Practical 7

Write a menu driven program to implement following operations on the singly linked list.

(a) Insert a node at the front of the linked list.

(b) Insert a node at the end of the linked list.

(c) Insert a node such that linked list is in ascending order.

(d) Delete a First node of the linked list.

(e) Delete a node before specified position.

(f) Delete a node after specified position.

(a) Insert a node at the front of the linked list.

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

struct node
{
        int data;
        struct node *next;
} *head;
void initialize()
{
        head = NULL;
}
void insertAtFront(int num)
{

        struct node* newNode = (struct node*) malloc(sizeof(struct
        node));  newNode->data = num;
        newNode->next = head;
        head = newNode;
        printf("Inserted Element : %d\n",
        num);
}
void printLinkedList(struct node *nodePtr)
 {
        printf("\nLinked List\n");
        while (nodePtr != NULL)
        {
                printf("%d", nodePtr->data);
```

```
                nodePtr = nodePtr->next;
                if(nodePtr != NULL)
                printf("-->");
        }
}

int main()
{
        initialize();
        insertAtFront(2);
        insertAtFront(4);
        insertAtFront(5);
        insertAtFront(9);
        printLinkedList(head);

 return 0;
}
```

OUTPUT

```
Inserted Element : 2
Inserted Element : 4
Inserted Element : 5
Inserted Element : 9

Linked List
9-->5-->4-->2
```

(b) Insert a node at the end of the linked list.

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

struct node
{
        int data;
        struct node *next;
} *head;
void initialize()
{
       head = NULL;
}
void insertAtFront(int num)
 {

        struct node* newNode = (struct node*) malloc(sizeof(struct
       node));  newNode->data = num;
        newNode->next = head;
        head = newNode;

}
void insertAtEnd(struct node* head, int num)
{

        if (head == NULL)
       {
                printf("Error : Invalid node pointer !!!\n");
                return;
       }

        struct node* newNode =(struct node*) malloc(sizeof(struct
       node));  newNode->data = num;
        newNode->next = NULL;

        while(head->next != NULL)
        head = head->next;

        head->next = newNode;
}
void printLinkedList(struct node *nodePtr)
 {
        printf("\nLinked List\n");
        while (nodePtr != NULL)
       {
                printf("%d", nodePtr->data);
                nodePtr = nodePtr->next;
                if(nodePtr != NULL)
                printf("-->");
```

```
        }
}

int main()
{
        initialize();
        insertAtFront(2);
        insertAtEnd(head, 10);
        printf("\n\nAfter Insertion At End\n");
        printLinkedList(head);
        return 0;
}
```

OUTPUT

```
After Insertion At End

Linked List
2-->10
```

(c) Insert a node such that linked list is in ascending order.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
        int data;
        struct Node* next;
};
void sortedInsert(struct Node** head_ref,struct Node* new_node)
{
        struct Node* current;
        if (*head_ref == NULL || (*head_ref)->data>= new_node->data)  {
                new_node->next = *head_ref;
                *head_ref = new_node;
        }
        else {
                current = *head_ref;
                while (current->next != NULL && current->next->data
                <  new_node->data)
                {
                        current = current->next;
                }
                new_node->next = current->next;
                current->next = new_node;
        }
}
struct Node* newNode(int new_data)
{
                struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
        new_node->data = new_data;
        new_node->next = NULL;
        return new_node;
}
void printList(struct Node* head)
{
        struct Node* temp = head;
        while (temp != NULL) {
                printf("%d ", temp->data);
                temp = temp->next;
        }
}
int main()
{
        struct Node* head = NULL;
        struct Node* new_node = newNode(5);
        sortedInsert(&head, new_node);
        new_node = newNode(10);
        sortedInsert(&head, new_node);
        new_node = newNode(7);
```

```
        sortedInsert(&head, new_node);
        new_node = newNode(3);
        sortedInsert(&head, new_node);
        new_node = newNode(1);
        sortedInsert(&head, new_node);
        new_node = newNode(9);
        sortedInsert(&head, new_node);
        printf("\n Created Linked List\n");
        printList(head);
        return 0;
}
```

OUTPUT

```
 Created Linked List
1 3 5 7 9 10
--------------------------------
```

(d) Delete a First node of the linked list.

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

struct node
{
        int num;
        struct node *nextptr;
}*stnode;

void createNodeList(int n);
void FirstNodeDeletion();
void displayList();

int main()
{
        int n,num,pos;
                printf("\n\n Linked List : Delete first node of Singly Linked List  :\n");
                        printf("-------------------------------------------------------------\n");
        printf(" Input the number of nodes : ");
```

```
        scanf("%d", &n);
        createNodeList(n);
        printf("\n Data entered in the list are : \n");
        displayList();
        FirstNodeDeletion();
        printf("\n Data, after deletion of first node : \n");
        displayList();
        return 0;
}
void createNodeList(int n)
{
        struct node *fnNode, *tmp;
        int num, i;
        stnode = (struct node *)malloc(sizeof(struct
        node));
        if(stnode == NULL)
        {
                printf(" Memory can not be
                allocated.");
        }
        else
        {

                printf(" Input data for node 1 : ");
                scanf("%d", &num);
                stnode-> num = num;
                stnode-> nextptr = NULL;
                tmp = stnode;

                for(i=2; i<=n; i++)
                {
                        fnNode = (struct node
                        *)malloc(sizeof(struct node));
                        if(fnNode == NULL)
                        {
                                printf(" Memory can
                                not be
                                allocated.");  break;
                        }
                        else
                        {
                                        printf("
                                        Input data for
                                        node %d : ",
```

```
                                        i);  scanf("
                                        %d",
                                        &num);
                                fnNode->num = num;
                                fnNode->nextptr = NULL;
                                tmp->nextptr = fnNode;
                                tmp = tmp->nextptr;
                        }
                }
        }
}
void FirstNodeDeletion()
{
        struct node *toDelptr;
        if(stnode == NULL)
        {
                printf(" There are no node in the list.");
        }
        else
        {
                toDelptr = stnode;
                stnode = stnode->nextptr;
                printf("\n Data of node 1 which is being deleted is : %d\n",
                toDelptr- >num);
                free(toDelptr);
        }
}


void displayList()
{
        struct node *tmp;
        if(stnode == NULL)
        {
                printf(" No data found in the list.");
        }
        else
        {
                tmp = stnode;
                while(tmp != NULL)
                {
                        printf(" Data = %d\n", tmp->num);
                        tmp = tmp->nextptr;
```

```
                }
            }
        }
```

OUTPUT

```
Linked List : Delete first node of Singly Linked List :
------------------------------------------------------------
Input the number of nodes : 2
Input data for node 1 : 10
Input data for node 2 : 20

Data entered in the list are :
Data = 10
Data = 20

Data of node 1 which is being deleted is :  10

Data, after deletion of first node :
Data = 20
```

(e) Delete a node before specified position.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
        int data;
        struct Node *next;
};


void push(struct Node** head_ref, int new_data)
{
        struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
        new_node->data = new_data;
        new_node->next = (*head_ref);
        (*head_ref) = new_node;
}


void deleteNode(struct Node **head_ref, int position)
{
        if (*head_ref == NULL)
        return;


struct Node* temp = *head_ref;
        position--;

        if (position == 0)
        {
                *head_ref = temp->next;
                free(temp);
                return;
        }


        int i;
        for (i=0; temp!=NULL && i<position-1; i++)
                temp = temp->next;
```

```
        if (temp == NULL || temp->next == NULL)
                return;

        struct Node *next = temp->next->next;
        free(temp->next);
        temp->next = next;
}


void printList(struct Node *node)
{
        while (node != NULL)
        {
                printf(" %d ", node->data);
                node = node->next;
        }
}

int main()
{
        struct Node* head = NULL;

        push(&head, 7);
        push(&head, 1);
        push(&head, 3);
        push(&head, 2);
        push(&head, 8);

        puts("Created Linked List: ");
        printList(head);
        deleteNode(&head, 2);
        puts("\nLinked List after Deletion");
        printList(head);
        return 0;
}
```

OUTPUT

```
Created Linked List:
 8  2  3  1  7
Linked List after Deletion
 8  3  1  7
```

(f) Delete a node after specified position.

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

struct Node
{
        int data;
        struct Node *next;
};


void push(struct Node** head_ref, int new_data)
{
        struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
        new_node->data = new_data;
        new_node->next = (*head_ref);
        (*head_ref) = new_node;
}


void deleteNode(struct Node **head_ref, int position)
{
if (*head_ref == NULL)
        return;


struct Node* temp = *head_ref;
```

```
        position++;

        if (position == 0)
        {
                *head_ref = temp->next;
                free(temp);
                return;
        }


        int i;
for (i=0; temp!=NULL && i<position-1; i++) temp =
        temp->next;

if (temp == NULL || temp->next == NULL) return;

        struct Node *next = temp->next->next;
        free(temp->next);
        temp->next = next;
}



void printList(struct Node *node)
{
        while (node != NULL)
        {
                printf(" %d ", node->data);
                node = node->next;
        }
}
int main()
{
        struct Node* head = NULL;

        push(&head, 7);
        push(&head, 1);
        push(&head, 3);
        push(&head, 2);
        push(&head, 8);

        puts("Created Linked List: ");
        printList(head);
```

```
        deleteNode(&head, 2);
        puts("\nLinked List after Deletion");
        printList(head);
        return 0;
    }
```

OUTPUT

```
Created Linked List:
 8  2  3  1  7
Linked List after Deletion
 8  2  3  7
```

Practical 8

- **Write a program to implement stack using linked list.**

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
        int data;
        struct Node *next;
}*top=NULL;
void push(int x)
{
        struct Node *t;
        t=(struct Node*)malloc(sizeof(struct Node));
        if(t==NULL)
        printf("stack is full\n");
        else
        {
                t->data=x;
                t->next=top;
                top=t;
                printf("%d is inserted\n",t->data);
        }
}
int pop()
{
        struct Node *t;
        int x=-1;
        if(top==NULL)
        printf("Stack is Empty\n");
        else
        {
                t=top;
                top=top->next;
                x=t->data;
                printf("%d is
                Deleted\n",t->data);
                free(t);
        }
return x;
}
void Display()
{
      printf("Elemnts\n");
      struct Node *p;
```

```
        p=top;
        while(p!=NULL)
        {
                printf("%d ",p->data);
                p=p->next;
        }
        printf("\n");
}
int main()
{
        push(10);
        push(20);
        push(30);
        Display();
        pop();
        return 0;
}
```

OUTPUT

```
10 is inserted
20 is inserted
30 is inserted
Elemnts
30 20 10
30 is Deleted
```

Practical 9

- **Write a program to implement queue using linked list.**

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
        int data;
        struct Node *next;
}*front=NULL,*rear=NULL;
void enqueue(int x)
{
        struct Node *t;
        t=(struct Node*)malloc(sizeof(struct Node));
        if(t==NULL)
                printf("Queue is FUll\n");
        else
        {
                t->data=x;
                printf("%d is inserted\n",t->data);
                t->next=NULL;
                if(front==NULL)
                front=rear=t;
                else
                {
                        rear->next=t;
                        rear=t;
                }
        }
}
int dequeue()
{
      int x=-1;
      struct Node* t;
      if(front==NULL)
       printf("Queue is Empty\n");
      else
      {
                x=front->data;
                t=front;
                front=front->next;
                printf("%d is Deleted\n",t-
                >data); free(t);
      }
      return x;
```

```
}
void Display()
{
        struct Node *p=front;
        printf("Elements\n");
        while(p)
        {
                printf("%d ",p->data);
                p=p->next;
        }
        printf("\n");
}
int main()
{
        enqueue(10);
        enqueue(20);
        enqueue(30);
        enqueue(40);
        enqueue(50);
        Display();
        dequeue();
        Display();
        return 0;
}
```

OUTPUT

```
10 is inserted
20 is inserted
30 is inserted
40 is inserted
50 is inserted
Elements
10 20 30 40 50
10 is Deleted
Elements
20 30 40 50
```

Practical 10

- **Write a program to implement following operations on the doubly linked list.**
  (a) Insert a node at the front of the linked list.
  (b) Insert a node at the end of the linked list.
  (c) Delete a last node of the linked list.
  (d) Delete a node before specified position.

```
#include<stdio.h>

#include<stdlib.h>

struct node
{
  int info;
  struct node * lptr, * rptr;
}* l, * r;
void doubins(int);
void doubdel(int);
void display();
int chk;
void main()
{
  int ch, x;
  l = NULL;
  r = NULL;
  do {
    printf("\n Press:=>\n");
    printf("\n 1.Insert Node");
    printf("\n 2.Delete Node");
    printf("\n 3.Display Doubly Linked List");
    printf("\n 4.Exit");
    printf("\n Enter Choice: ");
    scanf("%d", & ch);
    switch (ch)
 {
    case 1:
      printf("\n\t Enter Element: ");
      scanf("%d", & x);
      doubins(x);
      display();
      break;
```

```
    case 2:
      printf("\n\t Enter Element which you want to Delete: ");
      scanf("%d", & x);
      chk = 0;
      if (l != NULL) //if(r!=NULL)
      {
        printf("\n\n Before Deletion:=>");
        printf("\n----------------");
        display();
      }
      doubdel(x);
      if (chk == 0) //if(r!=NULL)
      {
        printf("\n\n\n\n\n After Deletion:=>");
        printf("\n---------------");
        display();
      }
      break;
    case 3:
      display();
      break;
    case 4:
      exit(0);
    default:
      printf("\n\t Invalid Choice.\n\tTry Again.");
    }
  } while (ch != 4);
}
void doubins(int x) {
  struct node * New, * temp;
  New = (struct node * ) malloc(sizeof(struct node));
  New -> info = x;
  if (l == NULL) //if(r==NULL)
  {
    New -> lptr = NULL;
    New -> rptr = NULL;
    l = New;
    r = New;
    return;
  }
  if (x <= l -> info)
  {
    New -> lptr = NULL;
    New -> rptr = l;
    l -> lptr = New;
    l = New;
```

```
  return;
 }
 temp = l;
 while (temp -> info < x && temp != NULL)
  temp = temp -> rptr;
 if (temp != NULL) {
  New -> lptr = temp -> lptr;
  New -> rptr = temp;
  temp -> lptr = New;
  New -> lptr -> rptr = New;
  return;
 }
 New -> rptr = NULL;
 New -> lptr = r;
 r -> rptr = New;
 r = New;
}

void doubdel(int x) {
 struct node * temp;
 if (l == NULL) //if(r==NULL)
 {
  printf("\n\n\tDoubly Linked List Underflow on Delete.");
  chk = 1;
  return;
 }
 if (x == l -> info)
 {
  temp = l;
  l = l -> rptr;
  l -> lptr = NULL;
  free(temp);
  return;
 }
 if (x == r -> info)
 {
  temp = r;
  r = r -> lptr;
  r -> rptr = NULL;
  free(temp);
  return;
 }
 temp = l;
 while (temp -> info != x && temp != NULL)
  temp = temp -> rptr;
 temp -> lptr -> rptr = temp -> rptr;
```

```
  temp -> rptr -> lptr = temp -> lptr;
  free(temp);
  return;
}

void display() {
  struct node * temp;
  if (l == NULL) //if(r==NULL)
    printf("\n\n\tDoubly Linked List is Empty.");
  else {
    printf("\n\nDoubly Linked List:\n\n\n");
    printf("l = %u\n\n", l);
    temp = l;
    while (temp != NULL) {
      printf("[%u|%u|%d|%u]->", temp, temp -> lptr, temp -> info, temp -> rptr);
      temp = temp -> rptr;
    }
    printf("NULL");
    printf("\n\nr = %u", r);
  }
}
```

OUTPUT

```
1.Insert Node
 2.Delete Node
 3.Display Doubly Linked List
 4.Exit
 Enter Choice: 1
        Enter Element: 10
 1.Insert Node
 2.Delete Node
 3.Display Doubly Linked List
 4.Exit
 Enter Choice:1
                Enter Element: 20
1.Insert Node
 2.Delete Node
 3.Display Doubly Linked List
 4.Exit
 Enter Choice: 1
        Enter Element: 30
 1.Insert Node
 2.Delete Node
 3.Display Doubly Linked List
 4.Exit
 Enter Choice:1
        Enter Element: 40
 1.Insert Node
 2.Delete Node
 3.Display Doubly Linked List
 4.Exit
 Enter Choice:2
                Enter element which you want to delete: 2
 1.Insert Node
 2.Delete Node
 3.Display Doubly Linked List
 4.Exit
 Enter Choice:3
      10 20 40
```

Practical 11

- **Write a program to implement following operations on the circular linked list.**
  (a) Insert a node at the end of the linked list.
  (b) Insert a node before specified position.
  (c) Delete a first node of the linked list.
  (d) Delete a node after specified position.

(a) Insert a node at the end of the linked list.

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

struct node {
	int info;
	struct node* next;
};

struct node* last = NULL;


void addatlast(int data)
{

	struct node* temp;
	temp = (struct node*)malloc(sizeof(struct node));

	if (last == NULL) {
		temp->info = data;
		temp->next = temp;
		last = temp;
	}

	else {
		temp->info = data;
		temp->next = last->next;
		last->next = temp;
		last = temp;
	}
}

void viewList()
{
```

```
        if (last == NULL)
                printf("\nList is empty\n");
        else {
                struct node* temp;
                temp = last->next;
                do {
                        printf("\nData = %d", temp->info);
                        temp = temp->next;
                } while (temp != last->next);
        }
}
int main()
{

        addatlast(10);
        addatlast(20);
        addatlast(30);
        viewList();

        return 0;
}
```

OUTPUT

Data = 10

Data = 20

Data = 30

(b) Insert a node before specified position.

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

struct node {
	int info;
	struct node* next;
};
struct node* last = NULL;

void addatlast()
{
	int data;


	struct node* temp;
	temp = (struct node*)malloc(sizeof(struct node));


	printf("\nEnter data to be inserted : \n");
	scanf("%d", &data);


	if (last == NULL) {
		temp->info = data;
		temp->next = temp;
		last = temp;
	}


	else {
		temp->info = data;
		temp->next = last->next;
		last->next = temp;
		last = temp;
	}
}

void insertafter()
{

	int data, value;

	struct node *temp, *n;
```

```
	printf("\nEnter number after which"
		" you want to enter number: \n");
	scanf("%d", &value);
	temp = last->next;

	do {
		if (temp->info == value) {
			n = (struct node*)malloc(sizeof(struct node));

			printf("\nEnter data to be"
				" inserted : \n");
			scanf("%d", &data);
			n->info = data;
			n->next = temp->next;
			temp->next = n;
			if (temp == last)
				last = n;
			break;
		}
		else
			temp = temp->next;
	} while (temp != last->next);
}

void viewList()
{
	if (last == NULL)
		printf("\nList is empty\n");
	else {
		struct node* temp;
		temp = last->next;
		do {
			printf("\nData = %d", temp->info);
			temp = temp->next;
		} while (temp != last->next);
	}
}
int main()
{
	// Initialize the list
	addatlast();
	addatlast();
	addatlast();

	// Function Call
	insertafter();
```

```
        viewList();

        return 0;
}
```

OUTPUT

```
Enter data to be inserted :
1

Enter data to be inserted :
2

Enter data to be inserted :
3

Enter number after which you want to enter number:
2

Enter data to be inserted :
4

Data = 1
Data = 2
Data = 4
Data = 3
```

(c) Delete a first node of the linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct node {
        int info;
        struct node* next;
};
struct node* last = NULL;

void addatlast(int data)
{
        struct node* temp;
        temp = (struct node*)malloc(sizeof(struct node));
        if (last == NULL) {
                temp->info = data;
                temp->next = temp;
                last = temp;
        }

        else {
                temp->info = data;
                temp->next = last->next;
                last->next = temp;
                last = temp;
        }
}

void deletefirst()
{
        struct node* temp;


        if (last == NULL)
                printf("\nList is empty.\n");



        else {
                temp = last->next;
                last->next = temp->next;
                free(temp);
        }
}

void viewList()
```

```
{
	if (last == NULL)
		printf("\nList is empty\n");
	else {
		struct node* temp;
		temp = last->next;
		do {
			printf("\nData = %d", temp->info);
			temp = temp->next;
		} while (temp != last->next);
	}
}
int main()
{

	addatlast(10);
	addatlast(20);
	addatlast(30);

	printf("Before deletion:\n");
	viewList();
	deletefirst();
	printf("\n\nAfter deletion:\n");
	viewList();

	return 0;
}
```

OUTPUT

Before deletion:

Data = 10

Data = 20

Data = 30

After deletion:

Data = 20

Data = 30

(d) Delete a node after specified position.

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

struct node {
        int info;
        struct node* next;
};

struct node* last = NULL;


void addatlast()
{

        int data;


        struct node* temp;
        temp = (struct node*)malloc(sizeof(struct node));


        printf("\nEnter data to be inserted : \n");
        scanf("%d", &data);
```

```
	if (last == NULL) {
		temp->info = data;
		temp->next = temp;
		last = temp;
	}


	else {
		temp->info = data;
		temp->next = last->next;
		last->next = temp;
		last = temp;
	}
}

void deleteAtIndex()
{


	int pos, i = 1;
	struct node *temp, *position;
	temp = last->next;


	if (last == NULL)
		printf("\nList is empty.\n");


	else {


		printf("\nEnter index : ");
		scanf("%d", &pos);


		while (i <= pos - 1) {
			temp = temp->next;
			i++;
		}


		position = temp->next;
		temp->next = position->next;

		free(position);
```

```
        }
}
void viewList()
{

        if (last == NULL)
                printf("\nList is empty\n");


        else {
                struct node* temp;
                temp = last->next;
                do {
                        printf("\nData = %d", temp->info);
                        temp = temp->next;
                } while (temp != last->next);
        }
}

int main()
{

        addatlast();
        addatlast();
        addatlast();


        deleteAtIndex();


        viewList();

        return 0;
}
```

OUTPUT

```
Enter data you be inserted: 10
Enter data you be inserted: 20
Enter data you be inserted: 30

Enter index :1

Data:10
Data:30
```

Practical 12

- **Write a program to implement Bubble Sort**

```
#include <stdio.h>

#include<conio.h>

void main()
{
        int array[100], n, c, d, swap;

        printf("How Many Element You Want To Add:");
        scanf("%d", & n);

        printf("Enter %d integers\n", n);

        for (c = 0; c < n; c++)
          scanf("%d", & array);

       for (c = 0; c < n - 1; c++)
       {
         for (d = 0; d < n - c - 1; d++)
        {
          if (array[d] > array[d + 1])
         {
            swap = array[d];
            array[d] = array[d + 1];
            array[d + 1] = swap;
         }
        }
       }

        printf("Sorted list in ascending order:\n");

          for (c = 0; c < n; c++)
              printf("%d\n", array);
}
```

OUTPUT

```
How many Elements You Want To Add: 5
Enter 5 integers
99
45
67
32
45
Sorted list in ascending order:
32
45
45
67
99
```

Practical 13

- **Write a program to implement Quick Sort**

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

int partition(int a[],int lb, int ub)
{
        int pivot,start,end,temp;
        pivot=a[lb];
        start=lb;
        end=ub;
        while(start<end)
        {
            while(a[start]<=pivot)
            start++;
            while(a[end]>pivot)
            end--;
            if(start<end)
            {
                temp=a[start];
                a[start]=a[end];
                a[end]=temp;
            }
        }
        temp=a[lb];
        a[lb]=a[end];
        a[end]=temp;
        return end;
 }
void quick_sort(int a[],int lb,int ub)
{
        int loc;
        if(lb<ub)
        {
           loc=partition(a,lb,ub);
           quick_sort(a,lb,loc-1);
           quick_sort(a,loc+1,ub);
        }
```

```
}
void main()
{
        int a[100],n,i,j,temp;

        printf("How many element you want to store in the array-> \n");
        scanf("%d",&n);
        printf("\nentered num is\n",n);
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        quick_sort(a,0,n-1);
        printf("\nsorted list in accending order is \n");
        for(i=0;i<n;i++)
        {
                printf("%d\n",a[i]);
        }
        getch();
}
```

OUTPUT

```
How many element you want to store in the array->
7

entered num is
46
79
5
16
63
46
82

sorted list in accending order is
5
16
46
46
63
79
82
```

Practical 14

- **Write a program to implement Merge Sort.**

```
#include <stdio.h>

void mergeSort(int[], int, int, int);
void partition(int[], int, int);

int main()
{
  int list[50];
  int i, size;

  printf("Enter Total Number of Elements:");
  scanf("%d", & size);
  printf("Enter The Elements:\n");
  for (i = 0; i < size; i++)
 {
    scanf("%d", & list[i]);
  }
  partition(list, 0, size - 1);
  printf("After Merge Sort:\n");
  for (i = 0; i < size; i++)
{
    printf("%d   ", list[i]);
  }

  return 0;
}

void partition(int list[], int low, int high)
{
  int mid;

  if (low < high)
 {
    mid = (low + high) / 2;
    partition(list, low, mid);
    partition(list, mid + 1, high);
    mergeSort(list, low, mid, high);
  }
}

void mergeSort(int list[], int low, int mid, int high)
```

```
{
        int i, mi, k, lo, temp[50];

        lo = low;
        i = low;
        mi = mid + 1;
        while ((lo <= mid) && (mi <= high))
        {
          if (list[lo] <= list[mi])
          {
            temp[i] = list[lo];
            lo++;
          }
          else
          {
            temp[i] = list[mi];
            mi++;
          }
          i++;
        }
        if (lo > mid)
        {
          for (k = mi; k <= high; k++)
          {
            temp[i] = list[k];
            i++;
          }
        }
         else
         {
          for (k = lo; k <= mid; k++)
          {
            temp[i] = list[k];
            i++;
          }
        }

        for (k = low; k <= high; k++)
        {
          list[k] = temp[k];
        }
}
```

OUTPUT

```
Enter Total Number of Elements: 5
Enter the Elements:
 89
34
56
76
22
After Merge Sort:
22  34   56  76  89
```

Practical 15

- **Write a program to implement Binary Search.**

```
#include <stdio.h>

#include<conio.h>

void main()
 {
        int c, first, last, middle, n, search, array[100];

        printf("How Many Element You Want To Add:");
        scanf("%d", & n);

        printf("Enter %d Integer Elements\n", n);

        for (c = 0; c < n; c++)
          scanf("%d", & array);

        printf("Enter Value to Find\n");
        scanf("%d", & search);

        first = 0;
        last = n - 1;
        middle = (first + last) / 2;

        while (first <= last)
       {
          if (array[middle] < search)
            first = middle + 1;
          else if (array[middle] == search)
          {
            printf("%d found at location %d.\n", search, middle + 1);
            break;
          }
          else
            last = middle - 1;

          middle = (first + last) / 2;
        }
        if (first > last)
```

```
        printf("Not found! %d isn't present in the list.\n", search);

    }
```

OUTPUT

```
How Many Elements You Want To Add: 5
Enter 5 Integers Elements:
12
23
34
45
56
Enter Value to Find:
56
56 found at location 5.
```