\n");
  }
else
  {
printf("\nEnter Data?");
scanf("%d",&item);
ptr->data = item;
if(head == NULL)
  {
head = ptr;
ptr -> next = head;
  }
else
  {
temp = head;
while(temp -> next != head)
  {
temp = temp -> next;
  }
temp -> next = ptr;
ptr -> next = head;
  }
printf("\nnode inserted\n");
  }
  }
void begin_delete()
  {
struct node *ptr;
if(head == NULL)
  {
printf("\nUNDERFLOW");
  }
else if(head->next == head)
  {
head = NULL;
free(head);
printf("\nnode deleted\n");
  }
else
{ ptr = head;
while(ptr -> next != head)
```

```
ptr = ptr -> next;
ptr->next = head->next;
free(head);
head = ptr->next;
printf("\nnode deleted\n");
  }
  }
void last_delete()
  {
struct node *ptr, *preptr;
if(head==NULL)
  {
printf("\nUNDERFLOW");
  }
else if (head ->next == head)
  {
head = NULL;
free(head);
printf("\nnode deleted\n");
  }
else
  {
ptr = head;
while(ptr ->next != head)
  {
preptr=ptr;
ptr = ptr->next;
  }
preptr->next = ptr -> next;
free(ptr);
printf("\nnode deleted\n");
  }
  }
void search()
  {
struct node *ptr;  int
item,i=0,flag=1; ptr
= head;
if(ptr == NULL)
  {
printf("\nEmpty List\n");
  }
else
  {
printf("\nEnter item which you want to search?\n");
scanf("%d",&item);
if(head ->data == item)
  {
printf("item found at location %d",i+1);
flag=0;
```

```
  }
else
  {
while (ptr->next != head)
  {
if(ptr->data == item)
  {
printf("item found at location %d ",i+1);
flag=0;
break;
  }
else
  {
flag=1;
}
i++;
ptr = ptr -> next;
  }
  }
if(flag != 0)
  {
printf("Item not found\n");
  }
  }
  }
void display()
  {
struct node *ptr;
ptr=head;
if(head == NULL)
  {
printf("\nnothing to print");
  }
else
  {
printf("\n printing values ... \n");
while(ptr -> next != head)
  {
printf("%d\n", ptr -> data);
ptr = ptr -> next;
  }
printf("%d\n", ptr -> data);
  }
  }
```

Output:

*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Search for an element
6. Show
7. Exit
Enter your choice? 1
Enter the node data?10
node inserted
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Search for an element
6. Show
7. Exit
Enter your choice? 2
Enter Data?20
node inserted
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Search for an element
6. Show
7. Exit
Enter your choice? 2
Enter Data?30
node inserted
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Search for an element
6. Show
7. Exit
Enter your choice? 3

node deleted
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Search for an element
6. Show
7. Exit
Enter your choice? 4
node deleted
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Search for an element
6. Show
7. Exit
Enter your choice? 5
Enter item which you want to search?
20
item found at location 1
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Search for an element
6. Show
7. Exit
Enter your choice? 6
printing values ...
20
*********Main Menu*********
Choose one option from the following list ...
==========
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Search for an element

6. Show
7. Exit

Enter your choice? 7

EXPERIMENT- 4

Write a program that implement stack (its operations) using i) Arrays ii) Pointers

```
#include <stdio.h>
int stack[100],i,j,choice=0,n,top=-1;
void push();
void pop();
void show();
void main ()
 {
printf("Enter the number of elements in the stack ");
scanf("%d",&n);
printf("*********Stack operations using array*********");
printf("\n----------\n");
while(choice != 4)
 {
printf("Chose one from the below options...\n");
printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
printf("\n Enter your choice \n");
scanf("%d",&choice);
switch(choice)
 {
case 1:
 {
push();
break;
 }
case 2:
 {
pop();
break;
 }
case 3:
 {
show();
break;
 }
case 4:
 {
printf("Exiting..... ");
break;
 }
default:
```

```
 {
printf("Please Enter valid choice ");
 }
 }
 }
 }
void push ()
 {
int val;
if (top == n ) printf("\n
Overflow"); else
 {
printf("Enter the value?");
scanf("%d",&val);
top = top +1;
stack[top] = val;
 }
 }
void pop ()
 {
if(top == -1)
printf("Underflow");
else
top = top -1;
 }
void show()
 {
for (i=top;i>=0;i--)
 {
printf("%d\n",stack[i]);
 }
if(top == -1)
 {
printf("Stack is empty");
 }
 }
```

ii)Pointers

```
#include <stdio.h>
#include<stdlib.h>
#include<conio.h>
int stack[100],i,j,choice=0,n,top=-1;
void push();
void pop();
void show();
void main ()
 {
printf("Enter the number of elements in the stack ");
scanf("%d",&n);
```

```
printf("*********Stack operations using array*********");
printf("\n.....\n");
while(choice != 4)
  {
printf("Chose one from the below options...\n");
printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
printf("\n Enter your choice \n");
scanf("%d",&choice);
switch(choice)
  {
case 1:

                    push();
                    break;

case 2:

                    pop();
                    break;

case 3:

                    show();
                    break;

case 4:
                   exit(0);
                   break;

default:

                    printf("Please Enter valid choice ");

  }
  }
  }
void push ()
  {
int val;
if (top == n ) printf("\n
Overflow"); else
  {
printf("Enter the value?");
scanf("%d",&val);
top = top +1;
stack[top] = val;
  }
  }
void pop ()
```

```
 {
if(top == -1)
printf("Underflow");
else
top = top -1;
 }
void show()
 {
for (i=top;i>=0;i--)
 {
printf("%d\n",stack[i]);
 }
if(top == -1)
 {
printf("Stack is empty");
 }
 }
```

Output:

```
Chose one from the below options...

 1. Push
 2. Pop
 3. Show
 4. Exit
 Enter your choice
1
Enter the value?          35
Chose one from the below options...

 1. Push
 2. Pop
 3. Show
 4. Exit
 Enter your choice
1
Enter the value?          52
Chose one from the below options...

 1. Push
 2. Pop
 3. Show
 4. Exit
 Enter your choice
2
Chose one from the below options...

 1. Push
 2. Pop
```

3. Show
4. Exit
Enter your choice
3
35
Chose one from the below options...

1. Push
2. Pop
3. Show
4. Exit
Enter your choice 4

EXPERIMENT- 5

Write a program that implement Queue (its operations) using i) Arrays ii) Pointers

```
#include<stdio.h>
#include<stdlib.h>
#define maxsize 5
void insert();
void delete();
void display();
int front = -1, rear = -1;
int queue[maxsize];
void main ()
 {
int choice;
while(choice != 4)
 {
printf("\n*************************Main Menu*****************************\n");
printf("\n==========================================================
=====\n");
printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");
printf("\nEnter your choice ?");
scanf("%d",&choice);
switch(choice)
 {
case 1:
insert();
break;
case 2:
delete();
break;
case 3:
display();
break;
case 4:
exit(0);
```

```
break;
default:
printf("\nEnter valid choice??\n");
  }
  }
  }
void insert()
  {
int item;
printf("\nEnter the element\n");
scanf("\n%d",&item);
if(rear == maxsize-1)
  {
printf("\nOVERFLOW\n");
return;
  }
if(front == -1 && rear == -1)
  {
front = 0;
rear = 0;
  }
else
  {
rear = rear+1;
  }
queue[rear] = item;
printf("\nValue inserted ");
  }
void delete()
  {
int item;
if (front == -1 || front > rear)
  {
printf("\nUNDERFLOW\n");
return;
  }
else
  {
item = queue[front];
if(front == rear)
  {
front = -1;
rear = -1 ;
  }
else
  {
front = front + 1;
  }
printf("\nvalue deleted ");
  }
  }
```

```
void display()
  {
int i;
if(rear == -1)
  {
printf("\nEmpty queue\n");
  }
else
{ printf("\nprinting values..... \n");
for(i=front;i<=rear;i++)
  {
printf("\n%d\n",queue[i]);
  }
  }
  }
```

ii)Pointers

```
#include<stdio.h>
#include<stdlib.h>
struct node
  {
int data;
struct node *next;
  };
struct node *front;
struct  node  *rear;
void insert();
void delete();
void display();
void main ()
  {
int choice;
while(choice != 4)
  {
printf("\n*************************Main Menu*****************************\n");
printf("\n================================================================
=====\n");
printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");
printf("\nEnter your choice ?");
scanf("%d",& choice);
switch(choice)
  {
case 1:
insert();
break;
case 2:
delete();
break;
case 3:
```

```
display();
break;
case 4:
exit(0);
break;
default:
printf("\nEnter valid choice??\n");
  }
  }
  }
void insert()
  {
struct node *ptr;
int item;
ptr = (struct node *) malloc (sizeof(struct node));
if(ptr == NULL)
  {
printf("\nOVERFLOW\n");
return;
  }
else
  {
printf("\nEnter value?\n");
scanf("%d",&item);
ptr -> data = item;
if(front == NULL)
  {
front = ptr;
rear = ptr;
front -> next = NULL;
rear -> next = NULL;
  }
else
  {
rear -> next = ptr;
rear = ptr;
rear->next = NULL;
  }
  }
  }
void delete ()
  {
struct node *ptr;
if(front == NULL)
  {
printf("\nUNDERFLOW\n");
return;
  }
else
  {
```

```
ptr = front;
front = front -> next;
free(ptr);
  }
  }
void display()
  {
struct node *ptr; ptr
= front;  if(front ==
NULL)
  {
printf("\nEmpty queue\n");
  }
else
{ printf("\nprinting values..... \n");
while(ptr != NULL)
  {
printf("\n%d\n",ptr -> data);
ptr = ptr -> next;
  }
  }
  }
```

Output:

```
**********Main Menu**********
  ==============================
  1. insert an element
  2. Delete an element
  3. Display the queue
  4. Exit
Enter your choice ?1
Enter value?
123
**********Main Menu**********
  ==============================
  1. insert an element
  2. Delete an element
  3. Display the queue
  4. Exit
Enter your choice ?1
Enter value?
90
**********Main Menu**********
  ==============================
  1. insert an element
  2. Delete an element
  3. Display the queue
  4. Exit
Enter your choice ?3
```

```
printing values .....
123
90
**********Main Menu**********
 ==============================
                                                    1. insert an element
                                                    2. Delete an element
                                                    3. Display the queue
                                                    4. Exit
Enter your choice ?2
**********Main Menu**********
 ==============================
 1. insert an element
 2. Delete an element
 3. Display the queue
 4. Exit
Enter your choice ?3
printing values ......90
**********Main Menu**********
 ==============================
 1. insert an element
 2. Delete an element
 3. Display the queue
 4. Exit
Enter your choice ?4
```

EXPERIMENT- 6

Write a program that implements the following sorting methods to sort a given list of integers
in ascending order
Quick sort ii) Heap sort iii) Merge sort

i) Quick Sort

```
#include<stdio.h>
void quicksort(int number[25],int first,int last){
   int i, j, pivot, temp;

   if(first<last){
      pivot=first;
      i=first;
      j=last;

      while(i<j){
         while(number[i]<=number[pivot]&&i<last)
            i++;
         while(number[j]>number[pivot])
            j--;
```

```
        if(i<j){
          temp=number[i];
          number[i]=numb
          er[j];
          number[j]=temp;
        }
      }

      temp=number[pivot];
      number[pivot]=number[j];
      number[j]=temp;
      quicksort(number,first,j-1);
      quicksort(number,j+1,last);

    }
 }

int main(){
    int i, count, number[25];

    printf("How many elements are u going to enter?: ");
    scanf("%d",&count);

    printf("Enter %d elements: ", count);
    for(i=0;i<count;i++)
      scanf("%d",&number[i]);

    quicksort(number,0,count-1);

    printf("Order of Sorted elements: ");
    for(i=0;i<count;i++)
      printf(" %d",number[i]);

    return 0;
 }
```

Output:

```
How many elements are u going to enter?: 3
Enter 3 elements: 23 10 64
Order of Sorted elements: 10 23 64
 ---------------------------------
Process exited after 19.4 seconds with return value 0 Press
any key to continue . . .
```

ii) Heap sort

```
#include <stdio.h>

void maxHeapify(int arr[], int n, int i) {
    int largest = i;
```

```
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;

        maxHeapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
            int i;
    for (i = n / 2 - 1; i >= 0; i--)
        maxHeapify(arr, n, i);

    for (i = n - 1; i >= 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        maxHeapify(arr, i, 0);
    }
}

int main() {
    int n,i;

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];

    printf("Enter the elements: ");
    for ( i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    heapSort(arr, n);

    printf("Sorted array: ");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
```

```
    return 0;
}
```

Output

```
Enter the number of elements: 3
Enter the elements: 56 73 24
Sorted array: 24 56 73

--------------------------------
Process exited after 10.85 seconds with return value 0
Press any key to continue . . .
```

iii) Merge Sort

```
#include <stdio.h>

// Merge two subarrays of arr[]
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    // Create temporary arrays
    int L[n1], R[n2];

    // Copy data to temporary arrays L[] and R[]
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    // Merge the temporary arrays back into arr[l..r]
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
```

```
    // Copy the remaining elements of L[], if any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy the remaining elements of R[], if any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// l is for left index, and r is right index of the sub-array of arr to be sorted
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for large l and r
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

int main() {
    int n;

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n],i;

    printf("Enter the elements: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    mergeSort(arr, 0, n - 1);

    printf("Sorted array: ");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
```

```
    printf("\n");

    return 0;
}
```

Output

```
Enter the number of elements: 5
Enter the elements: 92 5 67 78 21
Sorted array: 5 21 67 78 92

--------------------------------
Process exited after 24.11 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT- 7

Write a program to implement the tree traversal methods (Recursive and Non Recursive).

i)Recursive

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

typedef struct Node Node;

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}
```

```
Node* insert(Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }

    return root;
}

void inOrderTraversal(Node* root) {
    if (root != NULL) {
        inOrderTraversal(root->left);
        printf("%d ", root->data);
        inOrderTraversal(root->right);
    }
}

void preOrderTraversal(Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preOrderTraversal(root->left);
        preOrderTraversal(root->right);
    }
}

void postOrderTraversal(Node* root) {
    if (root != NULL) {
        postOrderTraversal(root->left);
        postOrderTraversal(root->right);
        printf("%d ", root->data);
    }
}

int main() {
    Node* root = NULL;
    int choice, data;

    while (1) {
      printf("\n1. Insert\n2. In-order Traversal\n3. Pre-order Traversal\n4. Post-order
Traversal\n5. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the data to insert: ");
                scanf("%d", &data);
                root = insert(root, data);
```

```
            break;
        case 2:
            printf("In-order Traversal: ");
            inOrderTraversal(root);
            printf("\n");
            break;
        case 3:
            printf("Pre-order Traversal: ");
            preOrderTraversal(root);
            printf("\n");
            break;
        case 4:
            printf("Post-order Traversal: ");
            postOrderTraversal(root);
            printf("\n");
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }

   return 0;
}
```

Output:

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit

Enter your choice: 1 Enter
the data to insert: 5

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit

Enter your choice: 1 Enter
the data to insert: 2

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit

Enter your choice: 1 Enter
the data to insert: 8

```
  1. Insert
  2. In-order Traversal
  3. Pre-order Traversal
  4. Post-order Traversal
  5. Exit
Enter your choice: 2
In-order Traversal: 2 5 8

  1. Insert
  2. In-order Traversal
  3. Pre-order Traversal
  4. Post-order Traversal
  5. Exit
Enter your choice: 3
Pre-order Traversal: 5 2 8

  1. Insert
  2. In-order Traversal
  3. Pre-order Traversal
  4. Post-order Traversal
  5. Exit
Enter your choice: 4
Post-order Traversal: 2 8 5

  1. Insert
  2. In-order Traversal
  3. Pre-order Traversal
  4. Post-order Traversal
  5. Exit
Enter your choice: 5
```

ii)NonRecursive

```
#include <stdio.h>
#include <stdlib.h>

// Definition of a binary tree node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Structure for a stack node
struct StackNode {
    struct Node* data;
    struct StackNode* next;
};

// Function to create a new binary tree node
```

```
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
 }

// Function to create a new stack node
struct StackNode* createStackNode(struct Node* data) {
    struct StackNode* newNode = (struct StackNode*)malloc(sizeof(struct StackNode));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
 }

// Function to push a node onto the stack
 void push(struct StackNode** root, struct Node* data) {
    struct StackNode* newNode = createStackNode(data);
    newNode->next = *root;
    *root = newNode;
 }

// Function to pop a node from the stack struct
Node* pop(struct StackNode** root) {
    if (*root == NULL) {
        return NULL;
    }

    struct Node* popped = (*root)->data;
    struct StackNode* temp = *root;
    *root = (*root)->next;
    free(temp);

    return popped;
 }

// Function to check if the stack is empty
int isEmpty(struct StackNode* root) {
    return root == NULL;
 }

// Non-Recursive Inorder Traversal
 void inorderNonRecursive(struct Node* root) {
    struct StackNode* stack = NULL;
    struct Node* current = root;

    while (current != NULL || !isEmpty(stack)) {
        while (current != NULL) {
            push(&stack, current);
            current = current->left;
```

```
        }

        current = pop(&stack);
        printf("%d ", current->data);

        current = current->right;
    }
}

// Non-Recursive Preorder Traversal
void preorderNonRecursive(struct Node* root) {
    if (root == NULL) return;

    struct StackNode* stack = NULL;
    push(&stack, root);

    while (!isEmpty(stack)) {
        struct Node* current = pop(&stack);
        printf("%d ", current->data);

        if (current->right != NULL) {
            push(&stack, current->right);
        }

        if (current->left != NULL) {
            push(&stack, current->left);
        }
    }
}

// Non-Recursive Postorder Traversal
void postorderNonRecursive(struct Node* root) {
    if (root == NULL) return;

    struct StackNode* stack1 = NULL;
    struct StackNode* stack2 = NULL;

    push(&stack1, root);

    while (!isEmpty(stack1)) {
        struct Node* current = pop(&stack1);
        push(&stack2, current);

        if (current->left != NULL) {
            push(&stack1, current->left);
        }

        if (current->right != NULL) {
            push(&stack1, current->right);
        }
```

```
    }

    while (!isEmpty(stack2)) {
        struct Node* current = pop(&stack2);
        printf("%d ", current->data);
    }
}
```

// Driver program to test non-recursive traversals

```
int main() {
    // Constructing a sample binary tree
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);

    printf("Non-Recursive Inorder Traversal: ");
    inorderNonRecursive(root);
    printf("\n");

    printf("Non-Recursive Preorder Traversal: ");
    preorderNonRecursive(root);
    printf("\n");

    printf("Non-Recursive Postorder Traversal: ");
    postorderNonRecursive(root);
    printf("\n");

    return 0;
}
```

Output:

```
Non-Recursive Inorder Traversal: 4 2 5 1 3
Non-Recursive Preorder Traversal: 1 2 4 5 3
Non-Recursive Postorder Traversal: 4 5 2 3 1

--------------------------------
Process exited after 0.2789 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT- 8

Write a program to implement
i) Binary Search tree ii) B Trees iii) B+ Trees iv) AVL trees v) Red - Black trees

i) Binary Search tree

```
#include <stdio.h>
#include <stdlib.h>
// Define the structure of a tree node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Function to insert a node in the BST
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}

// Function to find the minimum value node in a BST
struct Node* findMinNode(struct Node* node) {
    struct Node* current = node;
    while (current && current->left != NULL) {
        current = current->left;
    }
    return current;
}

// Function to delete a node in the BST
struct Node* deleteNode(struct Node* root, int key) {
```

```
    if (root == NULL) {
        return root;
    }
    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->data) {
        root->right = deleteNode(root->right, key);
    } else {
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }
        struct Node* temp = findMinNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

// Function to search for a node in the BST
struct Node* search(struct Node* root, int key) {
    if (root == NULL || root->data == key) {
        return root;
    }
    if (key < root->data) {
        return search(root->left, key);
    }
    return search(root->right, key);
}

int main() {
    struct Node* root = NULL;
    int choice, data, key;

    while (1) {
        printf("\nBinary Search Tree Menu:\n");
        printf("1. Insert a node\n");
        printf("2. Delete a node\n");
        printf("3. Search for a node\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert: ");
```

```
            scanf("%d", &data);
            root = insert(root, data);
            break;
        case 2:
            printf("Enter data to delete: ");
            scanf("%d", &data);
            root = deleteNode(root, data);
            break;
        case 3:
            printf("Enter key to search: ");
            scanf("%d", &key);
            if (search(root, key)) {
                printf("%d found in the BST.\n", key);
            } else {
                printf("%d not found in the BST.\n", key);
            }
            break;
        case 4:
            // Clean up and exit
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
    }
  }

  return 0;
}
```

Output:

Binary Search Tree Menu:
1. Insert a node
2. Delete a node
3. Search for a node
4. Exit

Enter your choice: 1 Enter
data to insert: 33

Binary Search Tree Menu:
1. Insert a node
2. Delete a node
3. Search for a node
4. Exit

Enter your choice: 1 Enter
data to insert: 23

Binary Search Tree Menu:
1. Insert a node
2. Delete a node
3. Search for a node
4. Exit

```
Enter your choice: 3
Enter key to search: 23
23 found in the BST.

Binary Search Tree Menu:
  1. Insert a node
  2. Delete a node
  3. Search for a node
  4. Exit
Enter your choice: 2 Enter
data to delete: 23

Binary Search Tree Menu:
  1. Insert a node
  2. Delete a node
  3. Search for a node
  4. Exit
Enter your choice: 3
Enter key to search: 23 23
not found in the BST.
```

ii) B Trees

```
#include <stdio.h>
#include <stdlib.h>

#define M 4  // Order of the B-tree

struct Node {
    int keys[M - 1];
    struct Node* children[M];
    int isLeaf;
    int numKeys;
 };

typedef struct Node Node;

Node* createNode() {
        int i;
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    newNode->isLeaf = 1;
    newNode->numKeys = 0;

    for ( i = 0; i < M; i++) {
        newNode->children[i] = NULL;
        if (i < M - 1) {
```

```
            newNode->keys[i] = 0;
        }
    }

    return newNode;
}

Node* root = NULL;
// Insert a key into the B-tree
void insert(int key) {
    if (root == NULL) {
        root = createNode();
        root->keys[0] = key;
        root->numKeys++;
    } else {
        Node* currentNode = root;
        while (!currentNode->isLeaf) {
            int i = 0;
            while (i < currentNode->numKeys && key >= currentNode->keys[i]) {
                i++;
            }
            currentNode = currentNode->children[i];
        }

        // Insert the key into the leaf node
        int i = currentNode->numKeys - 1;
        while (i >= 0 && key < currentNode->keys[i]) {
            currentNode->keys[i + 1] = currentNode->keys[i];
            i--;
        }
        currentNode->keys[i + 1] = key;
        currentNode->numKeys++;

        // Split the node if it's full
        if (currentNode->numKeys == M - 1) {
            // TODO: Implement node splitting
        }
    }
}

// In-order Traversal
void inOrderTraversal(Node* currentNode) {
            int i;
    if (currentNode != NULL) {
        for (i = 0; i < currentNode->numKeys; i++) {
            inOrderTraversal(currentNode->children[i]);
            printf("%d ", currentNode->keys[i]);
        }
        inOrderTraversal(currentNode->children[currentNode->numKeys]);
    }
}
```

```
int main() {
    int choice, key;

    while (1) {
        printf("\n1. Insert\n2. In-order Traversal\n3. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the key to insert: ");
                scanf("%d", &key);
                insert(key);
                break;
            case 2:
                printf("In-order Traversal: ");
                inOrderTraversal(root);
                printf("\n");
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Output:

```
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1 Enter
the key to insert: 45

1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1 Enter
the key to insert: 22

1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1 Enter
the key to insert: 78

1. Insert
```

```
  2. In-order Traversal
  3. Exit
Enter your choice: 2
In-order Traversal: 22 45 78
  1. Insert
  2. In-order Traversal
  3. Exit
Enter your choice: 3

 ------------------------------------
Process exited after 22.46 seconds with return value 0
Press any key to continue . . .
```

iii) B+ Trees

```
#include <stdio.h>
#include <stdlib.h>

#define M 5  // Order of the B+ tree

struct Node {
    int keys[M - 1];
    struct Node* children[M];
    int isLeaf;
    int numKeys;
    struct Node* next;
};

typedef struct Node Node;

Node* createNode() {
        int i;
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    newNode->isLeaf = 1;
    newNode->numKeys = 0;
    newNode->next = NULL;

    for (i = 0; i < M; i++) {
        newNode->children[i] = NULL;
        if (i < M - 1) {
            newNode->keys[i] = 0;
        }
    }

    return newNode;
```

```
 }

Node* root = NULL;
// Insert a key into the B+ tree
void insert(int key) {
    if (root == NULL) {
        root = createNode();
        root->keys[0] = key;
        root->numKeys++;
    } else {
        Node* currentNode = root;
        while (!currentNode->isLeaf) {
            int i = 0;
            while (i < currentNode->numKeys && key >= currentNode->keys[i]) {
                i++;
            }
            currentNode = currentNode->children[i];
        }

        // Insert the key into the leaf node
        int i = currentNode->numKeys - 1;
        while (i >= 0 && key < currentNode->keys[i]) {
            currentNode->keys[i + 1] = currentNode->keys[i];
            i--;
        }
        currentNode->keys[i + 1] = key;
        currentNode->numKeys++;

        // Split the node if it's full
        if (currentNode->numKeys == M - 1) {
            // TODO: Implement node splitting
        }
    }
 }

// In-order Traversal
 void inOrderTraversal(Node* currentNode) {
            int i;
    if (currentNode != NULL) {
        for (i = 0; i < currentNode->numKeys; i++) {
            inOrderTraversal(currentNode->children[i]);
            printf("%d ", currentNode->keys[i]);
        }
        inOrderTraversal(currentNode->children[currentNode->numKeys]);
    }
 }

int main() {
    int choice, key;

    while (1) {
```

```
        printf("\n1. Insert\n2. In-order Traversal\n3. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the key to insert: ");
                scanf("%d", &key);
                insert(key);
                break;
            case 2:
                printf("In-order Traversal: ");
                inOrderTraversal(root);
                printf("\n");
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Output:

1. Insert
2. In-order Traversal
3. Exit

Enter your choice: 1 Enter
the key to insert: 45

1. Insert
2. In-order Traversal
3. Exit

Enter your choice: 1 Enter
the key to insert: 22

1. Insert
2. In-order Traversal
3. Exit

Enter your choice: 1 Enter
the key to insert: 56

1. Insert
2. In-order Traversal
3. Exit

Enter your choice: 2
In-order Traversal: 22 45 56

```
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 3

--------------------------------
Process exited after 16.26 seconds with return value 0
Press any key to continue . . .
```

iv) AVL trees

```
#include <stdio.h>
#include <stdlib.h>

// Definition of an AVL tree node
struct AVLNode {
    int data;
    struct AVLNode *left;
    struct AVLNode *right;
    int height;
};

// Function to get the height of a node
int height(struct AVLNode *node) {
    if (node == NULL) return 0;
    return node->height;
}

// Function to get the maximum of two integers int
max(int a, int b) {
    return (a > b) ? a : b;
}

// Function to create a new AVL tree node
struct AVLNode* createNode(int data) {
    struct AVLNode* newNode = (struct AVLNode*)malloc(sizeof(struct AVLNode));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    newNode->height = 1; // New node is initially at height 1
    return newNode;
}

// Function to rotate the subtree rooted with y to the left
struct AVLNode* leftRotate(struct AVLNode *y) {
    struct AVLNode *x = y->right;
    struct AVLNode *T2 = x->left;

    // Perform rotation
    x->left = y;
```

```
    y->right = T2;

    // Update heights
    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;

    // Return new root
    return x;
 }

// Function to rotate the subtree rooted with x to the right
struct AVLNode* rightRotate(struct AVLNode *x) {
    struct AVLNode *y = x->left;
    struct AVLNode *T2 = y->right;

    // Perform rotation
    y->right = x;
    x->left = T2;

    // Update heights
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;

    // Return new root
    return y;
 }

// Function to get the balance factor of a node
int getBalance(struct AVLNode *node) {
    if (node == NULL) return 0;
    return height(node->left) - height(node->right);
 }

// Function to insert a node into the AVL tree
struct AVLNode* insert(struct AVLNode *node, int data) {
    // Perform the standard BST insert
    if (node == NULL) return createNode(data);

    if (data < node->data)
        node->left = insert(node->left, data);
    else if (data > node->data)
        node->right = insert(node->right, data);
    else // Duplicate keys are not allowed in AVL
        return node;

    // Update height of the current node
    node->height = 1 + max(height(node->left), height(node->right));

    // Get the balance factor to check whether this node became unbalanced
    int balance = getBalance(node);
```

```
    // Left Left Case
    if (balance > 1 && data < node->left->data)
        return rightRotate(node);

    // Right Right Case
    if (balance < -1 && data > node->right->data)
        return leftRotate(node);

    // Left Right Case
    if (balance > 1 && data > node->left->data) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    // Right Left Case
    if (balance < -1 && data < node->right->data) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    // No balancing needed, return the unchanged node pointer
    return node;
}

// Function to perform in-order traversal of the AVL tree
void inorderTraversal(struct AVLNode* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Driver program to test AVL tree
int main() {
    struct AVLNode *root = NULL;

    // Example nodes to insert into the AVL tree
    int keys[] = {9, 5, 10, 0, 6, 11, -1, 1, 2};
    int i;

    for (i = 0; i < sizeof(keys) / sizeof(keys[0]); i++) {
        root = insert(root, keys[i]);
    }

    printf("In-order traversal of the AVL tree: ");
    inorderTraversal(root);
    printf("\n");
```

```
    return 0;
}
```

Output:

```
In-order traversal of the AVL tree: -1 0 1 2 5 6 9 10 11

--------------------------------
Process exited after 0.02169 seconds with return value 0 Press
any key to continue . . .
```

v) Red - Black trees

```
#include <stdio.h>
#include <stdlib.h>

enum Color { RED, BLACK };

struct Node {
    int data;
    enum Color color;
    struct Node* parent;
    struct Node* left;
    struct Node* right;
};

typedef struct Node Node;

Node* createNode(int data, Node* parent) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

    newNode->data = data;
    newNode->color = RED;
    newNode->parent = parent;
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}

void rotateLeft(Node** root, Node* x) {
    Node* y = x->right;
    x->right = y->left;
    if (y->left != NULL)
        y->left->parent = x;
```

```
    y->parent = x->parent;
    if (x->parent == NULL)
        *root = y;
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;
    y->left = x;
    x->parent = y;
}

void rotateRight(Node** root, Node* y) {
    Node* x = y->left;
    y->left = x->right;
    if (x->right != NULL)
        x->right->parent = y;
    x->parent = y->parent;
    if (y->parent == NULL)
        *root = x;
    else if (y == y->parent->left)
        y->parent->left = x;
    else
        y->parent->right = x;
    x->right = y;
    y->parent = x;
}

void fixInsertion(Node** root, Node* z) {
    while (z != *root && z->parent->color == RED) {
        if (z->parent == z->parent->parent->left) {
            Node* y = z->parent->parent->right;
            if (y != NULL && y->color == RED) {
                z->parent->color = BLACK;
                y->color = BLACK;
                z->parent->parent->color = RED;
                z = z->parent->parent;
            } else {
                if (z == z->parent->right) {
                    z = z->parent;
                    rotateLeft(root, z);
                }
                z->parent->color = BLACK;
                z->parent->parent->color = RED;
                rotateRight(root, z->parent->parent);
            }
        } else {
            Node* y = z->parent->parent->left;
            if (y != NULL && y->color == RED) {
                z->parent->color = BLACK;
                y->color = BLACK;
```

```
            z->parent->parent->color = RED;
            z = z->parent->parent;
        } else {
            if (z == z->parent->left) {
                z = z->parent;
                rotateRight(root, z);
            }
            z->parent->color = BLACK;
            z->parent->parent->color = RED;
            rotateLeft(root, z->parent->parent);
        }
    }
  }
  (*root)->color = BLACK;
}

void insert(Node** root, int data) {
  Node* z = createNode(data, NULL);
  Node* y = NULL;
  Node* x = *root;

  while (x != NULL) {
    y = x;
    if (z->data < x->data)
      x = x->left;
    else
      x = x->right;
  }

  z->parent = y;
  if (y == NULL)
    *root = z;
  else if (z->data < y->data)
    y->left = z;
  else
    y->right = z;

  fixInsertion(root, z);
}

void inOrderTraversal(Node* root) {
  if (root != NULL) {
    inOrderTraversal(root->left);
    printf("%d ", root->data,root->color);
    inOrderTraversal(root->right);
  }
}

int main() {
  Node* root = NULL;
```

```
    int data, choice;

    while (1) {
        printf("\n1. Insert\n2. In-order Traversal\n3. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the data to insert: ");
                scanf("%d",  &data);
                insert(&root, data);
                break;
            case 2:
                printf("In-order Traversal: ");
                inOrderTraversal(root);
                printf("\n");
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Output:

```
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1
Enter the data to insert: 45

1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1
Enter the data to insert: 23

1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1
Enter the data to insert: 78

1. Insert
2. In-order Traversal
3. Exit
```

```
Enter your choice: 2
In-order Traversal: 23  45  78

                                                    1. Insert
                                                2. In-order Traversal
                                                    3. Exit
Enter your choice: 3

 --------------------------------
Process exited after 14.52 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT- 9

Write a program to implement the graph traversal methods.

```
 #include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_VERTICES 100

// Structure for a graph node
struct Node {
    int data;
    struct Node* next;
};

// Structure for the adjacency list
struct Graph {
    int numVertices;
    struct Node* adjLists[MAX_VERTICES];
    bool visited[MAX_VERTICES];
};

// Create a new graph
struct Graph* createGraph(int vertices) {
        int i;
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    for (i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = false;
    }
    return graph;
}
```

```
// Add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = dest;
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Add edge from dest to src (for an undirected graph)
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = src;
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
 }

// Depth-First Search (DFS)
 void DFS(struct Graph* graph, int vertex) {
    if (!graph->visited[vertex]) {
        printf("Visited %d\n", vertex);
        graph->visited[vertex] = true;
        struct Node* adjList = graph->adjLists[vertex];
        while (adjList) {
            int connectedVertex = adjList->data;
            if (!graph->visited[connectedVertex]) {
                DFS(graph, connectedVertex);
            }
            adjList = adjList->next;
        }
    }
 }

// Breadth-First Search (BFS)
 void BFS(struct Graph* graph, int startVertex) {
    int queue[MAX_VERTICES];
    int front = 0, rear = 0;
    queue[rear++] = startVertex;
    graph->visited[startVertex] = true;
    printf("Visited %d\n", startVertex);

    while (front < rear) {
        int currentVertex = queue[front++];
        struct Node* adjList = graph->adjLists[currentVertex];
        while (adjList) {
            int connectedVertex = adjList->data;
            if (!graph->visited[connectedVertex]) {
                queue[rear++] = connectedVertex;
                graph->visited[connectedVertex] = true;
                printf("Visited %d\n", connectedVertex);
            }
            adjList = adjList->next;
        }
```

```
    }
}
int main() {
        int i;
    struct Graph* graph = createGraph(6);

    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 4);
    addEdge(graph, 3, 4);
    addEdge(graph, 3, 5);
    addEdge(graph, 4, 5);

    printf("Depth-First Search (DFS):\n");
    DFS(graph, 0);

    // Reset visited flags
    for (i = 0; i < graph->numVertices; i++) {
        graph->visited[i] = false;
    }

    printf("\nBreadth-First Search (BFS):\n");
    BFS(graph, 0);

    return 0;
}
```

Output:

```
Depth-First Search (DFS):
Visited 0
Visited 2
Visited 4
Visited 5
Visited 3
Visited 1

Breadth-First Search (BFS):
Visited 0
Visited 2
Visited 1
Visited 4
Visited 3
Visited 5

--------------------------------
Process exited after 0.04696 seconds with return value 0 Press
any key to continue . . .
```

EXPERIMENT- 10

Implement a Pattern matching algorithms using Boyer- Moore, Knuth-Morris-Pratt

i) Boyer- Moore

```
/* C Program for Bad Character Heuristic of Boyer Moore String Matching Algorithm */
# include <limits.h>
# include <string.h>
# include <stdio.h>

# define NO_OF_CHARS 256

// A utility function to get maximum of two integers

int max (int a, int b) { return (a > b)? a: b; }

// The preprocessing function for Boyer Moore's bad character heuristic

void badCharHeuristic( char *str, int size, int badchar[NO_OF_CHARS])
 {
    int i;

    // Initialize all occurrences as -1
    for (i = 0; i < NO_OF_CHARS; i++)
        badchar[i] = -1;

    // Fill the actual value of last occurrence of a character
    for (i = 0; i < size; i++)
        badchar[(int) str[i]] = i;
 }

// A pattern searching function that uses Bad Character Heuristic
void search( char *txt,  char *pat)
 {
    int m = strlen(pat);
    int n = strlen(txt);

    int badchar[NO_OF_CHARS];

    /* Fill the bad character array by calling the preprocessing function badCharHeuristic()
       for given pattern */
    badCharHeuristic(pat, m, badchar);

    int s = 0;  // s is shift of the pattern with respect to text
    while(s <= (n - m))
    {
       int j = m-1;

       /* Keep reducing index j of pattern while characters of pattern and text are
          matching at this shift s */
```

```
        while(j >= 0 && pat[j] == txt[s+j])
            j--;
        /* If the pattern is present at current shift, then index j will become -1 after
           the above loop */
        if (j < 0)
        {
            printf("\n pattern occurs at shift = %d", s);

            /* Shift the pattern so that the next character in text aligns with the last
               occurrence of it in pattern.The condition s+m < n is necessary for
               the case when pattern occurs at the end of text */

             s += (s+m < n)? m-badchar[txt[s+m]] : 1;

        }

        else
            /* Shift the pattern so that the bad character  in text aligns with the last occurrence of
               it in pattern. The max function is used to  make sure that we get a positive shift.
               We may get a negative shift if the last occurrence of bad character in pattern
               is on the right side of the current character. */

             s += max(1, j - badchar[txt[s+j]]);
    }
 }

/* Driver program to test above function */
int main()
 {
    char txt[] = "ABAAABCD";
    char pat[] = "ABC";
    search(txt, pat);
    return 0;
 }
```

Output:

```
  pattern occurs at shift = 4
 ------------------------------------
Process exited after 0.07759 seconds with return value 0 Press
any key to continue . . .
```

ii) Knuth-Morris-Pratt

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

void computeLPSArray(char *pat, int M, int *lps);

void KMPSearch(char *pat, char *txt) {
```

```
    int M = strlen(pat);
    int N = strlen(txt);

    // create lps[] that will hold the longest prefix suffix values for pattern
    int *lps = (int *) malloc(sizeof(int) * M);
    int j = 0; // index for pat[]

    // Preprocess the pattern (calculate lps[] array)
    computeLPSArray(pat, M, lps);

    int i = 0; // index for txt[]
    while (i < N) {
      if (pat[j] == txt[i]) {
        j++;
        i++;
      }

      if (j == M) {
        printf("Found pattern at index %d \n", i - j);
        j = lps[j - 1];
      }

      // mismatch after j matches
      else if (i < N && pat[j] != txt[i]) {
        // Do not match lps[0..lps[j-1]] characters,
        // they will match anyway
        if (j != 0)
          j = lps[j - 1];
        else
          i = i + 1;
      }
    }
    free(lps); // to avoid memory leak
  }

void computeLPSArray(char *pat, int M, int *lps) {
    int len = 0; // lenght of the previous longest prefix suffix
    int i;

    lps[0] = 0; // lps[0] is always 0
    i = 1;

    // the loop calculates lps[i] for i = 1 to M-1
    while (i < M) {
      if (pat[i] == pat[len]) {
        len++;
        lps[i] = len;
        i++;
      } else // (pat[i] != pat[len])
      {
```

```
            if (len != 0) {
                // This is tricky. Consider the example AAACAAAA and i = 7. len = lps[len -
                1];

                // Also, note that we do not increment i here
            } else // if (len == 0)
            {
                lps[i] = 0; i++;
            }
        }
    }
}

// Driver program to test above function int main() {
    char *txt = "ABABDABACDABCABAB";
    char *pat = "BCABAB";
    KMPSearch(pat, txt); return 0;
}
```

Output:

Found pattern at index 11

Process exited after 0.1387 seconds with return value 0 Press any key to continue . . .

OPEN ENDED EXPERIMENTS

INVERSION COUNT IN ARRAY USING MERGE SORT

Inversion Count for an array indicates – how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is 0, but if the array is sorted in reverse order, the inversion count is the maximum.
Given an array **arr[]**. The task is to find the inversion count of **arr[]**. Where two elements arr[i] and arr[j] form an inversion if a[i] > a[j] and i < j.
Examples:
Input: *arr[] = {8, 4, 2, 1}*
Output: *6*
Explanation: *Given array has six inversions: (8, 4), (4, 2), (8, 2), (8, 1), (4, 1), (2, 1).*
Input: ***arr[] = {1, 20, 6, 4, 5}***
Output: ***5***
Explanation: ***Given array has five inversions: (20, 6), (20, 4), (20, 5), (6, 4), (6, 5).***
Naive Approach:
Traverse through the array, and for every index, find the number of smaller elements on its right side of the array. This can be done using a nested loop. Sum up the counts for all indices in the array and print the sum.
Follow the below steps to Implement the idea:

- Traverse through the array from start to end
- For every element, find the count of elements smaller than the current number up to that index using another loop.
- Sum up the count of inversion for every index.
- Print the count of inversions.

Below is the Implementation of the above approach:

```
// C++ program to Count Inversions

// in an array

#include <bits/stdc++.h>

using namespace std;

int getInvCount(int arr[], int n)

{

        int inv_count = 0;

        for (int i = 0; i < n - 1; i++)

                for (int j = i + 1; j < n; j++)
```

```
                    if (arr[i] > arr[j])
.
                        inv_count++;
        return inv_count;
}
// Driver Code
int main()
{
        int arr[] = { 1, 20, 6, 4, 5 };
        int n = sizeof(arr) / sizeof(arr[0]);
        cout << " Number of inversions are "
             << getInvCount(arr, n);
        return 0;
}
```

Output

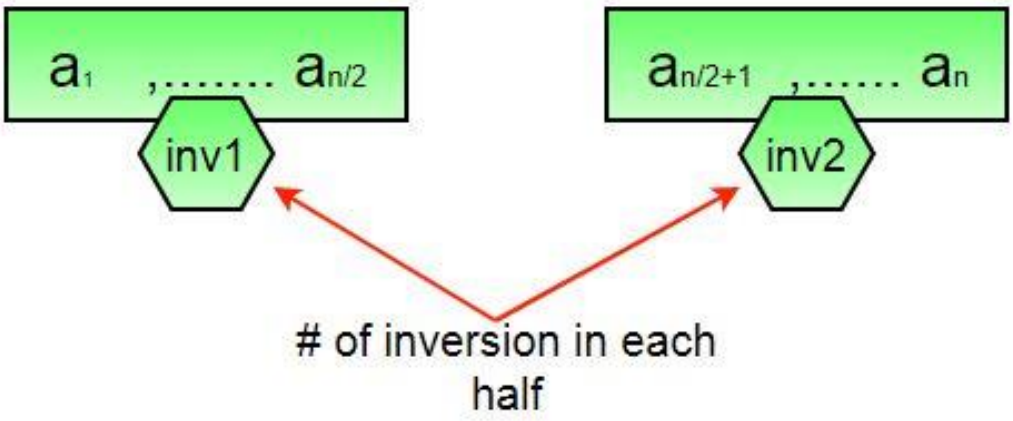
```
Number of inversions are 5
```

Time Complexity: O(N2), Two nested loops are needed to traverse the array from start to end.
Auxiliary Space: O(1), No extra space is required.

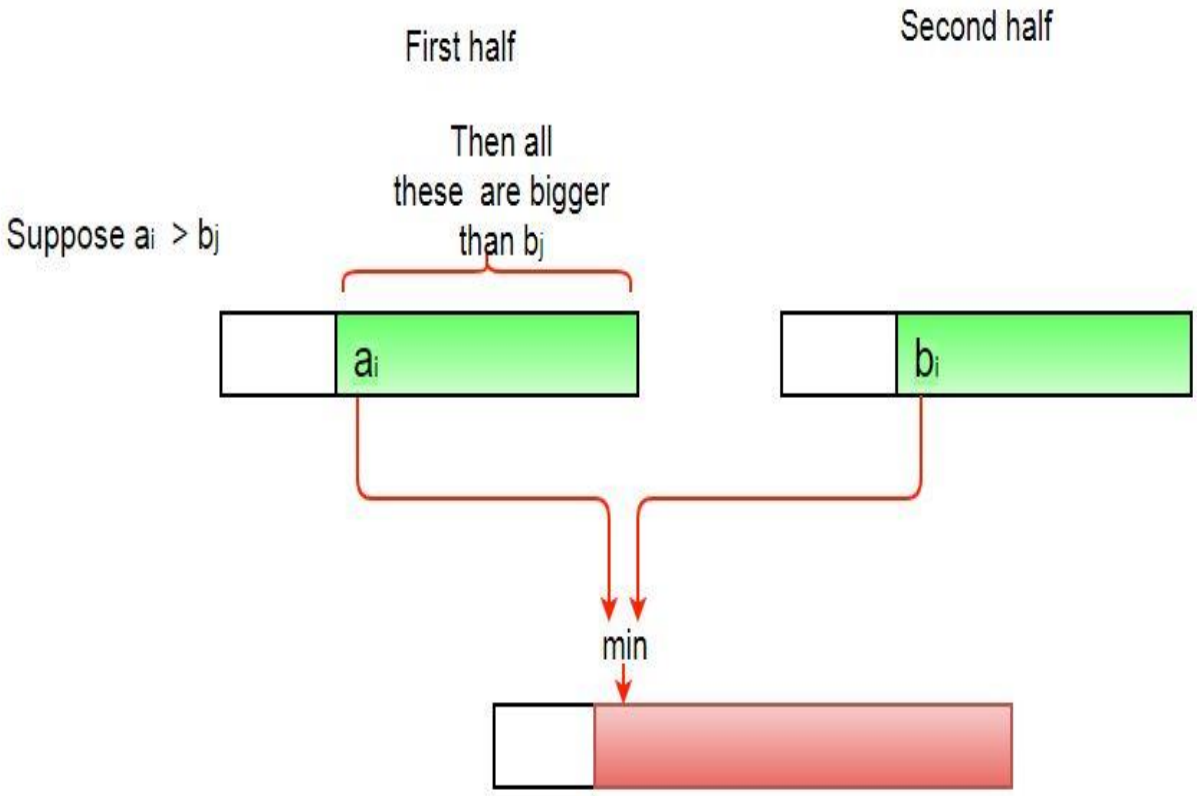
Below is the idea to solve the problem:

*Use **Merge sort** with modification that every time an unsorted pair is found increment **count** by one and return count at the end.*

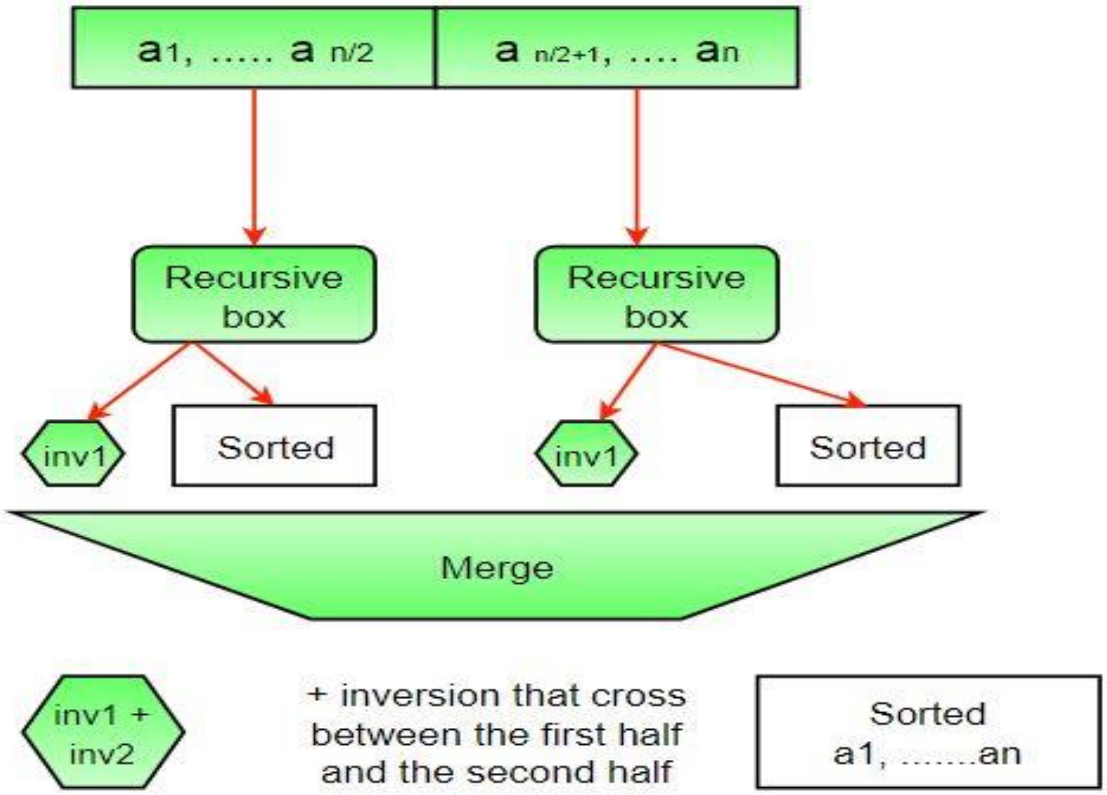
Illustration:
Suppose the number of inversions in the left half and right half of the array (let be inv1 and inv2); what kinds of inversions are not accounted for in Inv1 + Inv2? The answer is – the inversions that need to be counted during the merge step. Therefore, to get the total number of inversions that needs to be added are the number of inversions in the left subarray, right subarray, and merge().



How to get** the **number of inversions in merge()?
In merge process, let i is used for indexing left sub-array and j for right sub-array. At any step in merge(), if a[i] is greater than a[j], then there are (mid – i) inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray (a[i+1], a[i+2] … a[mid]) will be greater than a[j]



The complete picture:



Follow the below steps to Implement the idea:

- The idea is similar to merge sort, divide the array into two equal or almost equal halves in each step until the base case is reached.
- Create a function merge that counts the number of inversions when two halves of the array are merged,
    - Create two indices i and j, i is the index for the first half, and j is an index of the second half.
    - If a[i] is greater than a[j], then there are (mid – i) inversions because left and right subarrays are sorted, so all the remaining elements in left-subarray (a[i+1], a[i+2] … a[mid]) will be greater than a[j].
    - Create a recursive function to divide the array into halves and find the answer by summing the number of inversions in the first half, the number of inversions in the second half and the number of inversions by merging the two.
- The base case of recursion is when there is only one element in the given half.
- Print the answer.

VIVA 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) Differentiate between file and structure storage structure.

The key difference between both the data structure is the memory area that is being accessed. When dealing with the structure that resides the main memory of the computer system, this is referred to as storage structure. When dealing with an auxiliary structure, we refer to it as file structures.

3) When is a binary search best applied?

A binary search is an algorithm that is best applied to search a list when the elements are already in order or sorted. The list is searched starting in the middle, such that if that middle value is not the target search key, it will check to see if it will continue the search on the lower half of the list or the higher half. The split and search will then continue in the same manner.

4) What is a linked 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.

5) How do you reference all the elements in a one-dimension array?

To reference all the elements in a one -dimension array, you need to use an indexed loop, So that, the counter runs from 0 to the array size minus one. In this manner, You can reference all the elements in sequence by using the loop counter as the array subscript.

6) In what areas do data structures are applied?

Data structures are essential in almost every aspect where data is involved. In general, algorithms that involve efficient data structure is applied in the following areas: numerical analysis, operating system, A.I., compiler design, database management, graphics, and statistical analysis, to name a few.

7) What is LIFO?

LIFO is a short form of Last In First Out. It refers how data is accessed, stored and retrieved. Using this scheme, data that was stored last should be the one to be extracted first. This also means that in order to gain access to the first data, all the other data that was stored before this first data must first be retrieved and extracted.

8) What is a 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.

9) What are binary trees?

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.

10) Which data structures are applied when dealing with a recursive function?

Recursion, is a function that calls itself based on a terminating condition, makes use of the stack. Using LIFO, a call to a recursive function saves the return address so that it knows how to return to the calling function after the call terminates.

11) What is a 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.

12) Explain Binary Search Tree

A binary search tree stores data in such a way that they can be retrieved very efficiently. The left subtree contains nodes whose keys are less than the node's key value, while the right subtree contains nodes whose keys are greater than or equal to the node's key value. Moreover, both subtrees are also binary search trees.

13) What are multidimensional arrays?

Multidimensional arrays make use of multiple indexes to store data. It is useful when storing data that cannot be represented using single dimensional indexing, such as data representation in a board game, tables with data stored in more than one column.

14) Are linked lists considered linear or non-linear data structures?

It depends on where you intend to apply linked lists. If you based it on storage, a linked list is considered non-linear. On the other hand, if you based it on access strategies, then a linked list is considered linear.

15) How does dynamic memory allocation help in managing data?

Apart from being able to store simple structured data types, dynamic memory allocation can combine separately allocated structured blocks to form composite structures that expand and contract as needed.

16) What is FIFO?

FIFO stands for First-in, First-out, and is used to represent how data is accessed in a queue. Data has been inserted into the queue list the longest is the one that is removed first.

17) What is an ordered list?

An ordered list is a list in which each node's position in the list is determined by the value of its key component, so that the key values form an increasing sequence, as the list is traversed.

18) 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.

19) Differentiate NULL and VOID

Null is a value, whereas Void is a data type identifier. A variable that is given a Null value indicates an empty value. The void is used to identify pointers as having no initial size.

20) What is the primary advantage of a linked list?

A linked list is an ideal data structure because it can be modified easily. This means that editing a linked list works regardless of how many elements are in the list.

21) What is the difference between a PUSH and a 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.

22) 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.

23) How does variable declaration affect memory allocation?

The amount of memory to be allocated or reserved would depend on the data type of the variable being declared. For example, if a variable is declared to be of integer type, then 32 bits of memory storage will be reserved for that variable.

24) What is the advantage of the heap over a stack?

The heap is more flexible than the stack. That's because memory space for the heap can be dynamically allocated and de-allocated as needed. However, the memory of the heap can at times be slower when compared to that stack.

25) What is a postfix expression?

A postfix expression is an expression in which each operator follows its operands. The advantage of this form is that there is no need to group sub-expressions in parentheses or to consider operator precedence.

26) What is Data abstraction?

Data abstraction is a powerful tool for breaking down complex data problems into manageable chunks. This is applied by initially specifying the data objects involved and the operations to be performed on these data objects without being overly concerned with how the data objects will be represented and stored in memory.

27) How do you insert a new item in a binary search tree?

Assuming that the data to be inserted is a unique value (that is, not an existing entry in the tree), check first if the tree is empty. If it's empty, just insert the new item in the root node. If it's not empty, refer to the new item's key. If it's smaller than the root's key, insert it into the root's left subtree, otherwise, insert it into the root's right subtree.

28) How does a selection sort work for an array?

The selection sort is a fairly intuitive sorting algorithm, though not necessarily efficient. In this process, the smallest element is first located and switched with the element at subscript zero, thereby placing the smallest element in the first position. The smallest element remaining in the subarray is then located next to subscripts 1 through n-1 and switched with the element at subscript 1, thereby placing the second smallest element in the second position. The steps are repeated in the same manner till the last element.

29) How do signed and unsigned numbers affect memory?

In the case of signed numbers, the first bit is used to indicate whether positive or negative, which leaves you with one bit short. With unsigned numbers, you have all bits available for that number. The effect is best seen in the number range (an unsigned 8-bit number has a range 0-255, while the 8-bit signed number has a range -128 to +127.

30) What is the minimum number of nodes that a binary tree can have?

A binary tree can have a minimum of zero nodes, which occurs when the nodes have NULL values. Furthermore, a binary tree can also have 1 or 2 nodes.