J.B. INSTITUTE OF ENGINEERING AND TECHNOLOGY

(UGC AUTONOMOUS)
Bhaskar Nagar, Moinabad Mandal, R.R. District, Hyderabad -500075

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

OBSERVATION

DATA STRUCTURES LAB
(R20)

Name of the Student:

Roll No:

Class & Section:

Index

12	Write C programs that implement the following sorting methods to sort agiven list of integers in ascending order: I) Bubble sort II) Selection sort III) Insertion Sort	
13	Write C programs that implement the following sorting methods to sort agiven list of integers in ascending order:	
14	Write a C Program to Implement the Hashing technique Write a C Program to Implement the KMP Pattern Searching Algorithm	

1. SINGLY LINKED LIST

```
#include<stdio.h>
struct node
{
int data;
struct node *next;
};
struct node *start=NULL;
struct node *first=NULL;
void createlist1()
{
struct node *newnode,*ptr; int n;
printf("enter the element to be inserted\n");
scanf("%d",&n);
newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=n;
if(start==NULL)
{
newnode->next=NULL;
start=newnode;
}
else
{
ptr=start;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
ptr->next=newnode;
newnode->next=NULL;
}
}
void insert_beg()
{
struct node *newnode,*ptr; int n;
printf("enter the element that has to inserted at beg\n");
```

```
scanf("%d",&n);

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

//ptr=start; newnode-
>data=n;

newnode->next=start;
start=newnode;

}

void insert_end()

{

struct node *newnode,*ptr; int n;

printf("enter the element that has to inserted at end\n");
scanf("%d",&n);

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

 ptr=start;

newnode->data=n;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=newnode;
newnode->next=NULL;

}

void insert_after()

{

struct node *newnode,*ptr,*preptr; int
n,num;

printf("enter a node after which the element has to be inserted\n");
scanf("%d",&num);

printf("enter the element that has to inserted\n");
scanf("%d",&n);

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

newnode->data=n;
```

```
ptr=start;
preptr=ptr;
while(preptr->data!=num)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=newnode;
newnode->next=ptr;
}
void insert_before()
{
struct node *newnode,*ptr,*preptr;
int n,num;
printf("enter a node before which the element has to be inserted\n");
scanf("%d",&num);
printf("enter the element that has to inserted\n");
scanf("%d",&n);
newnode=(struct node*)malloc(sizeof(struct node));
 newnode->data=n;
ptr=start;
preptr=ptr;
while(ptr->data!=num)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=newnode;
newnode->next=ptr;
}
void delete_beg()
{
struct node *ptr;
if(start==NULL)
printf("underflow\n") ;
else
{
```

```
ptr=start; start=start->next;
printf("the beg element deleted");
free(ptr);
}
}
void delete_end()
{
struct node *ptr,*preptr;
if(start==NULL)
printf("underflow\n") ;
else
{
ptr=start;
preptr=ptr;
while(ptr->next!=NULL)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=NULL;
printf("the end element deleted");
free(ptr);
}
}
void delete_after()
{
struct node *ptr,*preptr,*temp;
int num;
printf("enter the node after which we have to delete the node\n");
scanf("%d",&num);
if(start==NULL)
printf("underflow\n") ;
else
{
ptr=start;
preptr=ptr;
while(preptr->data!=num)
```

```
{
preptr=ptr;
ptr=ptr->next;
}
temp=ptr;
preptr->next=ptr->next;
printf("the element deleted");
free(temp);
}
}
void display1()
{
struct node *ptr;
ptr=start;
printf("elements of linked list-1 are\n");
while(ptr->next!=NULL)
{
printf("|%d|%u|->",ptr->data,ptr->next);
ptr=ptr->next;
}
printf("|%d|%u|\n",ptr->data,ptr->next);
}
void count_nodes()
{
struct node *ptr; int
c=0;
ptr=start;
while(ptr!=NULL)
{
c++;
ptr=ptr->next;
}
printf("no of nodes in linked list are %d\n",c);
}
void createlist2()
{
struct node *newnode,*ptr;
```

```
int n;
printf("enter the element to be inserted\n");
scanf("%d",&n);
newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=n;
if(first==NULL)
{
newnode->next=NULL;
first=newnode;
}
else
{
ptr=first;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
ptr->next=newnode;
newnode->next=NULL;
}
}
void display2()
{
struct node *ptr;
ptr=first;
printf("elements of linked list-2 are\n");
while(ptr->next!=NULL)
{
printf("|%d|%u|->",ptr->data,ptr->next);
ptr=ptr->next;
}
printf("|%d|%u|\n",ptr->data,ptr->next);
}
void mergelist()
{
struct node *ptr,*newptr;
ptr=start;
while(ptr->next!=NULL)
```

```
{
ptr=ptr->next;
}
ptr->next=first;
newptr=start;
printf("elements of list after merging are\n");
while(newptr->next!=NULL)
{
printf("|%d|%u|->",newptr->data,newptr->next);
newptr=newptr->next;
}
printf("|%d|%u|\n",newptr->data,newptr->next);
}
void main()
{
int ch;
clrscr(); printf("1.create1\n");
printf("2.Insert at beg\n");
printf("3.Insert at end\n");
printf("4.Insert after\n");
printf("5.Insert before\n");
printf("6.delete beg\n");
printf("7.delete end\n");
printf("8.delete after\n");
printf("9.display1\n");
printf("10.count nodes\n");
printf("11.create2 \n");
printf("12.display2\n");
printf("13.Merge 1 &2\n");
//printf("12.exit\n"); do
{
printf("enter the choice\n");
scanf("%d",&ch);
 switch(ch)
{
   case 1:createlist1();
          break;
   case 2:insert_beg();
          break;
   case 3:insert_end();
          break;
   case 4:insert_after();
          break;
   case 5:insert_before();
          break;
   case 6:delete_beg();
          break;
   case 7:delete_end();
          break;
   case 8:delete_after();
          break;
   case 9:display1();
          break;
 case 10:count_nodes();
          break;
   case 11:createlist2();
               break;
   case 12: display2();
               break;
 case 13:mergelist();
                break;
   default: printf("enetr correct choice\n");
                exit(0);
                }
                }
 while(ch<=13);
 getch();
 }
```

2. DOUBLY LINKED LIST

```
#include<stdio.h>
struct node
{
struct node *prev;
int data;
struct node *next;
};
struct node *start=NULL;
void createlist()
{
struct node *newnode,*ptr; int n;
printf("enter the element to be inserted\n");
scanf("%d",&n);
newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=n;
if(start==NULL)
{
newnode->next=NULL;
newnode->prev=NULL;
start=newnode;
}
else
{
ptr=start;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
ptr->next=newnode;
newnode->prev=ptr;
newnode->next=NULL;
}
}
void insert_beg()
{
struct node *newnode,*ptr;
```

```
int n;
printf("enter the element that has to inserted at beg\n");
scanf("%d",&n);
newnode=(struct node*)malloc(sizeof(struct node)); ptr=start;
newnode->data=n;
newnode->next=start;
newnode->prev=NULL;
start->prev=newnode;
start=newnode;
}
void insert_end()
{
struct node *newnode,*ptr; int n;
printf("enter the element that has to inserted at end\n");
scanf("%d",&n);
newnode=(struct node*)malloc(sizeof(struct node));
ptr=start;
newnode->data=n;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
ptr->next=newnode;
newnode->prev=ptr;
newnode->next=NULL;
}
void insert_after()
{
struct node *newnode,*ptr,*preptr;
int n,num;
printf("enter a node after which the element has to be inserted\n");
scanf("%d",&num);
printf("enter the element that has to inserted\n");
scanf("%d",&n);
newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=n;
```

```
ptr=start;
preptr=ptr;
while(preptr->data!=num)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=newnode;
newnode->prev=preptr;
newnode->next=ptr;
ptr->prev=newnode;
}
void insert_before()
{
struct node *newnode,*ptr,*preptr;
int n,num;
printf("enter a node before which the element has to be inserted\n");
scanf("%d",&num);
printf("enter the element that has to inserted\n");
scanf("%d",&n);
newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=n;
ptr=start;
preptr=ptr;
while(ptr->data!=num)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=newnode;
newnode->next=ptr;
newnode->prev=preptr;
ptr->prev=newnode;
}
void delete_beg()
{
struct node *ptr;
```

```
if(start==NULL)
printf("underflow\n") ; else
{
ptr=start;
 start=start->next;
start->prev=NULL;
printf("the beg element deleted");
free(ptr);
}
}
void delete_end()
{
struct node *ptr,*preptr;
if(start==NULL)
printf("underflow\n") ; else
{
ptr=start;
preptr=ptr;
while(ptr->next!=NULL)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=NULL;
printf("the end element deleted");
free(ptr);
}
}
void delete_after()
{
struct node *ptr,*preptr;
int num;
printf("enter the node after which we have to delete the node\n");
scanf("%d",&num);
if(start==NULL)
printf("underflow\n") ;
else
```

```
{
ptr=start;
preptr=ptr;
while(preptr->data!=num)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=ptr->next;
ptr->next->prev=preptr;
printf("the element deleted");
free(ptr);
}
}
void delete_before()
{
struct node *ptr,*preptr;
int num;
printf("enter the node after which we have to delete the node\n");
scanf("%d",&num);
if(start==NULL)
printf("underflow\n") ;
else
{
ptr=start;
preptr=ptr;
while(ptr->data!=num)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->prev->next=ptr;
ptr->prev=preptr->prev;
printf("the element deleted");
free(preptr);
}
}
```

```
void displayforward()
{
struct node *ptr;
ptr=start;
printf("elements of linked list-1 are\n");
while(ptr!=NULL)
{
printf("|%u|%d|%u|<-->",ptr->prev,ptr->data,ptr->next);
ptr=ptr->next;
}
}
void displaybackward()
{
struct node *ptr;
ptr=start;
printf("elements of linked list-1 are\n");
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
while(ptr!=NULL)
{
printf("|%u|%d|%u|<-->",ptr->prev,ptr->data,ptr->next);
ptr=ptr->prev;
}
}
void main()
{
int ch;
clrscr();
printf("1.create\n"); printf("2.Insert at
beg\n"); printf("3.Insert at end\n");
printf("4.Insert after\n");
printf("5.Insert before\n");
printf("6.delete beg\n");
printf("7.delete end\n");
printf("8.delete after\n");
printf("9.delete before\n");
printf("10.displayforward\n");
printf("11.displaybackward\n"); do

{

printf("enter the choice\n");
scanf("%d",&ch); switch(ch)

{
  case 1:createlist();
         break;

  case 2:insert_beg();
         break;

  case 3:insert_end();
         break;

  case 4:insert_after();
         break;

  case 5:insert_before();
         break;

  case 6:delete_beg();
         break;

  case 7:delete_end();
         break;

  case 8:delete_after();
         break;

  case 9:delete_before();
         break;

  case 10:displayforward();
         break;

  case 11:displaybackward();
         break;


  default: printf("enetr correct choice\n");
               exit(0);

               }

               }
while(ch<=11);
getch();

}
```

3. i) STACK USING ARRAY

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 100
Int stack[SIZE];

int top = -1;
void push()

{

    int e;

    if (top == SIZE-1) printf("Stack
    Overflow\n"); else

    {

    printf("Enter the element:"); scanf("%d",
    &e);

    top++;

    stack[top]=e;

    }

}

void pop()

{

     if (top ==-1)

     printf("Stack underflow \n"); else

     {

      printf("deleted element is : %d",stack[top]);
      stack[top]=NULL;

      top--;

     }

}

void peek()

{

   if (top ==-1)

     printf("Stack empty \n"); else

      printf("peek element is : %d",stack[top]);

}

void display()
```

```
{
int i;
if (top ==-1)
      printf("Stack empty \n"); else
{
printf("elements of stack are\n");
for(i=top;i>=0;i--)
printf("%d\n",stack[i]);
}
}
void main()
{
    int ch;
    clrscr();
    do
    {
      printf("\n1. Push\n");
      printf("2. Pop\n");
      printf("3. Peek\n");
      printf("4. Display\n");
      printf("Enter yourchoice: ");
      scanf("%d", &ch); switch(ch)
      {
          case 1:push();
                     break;
          case 2:pop();
                     break;
          case 3:peek();
                     break; case
          4:display();
                     break;
          default:
                  printf("Invalid choice, please try again.\n"); exit(0);
      }
    }
       while(ch<=4); getch();
}
```

ii)Stack using Linked List

```
#include<stdio.h>
#include<conio.h>
struct node
{
int data;
struct node *next;
};
struct node *top;
void push()
{
int val;
struct node *ptr;
ptr=(struct node*)malloc(sizeof(struct node));
printf("enter the value");
scanf("%d",val);
if(top==NULL)
{
ptr->data=val;
ptr->next=NULL;
top=ptr;
}
else
{
ptr->data=val; ptr-
>next=top;
top=ptr;
}
printf("item pushed");
}
void pop()
{
int item;
struct node *ptr;
if(top==NULL)
{
printf("underflow");
```

```
}
else
{
ptr=top; top=top-
>next; free(ptr);
printf("item popped");
}
}
void display()
{
int i;
struct node *ptr;
ptr=top;
if(ptr==NULL)
{
printf("stack is empty\n");
}
else
{
printf("printing stack elements\n");
while(ptr!=NULL)
{
printf("%d\n",ptr->data);
ptr=ptr->next;
}
}
}
void main()
{
int ch;
clrscr(); do
{
printf("\n\n chose one from the below options...\n");
printf("\n1.push\n2.pop\n3.display\n4.exit"); printf("\n enter
your choice\n");
scanf("%d",&ch);
switch(ch)
```

```
{
case 1:push();
break;
case 2:pop();
break;
case 3:display();
break;
default:printf("please enter valid choice"); exit(0);
}
}
while(ch<=3);
getch();
}
```

4. i) INFIX TO POST FIX CONVERSION

```
#include<stdio.h>
char stack[20];
int top=-1;
void push(char x)
{
stack[++top]=x;
}
char pop()
{
if (top==-1)
return-1; else
return stack[top--];
}
int priority(char x)
{ if(x=='(')
return 0;
if(x=='+'||x=='-')
return 1;
if(x=='*'||x=='/')
return 2;
}
void main()
{
char exp[20],pos[20];
char *e,x;
int i=0;
clrscr();
printf("enter the expression");
 scanf("%s",exp);
e=exp;
while(*e!='\0')
{
if(isalnum(*e))
pos[i++]=*e;
 else if(*e=='(')
push(*e);
else if(*e==')')
{
while((x=pop())!='(')
pos[i++]=x;
}
else
{
while(priority (stack[top])>=priority(*e)) pos[i++]=pop();
push(*e);
} e++;
}
while(top!=-1)
pos[i++]=pop();
pos[i]='\0';
printf("\n postfix expression %s" ,pos);
getch();
}
```

ii) POSTFIX EVALUATION

```
#include<stdio.h>
#include<conio.h>
Int stack[20];
int  top=-1;
void push(int x)
{
stack[++top]=x;
}
int pop()
{
return stack[top--];
}
void main()
{
char exp[20];
char *e;
int n1,n2,n3,num;
clrscr();
printf("enter the expression::");
scanf("%s",exp);
e=exp;
while(*e!='\0')
{
if(isdigit(*e))
{
num=*e-48;
push(num);
}
else
{
n1=pop();
n2=pop();
switch(*e)
{
```

```
case '+':
{
n3=n1+n2;
break;
}
case '-':
{
n3=n2-n1;
break;
}
case '*':
{
n3=n1*n2;
break;
}
case '/':
{
n3=n2/n1;
break;
}
}
push(n3);
} e++;
}
printf("\n the result of expression %s=%d\n\n" ,exp,pop());
getch();
}
```

5. Recursion

i)
```
#include<stdio.h>
int fact(int);
int main()
{
int num,val;
printf("enter the number");
scanf("%d" ,&num);
val=fact(num);
printf("factorial of %d=%d" ,num,val);
}
int fact(int n)
{
if(n==1||n==0)
return 1;
else
return(n*fact(n-1));
getch();
}
```

ii) INFIX TO PREFIX

```
#include<stdio.h>
char stack[20];

int top=-1;

void push(char x)

{

stack[++top]=x;

}

char pop()

{

if (top==-1)
return-1; else

return stack[top--];

}

int priority(char x)

{ if(x=='(')
return 0;

if(x=='+'||x=='-')
return 1;
if(x=='*'||x=='/')
return 2;

}

void main()

{

char exp[20],pos[20];
char *e,x;

int i=0;
clrscr();

printf("enter the expression");
scanf("%s" ,exp); e=strrev(exp);

while(*e!='\0')

{

if(isalnum(*e))
pos[i++]=*e; else
if(*e=='(') push(*e);
else if(*e==')')

{

while((x=pop())!='(')
pos[i++]=x;

}

else

{

while(priority (stack[top])>=priority(*e)) pos[i++]=pop();

push(*e);

} e++;

}

while(top!=-1)
pos[i++]=pop();

pos[i]='\0';

printf("\n prefix expression %s" ,strrev(pos));

 getch();

}
```

6. **i) QUEUE USING ARRAY**

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 100 int
queue[SIZE];
 int front=-1,rear=-1;
 void insert()
 {
 int e;
 if(rear==SIZE-1)
 printf("queue overflow");
 else
 {
 printf("enter the element");
 scanf("%d",&e);
 if(front==- 1&&rear==-1)
 front=rear=0;
 else
 rear=rear+1;
 }
 queue[rear]=e;
 }
 void delete()
 {
 int val;
 if(front==-1||front>rear)
 printf("queue underflow"); else
 {
 val=queue[front];
 printf("element deleted is %d" ,val);
 front=front+1;
 }
 }
 void display()
 {
 int i;
 if(front==-1||front>rear)
 printf("queue is empty");
 else
 {
 printf("elements in the queue are");
 for(i=front;i<=rear;i++)
 printf("%d\n",queue[i]);
 }
 }
 void main()
 {
 int ch;
 clrscr();
 do
 {
 printf("\n1.insert\n");
 printf("2.delete\n");
 printf("3.dispaly\n");
 printf("enter your choice");
 scanf("%d" ,&ch);
 switch(ch)
 {
 case  1:  insert();  break;
 case 2: delete(); break;
 case 3: display(); break;
 default:
 printf("invalid choice,please try again\n"); exit(0);
 }
 }
 while(ch<=3);
 getch();
 }
```

ii) QUEUE LINKEDLIST

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node*next;
};
struct      node       *rear=NULL;
struct  node*front=NULL;  void
insert()
{
int val;
struct node *ptr;
ptr=(struct node*)malloc(sizeof(struct node));
printf("enter the value");
scanf("%d" ,&val);
ptr->data=val;
if(front==NULL)
{
front=rear=ptr;
rear->next=NULL;
}
else
{
rear->next=ptr; ptr-
>next=NULL;
rear=ptr;
}
printf("%d element inserted",ptr->data);
}
void del()
{
int item;
```

```
struct node*ptr;
if(front==NULL)
printf("underflow");
else
{
ptr=front;
front=front->next;
free(ptr);
}
printf("item deleted");
}
void display()
{
int i;
struct node *ptr; ptr=front;
if(front==NULL)
printf("queue is empty \n"); else
{
printf("printing queue elements \n");
while(ptr->next!=NULL)
{
printf("%d->" ,ptr->data);
ptr=ptr->next;
}
printf("%d" ,ptr->data);
}
}
void main()
{
int ch;
clrscr();
do
{
printf("\n\nchoose one from the below option\n");
printf("\n1.insert\n2.delete\n3.display\n4.exit"); printf("\n
enter your choice\n");
scanf("%d" ,&ch);
switch(ch)
{
case 1:insert();
break;
case 2:del();
break;
case 3:display();
break;
case 4:printf("exisiting");
break;
default:
{
printf("please enter valid choice");
}
}
}
while(ch<=4);
getch();
}
```

7. CIRCULAR QUEUE

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 100
Int queue[SIZE];

int front=-1,rear=-1;
void insert()

{

int val; if(front==0&&rear==SIZE-1)
printf("queue overflow");

else

{

printf("enter the element");
scanf("%d" ,&val);

if(front==-1&&rear==-1)
front=rear=0;

else if(rear==SIZE-1&&front!=0)

rear=0;

else
rear=rear+1;

}

queue[rear]=val;

printf("element inserted\n");

}

void delete()

{

int val;
if(front==-1)

{

printf("queue underflow\n");

}

val=queue[front];

printf("element deleted is %d" ,val);


if(front==rear)front=rear-1;

else if(front==SIZE-1)
```

```
front=0;
else
front=front+1;
}
void display()
{
int i;
if(front==-1&&rear==-1)
printf("queue is empty");
else
if(front<rear)
{
for(i=front;i<=rear;i++)
printf("%d",queue[i]);
}
else
{
for(i=front;i<=SIZE;i++)
printf("%d" ,queue[i]);
for(i=0;i<=rear;i++)
 printf("%d",queue[i]);
}
}
void main()
{
int ch;
clrscr();
 do
{
printf("1.insert \n");
printf("2.delete\n");
printf("3.display\n");
printf("enter your choice");
scanf("%d" ,&ch); switch(ch)
{
case 1: insert(); break;
case 2: delete(); break;

case 3: display(); break;

default:

printf("invalid choice please try again\n"); exit(0);

}

}

while(ch<=3);
getch();

}
```

8. DEQUEUE

```
#include<stdio.h>
#include<conio.h>
#define MAX 10 int
deque[MAX];
int left=-1,right=-1;
void input_deque();

void output_deque();

void insert_left(); void
insert_right(); void
delete_left(); void
delete_right(); void
display();


void main()

{

int option;
clrscr();

printf("\n *****main menu*****"); printf("\n
1.input restricted deque"); printf("\n 2.output
restricted deque"); printf("enter your option:");
scanf("%d",&option);

switch(option)

{

case 1:
input_deque();
break;

case 2:
output_deque();
break;

}

getch();

}

void input_deque()

{

int option;

do
```

```
{
printf("\n INPUT RESTRICTED DEQUE");
printf("\n 1.insert at right"); printf("\n
2.delete from left"); printf("\n
3.delete from right"); printf("\n
4.display"); printf("\n 5.quit");
printf("enter your option");
scanf("%d",&option); switch(option)
{
case 1:
insert_right();
break;
case 2:
delete_left(); break;
case 3:
delete_right();
break;
case 4:
display();
break;
}
}
while(option!=5);
}
void output_deque()
{
int option; do
{
printf("\n OUTPUT RESTRICTED DEQUE");
printf("\n 1.insert at right");
printf("\n 2.insert at left"); printf("\n
3.delete from left"); printf("\n
4.display"); printf("\n 5.quit");
printf("enter your option");
scanf("%d",&option); switch(option)
{
case 1:
insert_right();
break;
case 2:
insert_left();
break;
case 3:
delete_left(); break;
case 4:
display();
break;
}
}
while(option!=5);
}
void insert_right()
{
int val;
printf("\n enter the value to be added");
scanf("%d",&val);
if((left==0&&right==MAX-1)||(left==right+1))
{
printf("\n OVERFLOW"); return;
}
if(left==-1)
{
left=0;
right=0;
}
else
{
if(right==MAX-1)
right=0;
else right=right+1;
}
```

```
deque[right]=val;
}
void insert_left()
{
int val;
printf("\n enter the value to be added:");
scanf("%d",&val);
if((left==0&&right==MAX-1)||(left==right+1))
{
printf("\n OVERFLOW"); return;
}
if(left==-1)
{
left=0;
right=0;
}
else
{
if(left==0)
left=MAX-1; else
left=left-1;
}
deque[left]=val;
}
void delete_left()
{
if(left==-1)
{
printf("\n UNDERFLOW"); return;
}
printf("\n the deleted element is :%d",deque[left]);
if(left==right)
{
left=-1;
right=-1;
}
```

```
else
{
if(left==MAX-1)
left=0;
else
left=left+1;
}
}
void delete_right()
{
if(left==-1)
{
printf("\n UNDERFLOW"); return;
}
printf("\n the element deleted is:%d",deque[right]);
if(left==right)
{
left=-1;
right=-1;
}
else
{
if(right==0)
right=MAX-1; else
right=right-1;
}
}
void display()
{
int front=left,rear=right;
if(front==-1)
{
printf("\n QUEUE IS EMPTY"); return;
}
printf("\n the elements of the queue are:");
if(front<=rear)
{
    while(front<=rear)
    {
    printf("%d",deque[front]);
    front++;
    }
    }
    else
    {
    while(front<=MAX-1)
    {
    printf("%d",deque[front]);
    front++;
    }
    front=0;
    while(front<=rear)
    {
    printf("%d",deque[front]);
    front++;
    }
    }
    printf("\n");
}
```

9.

i) BINARY TREE TRAVERSALS

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

#include <stdlib.h>

struct node

{

int data; //node will store an integer struct node
*right_child; // right child struct node
*left_child; // left child

};

struct node* search(struct node *root, int x)

{

if(root==NULL || root->data==x) //if root->data is x then the element is found return root;

else if(x>root->data) // x is greater, so we will search the right subtree


return search(root->right_child, x);


else //x is smaller than the data, so we will search the left subtree return
search(root->left_child,x);

}

//function to find the minimum value in a node struct
node* find_minimum(struct node *root)

{

if(root == NULL) return
NULL;

else if(root->left_child != NULL) // node with minimum value will have no left child return
find_minimum(root->left_child); // left most element will be minimum

return root;

}

//function to create a node struct
node* new_node(int x)

{

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

 p->data= x;

p->left_child = NULL;

p->right_child = NULL;

return p;

}
```

```
struct node* insert(struct node *root,int x)

{

//searching for the place to insert if(root==NULL)

return new_node(x);

else if(x>root->data) // x is greater. Should be inserted to right root-
>right_child = insert(root->right_child, x);

else // x is smaller should be inserted to left

root->left_child = insert(root->left_child,x);

return root;

}

// function to delete a node

struct node* delet(struct node *root,int x)

{

struct node *temp;

//searching for the item to be deleted
if(root==NULL)

return NULL;

if (x>root->data)

root->right_child = delet(root->right_child,x);

elseif(x<root->data)

root->left_child = delet(root->left_child,x);

else

{

//No Children

if(root->left_child==NULL && root->right_child==NULL)

{

free(root);

return NULL;

}

//One Child

else if(root->left_child==NULL || root->right_child==NULL)

{

if(root->left_child==NULL)

temp =root->right_child;

else

temp = root->left_child;
free(root);

return temp;

}

//Two Children

else
{

struct node *temp = find_minimum(root->right_child);

root->data= temp->data;

root->right_child = delet(root->right_child,temp->data);

}

}

return root;

}

void inorder(struct node *root)

{

if(root!=NULL) // checking if the root is not null

{

inorder(root->left_child); // visiting left child

printf(" %d ",root->data); // printing data at root
inorder(root->right_child);// visiting right child

}

}

void preorder(struct node *root)

{

if(root!=NULL) // checking if the root is not null

{ printf(" %d ", root->data); // printing data at root
preorder(root->left_child); // visiting left child

preorder(root->right_child);// visiting right child

}

}

void postorder(struct node *root)

{

if(root!=NULL) // checking if the root is not null

{

postorder(root->left_child); // visiting left child
postorder(root->right_child);// visiting right child
```

```
printf("%d ", root->data); // printing data at root
}
}
void display(struct node *root)
{
printf("in order traversal:");
inorder(root);
printf("\npre order traversal:");
preorder(root);
printf("\npost order traversal:");
postorder(root);
}
void main()
{
struct node *root,*rt;
inti,r,n,x,num,d,ch;
clrscr();
do{ printf("\n1.insert\n2.delete\n3.search\n4.display\n5...exit");
printf("\nenter your choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("enter number of nodes to be inserterd:");
scanf("%d",&n);
printf("Enter the data of the root node: ");
scanf("%d",&r);
root=new_node(r);
printf("Enter the data of nodes: ");
for(i=0;i<n-1;i++)
{
scanf("%d",&num);
insert(root,num);
}
break;
case 2:
printf("Enter the data of the node to be deleted: ");
scanf("%d",&d);
root=delet(root,d);
printf("Nodeis deleted");

break;

case 3:printf("enter the data of node to be searched:");
scanf("%d",&x);

rt=search(root,x);
if(rt!=NULL) printf("node
exits"); else

printf("node not exist");
break;
case 4:display(root);
break; default:exit(0);

}

}while(ch<=4);
getch();

}

                        default:exit(0);

                       }

            }while(op!=5);
```

ii. AVL Tree

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

// structure of the tree node
struct node
{
    int data;
    struct node* left;
    struct node* right;
    int ht;
};

// global initialization of root node
struct node* root = NULL;

// function prototyping
struct node* create(int);
struct node* insert(struct node*, int);
struct node* delete(struct node*, int);
struct node* search(struct node*, int);
struct node* rotate_left(struct node*);
struct node* rotate_right(struct node*);
int balance_factor(struct node*);
int height(struct node*);
void inorder(struct node*);
void preorder(struct node*);
void postorder(struct node*);

int main()
{
    int user_choice, data;
    char user_continue = 'y';
    struct node* result = NULL;

    while (user_continue == 'y' || user_continue == 'Y')
    {
        printf("\n\n------- AVL TREE --------\n");
        printf("\n1. Insert");
        printf("\n2. Delete");
        printf("\n3. Search");
        printf("\n4. Inorder");
        printf("\n5. Preorder");
        printf("\n6. Postorder");
        printf("\n7. EXIT");
        printf("\n\nEnter Your Choice: ");
        scanf("%d", &user_choice);

        switch(user_choice)
        {
            case 1:
                printf("\nEnter data: ");
                scanf("%d", &data);
                root = insert(root, data);
                break;

            case 2:
                printf("\nEnter data: ");
                scanf("%d", &data);
                root = delete(root, data);
                break;

            case 3:
                printf("\nEnter data: ");
                scanf("%d", &data);
                result = search(root, data);
                if (result == NULL)
                {
                    printf("\nNode not found!");
                }
                else
                {
                    printf("\n Node found");
                }
                break;
            case 4:
                inorder(root);
                break;

            case 5:
                preorder(root);
                break;

            case 6:
                postorder(root);
                break;

            case 7:
                printf("\n\tProgram Terminated\n");
                return 1;

            default:
                printf("\n\tInvalid Choice\n");
        }

        printf("\n\nDo you want to continue? ");
        scanf(" %c", &user_continue);
    }

    return 0;
}

// creates a new tree ndoe
struct node* create(int data)
{
    struct node* new_node = (struct node*) malloc (sizeof(struct node));

    // if a memory error has occurred
    if (new_node == NULL)
    {
        printf("\nMemory can't be allocated\n");
        return NULL;
    }
    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}

// rotates to the left
struct node* rotate_left(struct node* root)
{
    struct node* right_child = root->right;
    root->right = right_child->left;
    right_child->left = root;

    // update the heights of the nodes
    root->ht = height(root);
    right_child->ht = height(right_child);

    // return the new node after rotation
    return right_child;
}
```

```
// rotates to the right
struct node* rotate_right(struct node* root)
{
    struct node* left_child = root->left;
    root->left = left_child->right;
    left_child->right = root;

    // update the heights of the nodes
    root->ht = height(root);
    left_child->ht = height(left_child);

    // return the new node after rotation
    return left_child;
}

// calculates the balance factor of a node
int balance_factor(struct node* root)
{
    int lh, rh;
    if (root == NULL)
        return 0;
    if (root->left == NULL)
        lh = 0;
    else
        lh = 1 + root->left->ht;
    if (root->right == NULL)
        rh = 0;
    else
        rh = 1 + root->right->ht;
    return lh - rh;
}

// calculate the height of the node
int height(struct node* root)
{
    int lh, rh;
    if (root == NULL)
    {
        return 0;
    }
    if (root->left == NULL)
        lh = 0;
    else
        lh = 1 + root->left->ht;
    if (root->right == NULL)
```

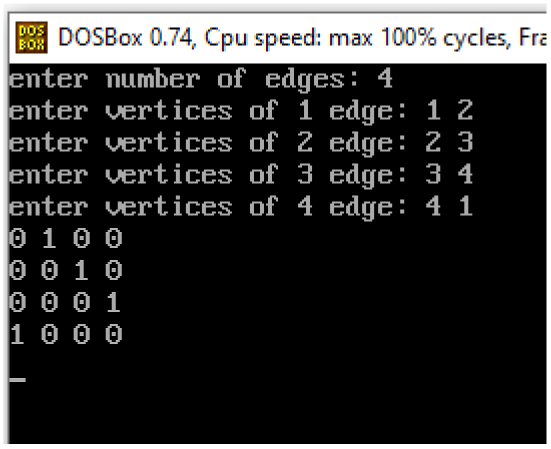
```
        rh = 0;
    else
        rh = 1 + root->right->ht;

    if (lh > rh)
        return (lh);
    return (rh);
}

// inserts a new node in the AVL tree
struct node* insert(struct node* root, int data)
{
    if (root == NULL)
    {
        struct node* new_node = create(data);
        if (new_node == NULL)
        {
            return NULL;
        }
        root = new_node;
    }
    else if (data > root->data)
    {
        // insert the new node to the right
        root->right = insert(root->right, data);

        // tree is unbalanced, then rotate it
        if (balance_factor(root) == -2)
        {
            if (data > root->right->data)
            {
                root = rotate_left(root);
            }
            else
            {
                root->right = rotate_right(root->right);
                root = rotate_left(root);
            }
        }
    }
    else
    {
        // insert the new node to the left
        root->left = insert(root->left, data);
        // tree is unbalanced, then rotate it
        if (balance_factor(root) == 2)
        {
            if (data < root->left->data)
            {
                root = rotate_right(root);
            }
            else
            {
                root->left = rotate_left(root->left);
                root = rotate_right(root);
            }
        }
    }
    // update the heights of the nodes
    root->ht = height(root);
    return root;
}

// deletes a node from the AVL tree
struct node * delete(struct node *root, int x)
{
    struct node * temp = NULL;

    if (root == NULL)
    {
        return NULL;
    }

    if (x > root->data)
    {
        root->right = delete(root->right, x);
        if (balance_factor(root) == 2)
        {
            if (balance_factor(root->left) >= 0)
            {
                root = rotate_right(root);
            }
            else
            {
                root->left = rotate_left(root->left);
                root = rotate_right(root);
            }
        }
    }
    else if (x < root->data)
    {
        root->left = delete(root->left, x);
        if (balance_factor(root) == -2)
        {
            if (balance_factor(root->right) <= 0)
            {
                root = rotate_left(root);
            }
            else
            {
                root->right = rotate_right(root->right);
                root = rotate_left(root);
            }
        }
    }
    else
    {
        if (root->right != NULL)
        {
            temp = root->right;
            while (temp->left != NULL)
                temp = temp->left;

            root->data = temp->data;
            root->right = delete(root->right, temp->data);
            if (balance_factor(root) == 2)
            {
                if (balance_factor(root->left) >= 0)
                {
                    root = rotate_right(root);
                }
                else
                {
                    root->left = rotate_left(root->left);
                    root = rotate_right(root);
                }
            }
        }
        else
        {
            return (root->left);
        }
    }
    root->ht = height(root);
    return (root);
}

// search a node in the AVL tree
struct node* search(struct node* root, int key)
{
    if (root == NULL)
    {
        return NULL;
    }

    if(root->data == key)
    {
        return root;
    }

    if(key > root->data)
    {
        search(root->right, key);
    }
    else
    {
        search(root->left, key);
    }
}

// inorder traversal of the tree
void inorder(struct node* root)
{
    if (root == NULL)
    {
        return;
    }

    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

// preorder traversal of the tree
void preorder(struct node* root)
{
    if (root == NULL)
    {
        return;
    }

    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}

// postorder traversal of the tree
void postorder(struct node* root)
{
    if (root == NULL)
    {
        return;
    }

    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}
```

10.

i) ADJACENCY MATRIX

```
#include<stdio.h>
#define V 5

//init matrix to 0 void
init(int arr[][V])

{

       int i,j; for(i=0;i<V;i++)

               for(j=0;j<V;j++)
                   arr[i][j]=0;

}

//Add edge. set arr[src][dest] = 1

void addEdge(int arr[][V],int src, int dest)

{

        arr[src][dest]=1;

}

void printAdjMatrix(int arr[][V])

{

        int i,j; for(i=1;i<V;i++)

        {

                for(j=1;j<V;j++)

                {

                    printf("%d ", arr[i][j]);

                }

                printf("\n");

        }

}

//print the adjMatrix void
main()

{

       int adjMatrix[V][V]; int
       x,y,n,i;

       clrscr(); init(adjMatrix);

       printf("enter number of edges: ");
       scanf("%d",&n); for(i=1;i<=n;i++)

       {

       printf("enter vertices of %d edge: ",i);
       scanf("%d%d",&x,&y); addEdge(adjMatrix,x,y);
       }
       printAdjMatrix(adjMatrix);
        getch();
       }
```

Output:



DOSBox 0.74, Cpu speed: max 100% cycles, Fra

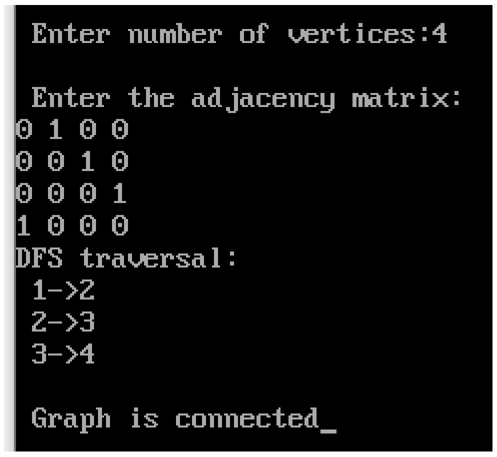
```
enter number of edges: 4
enter vertices of 1 edge: 1 2
enter vertices of 2 edge: 2 3
enter vertices of 3 edge: 3 4
enter vertices of 4 edge: 4 1
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0
_
```

ii) GRAPH TRAVERSALS

a)DFS

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

int a[20][20],reach[20],n; void
dfs(int v) {

        int i;
        reach[v]=1;

        for (i=1;i<=n;i++) if(a[v][i] &&
          !reach[i]) {

                printf("\n %d->%d",v,i); dfs(i);

        }

}

void main() {

        int i,j,count=0; clrscr();

        printf("\n Enter number of vertices:");
        scanf("%d",&n);

        for (i=1;i<=n;i++) {

                reach[i]=0;

                for (j=1;j<=n;j++)
                  a[i][j]=0;

        }

        printf("\n Enter the adjacency matrix:\n");

         for(i=1;i<=n;i++)

          for (j=1;j<=n;j++)
           scanf("%d",&a[i][j]);

        printf("DFS traversal:");

        dfs(1);

        printf("\n");
        for (i=1;i<=n;i++) {
                if(reach[i])
                   count++;
        }
        if(count==n)
          printf("\n Graph is connected"); else
          printf("\n Graph is not connected"); getch();
}
```

Output:

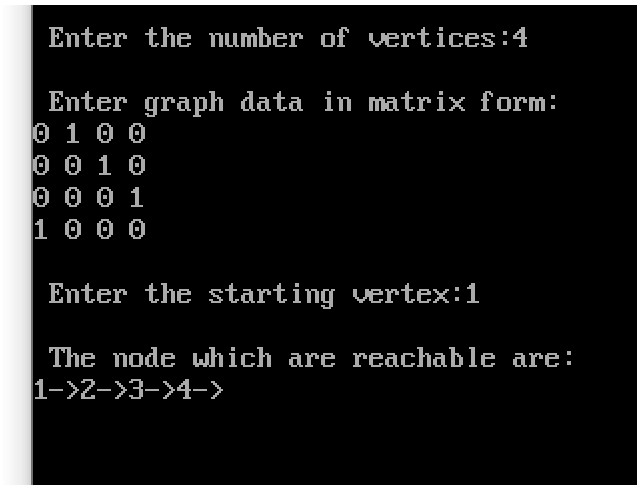


```
 Enter number of vertices:4

 Enter the adjacency matrix:
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0
DFS traversal:
 1->2
 2->3
 3->4

 Graph is connected_
```

b) BFS #include<stdio.h>

```
#include<conio.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1; void
bfs(int v) {

        for (i=1;i<=n;i++) if(a[v][i] &&
          !visited[i])

          q[++r]=i;

          if(f<=r) {

                visited[q[f]]=1;

                bfs(q[f++]);

        }

}

void main() {

        int v;
        clrscr();

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

        for (i=1;i<=n;i++)

         {

        q[i]=0;

         visited[i]=0;

        }

        printf("\n Enter graph data in matrix form:\n"); for
        (i=1;i<=n;i++)

          for (j=1;j<=n;j++)
           scanf("%d",&a[i][j]);

        printf("\n Enter the starting vertex:");
        scanf("%d",&v);

        bfs(v);

        printf("\n The node which are reachable are:\n"); for
        (i=1;i<=n;i++)

          if(visited[i])
           printf("%d->",i);
           else

           printf("\n Bfs is not possible"); getch();

}
```

Output:



```
 Enter the number of vertices:4

 Enter graph data in matrix form:
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0

 Enter the starting vertex:1

 The node which are reachable are:
1->2->3->4->
```

11.

i) Write a C program for linear search.

Program:

```
#include<stdio.h> void
main()

{

int a[100],n,key,pos,i; clrscr();

printf("enter size");
scanf("%d",&n);

printf("enter elements");
for(i=0;i<n;i++)
scanf("%d",&a[i]);

printf("enter key");
scanf("%d",&key);
for(i=0;i<n;i++)

{

if(key==a[i])

{

pos=0;
break;

}

}

if(pos!=0)

printf("Element is not found"); else

printf("Element %d is found at %d position",key,i+1);

getch();

}
```

```
Output enter
size:5

Enter elements:12 34 2 67 9

enter key: 9

Element 9 found at 5 position.
```

ii) Write a C program for using binary search method Program:

```
#include<stdio.h>
#include<conio.h> void
main()

{

int a[100],i,n,key,pos; int
mid,low,high; clrscr();

printf("enter size");
scanf("%d",&n);

printf("enter sorted elements");
for(i=0;i<n;i++)

scanf("%d",&a[i]);
printf("enter key");
scanf("%d",&key);

low=0;
high=n-1;

while(low<=high)

{

mid=(low+high)/2;
if(key==a[mid])

{

pos=0;
break;

}

else if(key>a[mid]) low=mid+1;

else high=mid-1;

}

if(pos!=0)

printf("element is not found"); else

printf("element %d is found at %d position",key,mid+1); getch();

}
```

OUTPUT:

```
Enter size: 5

Enter sorted elements: 12 23 56 78 99 Enter
key:78

Element 78 is found at 4 position
```

12.

i) Write a C program that implements the Bubble sort method Program:

```
#include<stdio.h>
#include<conio.h>
void main()

{

int i,j,t,a[5],n; clrscr();

printf("enter the range of array:");

scanf("%d",&n);

printf("enter elements into array:");
for(i=0;i<n;i++)

scanf("%d",&a[i]);

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

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

if(a[i]>a[j])

{

t=a[i];
a[i]=a[j];
a[j]=t;

}

printf("the sorted order is:");
for(i=0;i<n;i++) printf("\t%d",a[i]);

getch();

}
```

OUTPUT:

```
Enter the range of the array:5

Enter the elements in the array:23 2 56 26 7

The sorted order is:2 7 23 26 56
```

ii) Write a C program that implements the Selection sort method

```
#include<stdio.h>
#include<conio.h> void
main()

{

int a[100],n,i,j,min,temp;
clrscr();

printf("\n Enter the Number of Elements: ");

scanf("%d",&n);printf("\n Enter %d Elements: ",n);
for(i=0;i<n;i++)
scanf("%d",&a[i]);

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

{
min=i;

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

{

if(a[min]>a[j]) min=j;

}

if(min!=i)

{

temp=a[i]; a[i]=a[min];
a[min]=temp;

}

}

printf("\n The Sorted array in ascending order: ");
for(i=0;i<n;i++)

{

printf("%d ",a[i]);

}

getch();

}
```

OUTPUT:

```
Enter the size of the array: 5 Enter the
elements: 23 2 56 26 7

Elements in Sorted order: 2 7 23 26 56
```

iii) Write a C program that implements the Insertion sort method

Program:

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

void insertionsort(int a[],int size); void
main()

{

int n,a[10],i,x; clrscr();

printf("\nEnter the size of array: ");
scanf("%d",&n);

printf("\nEnter the array elements: ");
for(i=0;i<n;i++)

scanf("%d",&a[i]);

printf("\nElements in array before swapping: "); for(i=0;i<n;i++)
printf(" %d",a[i]);
printf("\n");
insertionsort(a,n);

printf("\n\nElements in array after sorting: ");
for(x=0;x<n;x++)

printf(" %d",a[x]);

getch();

}

void insertionsort(int a[],int n)

{

int k,j,temp,x;
for(k=0;k<n;k++)

{

temp=a[k];
j=k-1;

while((j>=0)&&(temp<=a[j]))

{

a[j+1]=a[j];
a[j]=temp; j=j-
1;

}

}

}
```

OUTPUT:

```
Enter the size of the array: 5
Enter the array elements: 23 2 56 26 7

Elements in array before swapping: 23 2 56 26 7

Elements in array after swapping: 2 7 23 26 56
```

13.

i) Write a C program that implements the Merge sort method

Program:

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

void merge(int [],int,int,int);

void mergesort(int a[],int low,int high)

{

int mid;
if(low<high)

{

mid = (low + high)/2;
mergesort(a,low,mid);
mergesort(a,mid+1,high);
merge(a,low,high,mid);

}

}

void merge(int a[],int l,int h,int m)

{

int c[100],i,j,k;

i = l; j = m + 1; k = l; while(i <= m
&& j <= h)

{

if(a[i] < a[j])

{

c[k] = a[i]; i++;

k++;

}

else

{

c[k] =a[j]; j++;

k++;

}

}
while(i <= m)
c[k++]= a[i++];
while(j <= h)
c[k++] = a[j++];
for(i = l; i < k; i++) a[i] = c[i];
}
void main()
{
int i,n,a[100];
clrscr();
printf("\n Enter the size of the array :");
scanf("%d",&n);
printf("\n Enter the elements :\n");
for(i =0; i < n; i++) scanf("%d",&a[i]);
mergesort(a,0,n-1);
printf("\n Elements in sorted order :\n");
for(i =0; i < n; i++)
printf("%5d",a[i]);
getch();
}
```

OUTPUT:

```
Enter the size of the array: 5 Enter the
elements: 23 2 56 26 7

Elements in Sorted order: 2 7 23 26 56
```

ii) Write a C program that implements the Quicksort method Program:

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

void quicksort(int [10],int,int); int
main()

{

int x[20],size,i;
clrscr();

printf("\nEnter size of the array: ");
scanf("%d",&size);

printf("\nEnter %d elements: ",size);
for(i=0;i<size;i++)

scanf("%d",&x[i]); quicksort(x,0,size-
1); printf("\nSortedelements: ");
for(i=0;i<size;i++)

printf(" %d",x[i]);
getch();

return 0;

}

void quicksort(int x[10],int first,int last){ int
pivot,j,temp,i;

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

while(x[i]<=x[pivot]&&i<last) i++;

while(x[j]>x[pivot]) j--;

if(i<j){
temp=x[i];
x[i]=x[j];
x[j]=temp;

}

}

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

}

}
```

OUTPUT:

```
Enter the size of the array: 5 Enter 5
elements: 23 2 56 26 7

Sorted elements: 2 7 23 26 56
```

14.

i) HASHING

Program:

```
/* HASHING - LINEAR AND QUADRATIC PROBING */

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

Int tsize;

int hasht(int key)

{

   int i ;

   i = key%tsize ;

   return i;

}

//-------LINEAR PROBING-------

int rehashl(int key)

{

   int i ;

   i = (key+1)%tsize ; return i ;

}

//-------QUADRATIC PROBING-------

int rehashq(int key, int j)

{

   int i ;

   i = (key+(j*j))%tsize ; return i ;

}


void main()

{

       int key,arr[20],hash[20],i,n,s,op,j,k ; clrscr() ;

       printf ("Enter the size of the hash table: ");

       scanf("%d",&tsize);

       printf ("\nEnter the number of elements: ");
       scanf("%d",&n);
       for (i=0;i<tsize;i++)
   hash[i]=-1 ;
   printf ("Enter Elements: "); for
   (i=0;i<n;i++)

   {

scanf("%d",&arr[i]);

   }
   do

   {

printf("\n\n1.Linear Probing\n2.Quadratic Probing \n3.Exit \nEnter your option: "); scanf("%d",&op);

switch(op)

{

case 1:

    for (i=0;i<tsize;i++)
    hash[i]=-1 ;

    for(k=0;k<n;k++)

    {

 key=arr[k] ;

 i = hasht(key); while
 (hash[i]!=-1)

 {

     i = rehashl(i);

 }

 hash[i]=key ;

    }

    printf("\nThe elements in the array are: "); for
    (i=0;i<tsize;i++)

    {

 printf("\n Element at position %d: %d",i,hash[i]);

    }

    break ;


case 2:

    for (i=0;i<tsize;i++)
 hash[i]=-1 ;
```

```
          for(k=0;k<n;k++)

          { j=1;

       key=arr[k] ;

       i = hasht(key); while
       (hash[i]!=-1)

       {

           i = rehashq(i,j); j++ ;

       }

       hash[i]=key ;

          }

          printf("\nThe elements in the array are: "); for
          (i=0;i<tsize;i++)

          {

       printf("\n Element at position %d: %d",i,hash[i]);

          }

          break ;


      }

         }while(op!=3);


         getch() ;

}
```

Output:

```
Enter the size of the hash table: 10


Enter the number of elements: 8

Enter Elements: 72 27 36 24 63 81 92 101


1.Linear Probing 2.Quadratic
Probing 3.Exit

Enter your option: 1


  The elements in the array are:
    Element at position 0: -1

    Element at position 1: 81

    Element at position 2: 72

    Element at position 3: 63
    Element at position 4: 24

    Element at position 5: 92

    Element at position 6: 36

    Element at position 7: 27

    Element at position 8: 101

    Element at position 9: -1


1.Linear Probing 2.Quadratic
Probing 3.Exit

Enter your option: 2


  The elements in the array are:
   Element at position 0: -1

   Element at position 1: 81

   Element at position 2: 72

   Element at position 3: 63

   Element at position 4: 24

   Element at position 5: 101

   Element at position 6: 36

   Element at position 7: 27

   Element at position 8: 92

   Element at position 9: -1


1.Linear Probing 2.Quadratic
Probing 3.Exit

Enter your option: 3
```

ii) KMP

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h> void
main()

{

      char string[100], matchcase[20];

       int i=0,j=0,index,flag;
```

```
clrscr();

printf("Enter string: ");
gets(string);

printf("Enter substring: "); gets(matchcase);

for (i=0;i<strlen(string)-strlen(matchcase)+1;i++)

{

        index = i; if(string[i]==matchcase[j])

        {

            do

            {

                    i++;
                    j++;

            } while(j!=strlen(matchcase)&&string[i]==matchcase[j]); if
            (j==strlen(matchcase))

            {

                                    printf("Match found from position %d to %d.\n",index+1,i); flag=1;


             }

            else

            {

                    i=index+1; j=0;



             }

        }

}
```

```
     if(flag!=1)

     printf("No substring match found in the string.\n");
    getch();

}
```

Ouput:

```
Enter string: programming Enter
substring: gram

Match found from position 4 to 7.
```