M.Kumarasamy
College of Engineering
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur, Tamilnadu.

B.E /B.TECH ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

LAB MANUAL
(REGULATION 2018)

18AIC203J - DATA STRUCTURES AND ALGORITHMS LABORATORY

PREPARED BY
MR.K. JEYA GANESH KUMAR
DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

LIST OF EXPERIMENTS

Ex.No	Name of the Programs
1(a)	Array Implementation of List ADT
1(b)	Linked list Implementation of List ADT
2(a)	Array Implementation of Stack ADT
2(b)	Linked list Implementation of Stack ADT
3(a)	Array Implementation of Queue ADT
3(b)	Linked list Implementation of Queue ADT
4	Implementation of Binary Search Tree.
5	Implementation of AVL Tree.
6	Implementation of Heaps
7	Graph representation and Traversal algorithms
8	Applications of graphs
9	Implementation of Searching and sorting algorithms
10	Hashing – any two collision techniques

EX.NO:1 (A) **ARRAY IMPLEMENTATION OF LIST ADT**
DATE:

AIM: To Write a C++ Program for array implementation of list ADT.

ALGORITHM:

```
STEP 1: Create nodes first, last, next, prev and cur then set the value as NULL.
STEP 2: Read the list operation type.
STEP 3: If operation type is create then  process the following steps.
            1.  Allocate memory for node cur.
            2.  Read data in cur's data area.
            3.  Assign cur node as NULL.
            4.  Assign first=last=cur.
STEP 4:    If operation type is Insert then process the following steps.
            1. Allocate memory for node cur.
            2.  Read data in cur's data area.
            3.  Read the position the Data to be insert.
            4. Availability of the position is true then assing cur's node as first and  first=cur.
            5.  If availability of position is false then do following steps.
            1. Assign next as cur and count as zero.
            2. Repeat the following steps until count less than postion.
            1 .Assign prev as next
            2. Next as prev of node.
            3. Add count by one.
            4. If prev as NULL then display the message INVALID POSITION.
            5. If prev not qual to NULL then do the following steps.
            1. Assign cur's node as prev's node.
            2. Assign prev's node as cur.
STEP 5:  If operation type is delete then do the following steps.
            1. Read the position .
            2. Check list is Empty .If it is true display the message List empty.
            3. If position is first.
            1. Assign cur as first.
            2. Assign First as first of node.
            3. Reallocate the cur from  memory.
            4. If position is last.
            1. Move the current node to prev.
            2. Cur's node as Null.
            3. Reallocate the Last from memory.
            4. Assign last as cur.
            5. If position is enter Mediate.
            1. Move the cur to required postion.
```

2. Move the Previous to cur's previous position
3. Move the Next to cur's Next position.
4. Now assign previous of node as next.
5. Reallocate the cur from memory.

STEP 6: If operation is traverse.

1. Assign current as first.
2. Repeat the following steps untill cur becomes NULL.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
void create();
void insert();
void deletion();
void search();
void display();
int a,b[20],n,d,e,f,i; void main()
{
int c;
char g='y';
clrscr();
do
{
cout<<"\n Main Menu";
cout<<"\n 1.Create \n 2.Delete \n 3.Search \n 4.insert \n 5.Display \n 6.Exit"; cout<<"\n enter your
choice\n";
cin>>c;
switch(c)
{
case 1: create(); break;
case 2: deletion(); break;
case 3: search(); break;
case 4: insert(); break;
case 5: display(); break;
case 6: exit(0); break;
default:
cout<<"The given number is not between 1-5\n";
}
cout<<"\nDo u want to continue \n";
cin>>g;
clrscr();
}
while(g=='y'|| g=='Y');
getch();
}
void create()
{ cout<<"\n Enter the number\n";
cin>>n;
for(i=0;i<n;i++)
{
cin>>b[i];
}}
void deletion()
{
cout<<"Enter the limit u want to delete \n";
cin>>d;
for(i=0;i<n;i++)
{
if(b[i]==d)
{
b[i]=0;
}}}
void search()
{
cout<<"Enter the limit \n";
cin>>e;
for(i=0;i<n;i++)
{
if(b[i]==e)
{
cout<<"Value found the position\n"<<b[i];
}}}
void insert()
{
cout<<"enter how many number u want to insert \n";
cin>>f;
for(i=0;i<f;i++)
{
cin>>b[n++];
}}
```

```
void display()
{
cout<<"\n\n\n";
for(i=0;i<n;i++)
{
cout<<"\n\n\n"<<b[i];
} }
```

OUTPUT:

```
        Main Menu
        1.Create
        2.Delete
        3.Search
        4.Insert
        5.Display
        6.Exit
        Enter your choice
        1
        Enter the number
        2
        3
        4
        Do u want to continue
```

RESULT:

Thus, the array implementation of list ADT program has been written and executed successfully.

EX.NO:1 (B) LINKED LIST IMPLEMENTATION OF LIST ADT
DATE:

AIM:

To Write a C++ Program for linked list implementation of list ADT.

ALGORITHM:

STEP 1: Create nodes first,last,next,prev and cur then set the value as NULL.

STEP 2: Read the list operation type.

STEP 3: If operation type is create then process the following steps.

1. Allocate memory for node cur.
2. Read data in cur's data area.
3. Assign cur link as NULL.
4. Assign first=last=cur.

STEP 4: If operation type is Insert then process the following steps.

1. Allocate memory for node cur.
2. Read data in cur's data area.
3. Read the position the Data to be inserting.
4. Availability of the position is true then assign cur's link as first and first=cur.
5. If availability of position is false then do following steps.
1. Assign next as cur and count as zero.
2. Repeat the following steps until count less than position.
1 .Assign prev as next
2. Next as prev of link.
3. Add count by one.
4. If prev as NULL then display the message INVALID POSITION.
5. If prev not qual to NULL then do the following steps.
1. Assign cur's link as prev's link.
2. Assign prev's link as cur.

STEP 5: If operation type is delete then do the following steps.

1. Read the position .
2. Check list is Empty .If it is true display the message List empty.
3. If position is first.
1. Assign cur as first.
2. Assign First as first of link.
3. Reallocate the cur from memory.
4. If position is last.
1. Move the current node to prev.

2. cur's link as Null.
3. Reallocate the Last from memory.
4. Assign last as cur.
5. If position is enter Mediate.
1. Move the cur to required position.
2. Move the Previous to cur's previous position
3. Move the Next to cur's Next position.
4. Now assign previous of link as next.
5. Reallocate the cur from memory.

STEP 6: If operation is traverse.

1. Assign current as first.
2. Repeat the following steps until cur becomes NULL.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
int  element;
struct node *next;
}*list=NULL,*p;
struct node *find(int);
struct node *findprevious(int);
void insert(int X);
void deletion(int X);
void display();
void main()
{
int data,ch;
clrscr();
cout<<"1.Insert\n2.Deletion\n3.Display\n4.Exit";
do
{
cout<<"\nenter your choice";
cin>>ch;
switch(ch)
{
case 1:
cout<<"enter the element to insert";
cin>>data;
insert(data);
break;
case 2:
cout<<"enter the element to delete";
cin>>data;
deletion(data);
break;
case 3:
display();
break;
case 4:
exit(0);
}
}while(ch<4);
getch();
}
void insert(int X)
{
struct node *newnode;
int pos;
newnode=(struct node*)malloc(sizeof(struct node));
newnode->element=X;
if(list->next==NULL)
{

list->next=newnode;
newnode->next=NULL;
}
else
{
cout<<"\n enter the value after which the element to be inserted";
cin>>pos;
p=find(pos);
newnode->next=p->next;
p->next=newnode;
}
}
struct node *find(int s)
{
p=list->next;
while(p!=NULL&&p->element!=s)
p=p->next;
return p;
}
```

```
void deletion(int X)
{
struct node *temp;
temp=(struct node*)malloc(sizeof(struct node));
p=findprevious(X);
if(p->next!=NULL)
{
temp=p->next;
p->next=temp->next;
cout<<"\n the deleted element is"<<temp->element;
free(temp);
}
else
cout<<"\n element not found in the list";
}
struct node *findprevious(int s)
{
p=list;
while(p->next!=NULL&&p->next->element!=s)
p=p->next;
return p;
}
void display()
{
if(list->next==NULL)
cout<<"list is empty";
else
{
p=list->next;
cout<<"\n the contents of the list are:\n";
while(p!=NULL)
{
cout<<p->element;
p=p->next;
}
cout<<"null";
}
}
```

OUTPUT:

```
1. Insert 2.deletion 3.display 4.exit
Enter your choice: 1
Enter the element to insert: 12
Enter your choice: 1
Enter the element to insert: 13
Enter the value after which the element to be inserted: 12
Enter your choice: 1
Enter the element to insert: 14
Enter the value after which the element to be inserted: 12
Enter your choice: 3
The contents of the list are:
12->14->13->NULL
Enter your choice: 2
Enter the element to delete: 14
The deleted element is 14
Enter your choice: 3
The contents of the list are:
12->13->NULL
Enter your choice: 2
Enter the element to delete: 13
The deleted element is 13
Enter your choice: 3
The contents of list are:
12->NULL
Enter your choice: 4
```

RESULT:

Thus, the linked list implementation of list ADT for singly linked list program has been written and executed successfully.

EX: NO: 2(A) **ARRAY IMPLEMENTATION OF STACK ADT**
DATE:

AIM:

To Write a C++ Program to Stack ADT implementation using array

ALGORITHM:

STEP 1: Define a stack size.
STEP 2: Read the stack operation.
STEP 3: Read the stack element.
STEP 4: Check the stack operation is Push or Pop.
STEP 5: If operation is push then check the stack status.

i. If stack status is over flow we can't push the element in to stack. ii. Otherwise we can add the data into stack .
iii. Move top to next position.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
//using namespace std;
class stack
{
int stk[5];
int top;
public:
stack()
{
top=-1;
}
void push(int x)
{
if(top >  4)
{
cout <<"stack over flow";
return;
}
stk[++top]=x;
cout <<"inserted" <<x;
}
void pop()
{
if(top <0)
{
cout <<"stack under flow";
return;
}
cout <<"deleted" <<stk[top--];
}
void display()
{
if(top<0)
{
cout <<" stack empty";
return;
}
for(int i=top;i>=0;i--)
cout <<stk[i] <<" ";
 }
};

main()
{
Clrscr();
int ch;
stack st;
while(1)
{
cout <<"\n1.push 2.pop 3.display 4.exit\nEnter ur choice"; cin >> ch;
switch(ch)
{
case 1:  cout <<"enter the element";
           cin >> ch;
st.push(ch);
break;
case 2:  st.pop();  break; case 3:  st.display();break; case 4:  exit(0);
}
}
return (0);
}
```

OUTPUT

```
1.push 2.pop 3.display 4.exit Enter ur choice2
stack under flow
1.push 2.pop 3.display 4.exit Enter ur choice1
enter the element2 inserted2
1.push 2.pop 3.display 4.exit Enter ur choice1
enter the element3 inserted3
1.push 2.pop 3.display 4.exit Enter ur choice2
deleted3
1.push 2.pop 3.display 4.exit Enter ur choice1
enter the element5 inserted5
1.push 2.pop 3.display 4.exit Enter ur choice3
5 2
1.push 2.pop 3.display 4.exit Enter ur choice4
```

RESULT:

Thus, the stack ADT- array implementation program has been written and executed successfully.

EX: NO: 2(B) LINKED LIST IMPLEMENTATION OF STACK ADT

DATE:

AIM:

To Write a C++ Program to Stack ADT implementation using linked list

ALGORITHM:

```
STEP 1: Create a list.
        i) Create a new empty node top.
          ii) Read the stack element and store it in top's data area.
          iii) Assign top's link part as NULL (i.e. top->link=NULL).
          iv) Assign temp as top (i.e. temp=top).
STEP 2: Read next stack operation.
          i) If it is Create then go to step1.
          ii) If it is Push then it process following steps
          a) Check Main memory for node creation.
          b) Create a new node top.
          c) Read the stack element and store it in top's data area.
          d) Assign top's link part as temp (i.e. top->link=temp).
          e) Assign temp as top (i.e. temp=top).
          iii) If it is pop then it process following steps
          a) If top is NULL then display stack is empty.
          b) Otherwise assign top as temp (i.e. top=temp, bring the top to top position)
          c)  Assign temp as temp's link. (i.e. temp=temp->link, bring  the temp to top's
            previous position).
          d) Delete top from memory.
          iv) If it is traverse then process the following steps
          a) Bring the top to stack's top position(i.e. top=temp)
          b) Repeat until top becomes NULL
          i) Display the top's data.
          ii) Assign top as top's link (top=top->link).
```

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
//using namespace std;
class node
{
public:
class node *next; int data;
};
```

```
class stack : public node
{
node *head;
int tos;
public:
stack()
{
tos=-1;
}
void push(int x)
{
if (tos < 0 )
{
head =new node;
head->next=NULL; head->data=x;
tos ++;
}
else
{
node *temp,*temp1; temp=head;
if(tos >= 4)
{
cout <<"stack over flow";
return;
}
tos++;
while(temp->next != NULL)
temp=temp->next;
temp1=new node;
temp->next=temp1;
temp1->next=NULL;
temp1->data=x;
 }
}
void display()
{
node *temp;
temp=head;
if (tos < 0)
{
cout <<" stack under flow";
return;
}
while(temp != NULL)
{
cout <<temp->data<< " ";
temp=temp->next;
}
}
void pop()
{
node *temp;
temp=head;
if( tos < 0 )
{
cout <<"stack under flow";
return;
}
tos--;
while(temp->next->next!=NULL)
{
temp=temp->next;
}
temp->next=NULL;
}
};
main()
{
Clrscr();
stack s1;
int ch;
while(1)
{
cout <<"\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n enter ur choice:"; cin >> ch;
switch(ch)
{
case 1:  cout <<"\n enter a element";
cin >> ch;
s1.push(ch);
break;
case 2:  s1.pop();break;
case 3:  s1.display();
break;
case 4:  exit(0);
}
}
```

```
return (0);
}
```

OUTPUT

```
1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:1
enter a element23
1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:1
enter a element67
1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:3
23 67
1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:2
1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:3
23
1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:2
1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:2
stack under flow
1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:4
```

RESULT:

Thus, the stack ADT- linked list implementation program has been written and executed successfully.

EX: NO: 3(A) **ARRAY IMPLEMENTATION OF QUEUE ADT**
DATE:

AIM:

To Write a C++ Program to Queue ADT implementation using array

ALGORITHM

```
STEP 1: Initialize the queue variables front =0 and rear = -1
STEP 2: Read the queue operation type.
STEP 3:  Check the queue operations status.
        i) If it is Insertion then do the following steps
        1. Check rear < queue_size is true increment the rear by one and read the queue
           element and also display queue. Otherwise display the queue is full.
        2. Go to step2.
        ii) If it is deletion then do the following steps
            1. Check rear< front is true then display the queue is empty.
            2. Move the elements to one step forward (i.e. move to previous index ).
            3. Decreases the rear value by one (rear=rear-1).
            4. Display queue
            5. Go to step2.
```

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
//using namespace std;
class queue
{
int queue1[5];
int rear,front; public:
queue()
{
rear=-1;
front=-1; }
void insert(int x)
{
if(rear >  4)
{
cout <<"queue over flow"; front=rear=-1;
return;
}
```

```
queue1[++rear]=x;
cout <<"inserted" <<x;
}
void delet()
{
if(front==rear)
{
cout <<"queue under flow";
return;
}
cout <<"deleted" <<queue1[++front];
}
void display()
{
if(rear==front)
{
cout <<" queue empty";
return;
}
for(int i=front+1;i<=rear;i++)
cout <<queue1[i]<<" ";
 }
};
main()
{
Clrscr();
int ch;
queue qu;

while(1)
{
cout <<"\n1.insert  2.delet  3.display  4.exit\nEnter ur choice";
cin >> ch;
switch(ch)
{
case 1:    cout <<"enter the element";
cin >> ch;
qu.insert(ch);
break;
case 2:  qu.delet();
 break;
case 3:  qu.display();
break;
case 4: exit(0);
}
}
return (0);
}
```

OUTPUT :

```
        1.insert 2.delet 3.display 4.exit Enter ur choice1
        enter the element21 inserted21

        1.insert 2.delet 3.display 4.exit Enter ur choice1
        enter the element22 inserted22

        1.insert 2.delet 3.display 4.exit Enter ur choice1
        enter the element16 inserted16

        1.insert 2.delet 3.display 4.exit Enter ur choice3
        21 22 16

        1.insert 2.delet 3.display 4.exit Enter ur choice2
        deleted21

        1.insert 2.delet 3.display 4.exit Enter ur choice3
        22 16

        1.insert 2.delet 3.display 4.exit Enter ur choice
```

RESULT:

Thus, the queue ADT- array implementation program has been written and executed successfully

EX: NO: 3(B) LINKED LIST IMPLEMENTATION OF QUEUE ADT
DATE:

AIM:

To Write a C++ Program to Queue ADT implementation using linked list

ALGORITHM

STEP 1: Initialize the queue variables front =0 and rear = -1
STEP 2: Read the queue operation type.
STEP 3: Check the queue operations status.

1. If it is Insertion then do the following steps
2. Check rear not equal to null is true increment the rear by one and read thequeue element and also display queue. otherwise display the queue is full.
3. Go to step2.
4. If it is deletion then do the following steps
5. Check rear< front is true then display the queue is empty.
6. Move the elements to one step forward (i.e. move to previous index).
7. Decreases the rear value by one (rear=rear-1).
8. Display queue
9. Go to step2.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
//using namespace std;
class node
{
public:
class node *next; int data;
};
class queue : public node
{
node *head;
int front,rare;
public:
queue()
{
front=-1;
rare=-1;
}
```

```
void push(int x)
{
if (rare < 0 )
{
head =new node;
head->next=NULL; head->data=x;
rare ++;
}
else
{
node *temp,*temp1;
temp=head;
if(rare >= 4)
{
cout <<"queue over flow";
return;
}
rare++;
while(temp->next != NULL)
temp=temp->next;
temp1=new node;
temp->next=temp1;
temp1->next=NULL;
temp1->data=x;
}}
void display()
{
node *temp;
temp=head;
if (rare < 0)
{
cout <<" queue under flow";
return;
}
while(temp != NULL)
{
cout <<temp->data<< " "; temp=temp->next;
}
}
void pop()
{
```

```
node *temp;
temp=head;
if( rare < 0)
{
cout <<"queue under flow";
return;
}
if(front == rare)
{
front = rare =-1;
head=NULL;  return;
}
front++;
head=head->next;
}
};
main()
{
Clrscr();
queue s1;
int ch;
while(1)
{
cout <<"\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n enter ru choice:"; cin >> ch;
switch(ch)
{
case 1:
cout <<"\n enter a element"; cin >> ch;
s1.push(ch); break;
case 2: s1.pop();break;
case 3: s1.display();break; case 4: exit(0);
}
}
return (0);
}
```

OUTPUT

```
        1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:1
        enter a element23

        1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:1
        enter a element54
```

```
1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:3
23 54
1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:2

1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:2

1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:2
queue under flow

1.PUSH 2.POP 3.DISPLAY 4.EXIT enter ru choice:4
```

RESULT

Thus, the queue ADT- linked list implementation program has been written and executed successfully.

EX: NO: 4 IMPLEMENTATION OF BINARY SEARCH TREE

DATE:

AIM:

To Write a C++ Program to to perform Insert, Delete, Search an element into a binary search tree

ALGORITHM:

STEP1: Create a new instance of Binary Tree.
STEP2: Create a new instance of Tree Test and select Binary Tree instance in the object
STEP3: Bench as the parameter in the constructor.
STEP4: Call the populate method of Tree Test instance.
STEP5: Inspect the Binary Tree. Its attributes are a left sub tree, a right sub tree and a data

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
//using namespace std;
void insert(int,int );
void delte(int);
void display(int);
int search(int);
int search1(int,int);
int tree[40],t=1,s,x,i;
main()
{
clrscr();
int ch,y;
for(i=1;i<40;i++) tree[i]=-1;
while(1)
{
cout <<"1.INSERT\n2.DELETE\n3.DISPLAY\n4.SEARCH\n5.EXIT\nEnter your choice:";
cin >> ch;
switch(ch)
{
case 1:
cout <<"enter the element to insert"; cin >> ch;
insert(1,ch);
break;
case 2:
```

```
cout <<"enter the element to delete"; cin >>x;
y=search(1);
if(y!=-1) delte(y);
else cout<<"no such element in tree";
break;
case 3:
display(1);
cout<<"\n";
for(int i=0;i<=32;i++) cout <<i;
cout <<"\n";
break;
case 4:
cout <<"enter the element to search:"; cin >> x;
y=search(1);
if(y == -1) cout <<"no such element in tree";
else cout <<x << "is in" <<y <<"position";
break;
case 5:
exit(0);
}
}
}
void insert(int s,int ch )
{
int x;
if(t==1)
{
tree[t++]=ch;
return;
}
x=search1(s,ch);
if(tree[x]>ch)
tree[2*x]=ch;
else
tree[2*x+1]=ch;
t++;
}
void delte(int x)
{
if( tree[2*x]==-1 && tree[2*x+1]==-1)
tree[x]=-1;
```

```
else if(tree[2*x]==-1)
{
tree[x]=tree[2*x+1];
tree[2*x+1]=-1;
}
else if(tree[2*x+1]==-1)
{
tree[x]=tree[2*x];
tree[2*x]=-1;
}
else
{
tree[x]=tree[2*x]; delte(2*x);
}
t--;
}

int search(int s)
{
if(t==1)
{
cout <<"no element in tree"; return -1;
}
if(tree[s]==-1)
return tree[s];
if(tree[s]>x)
search(2*s);
else if(tree[s]<x)
search(2*s+1);
else
return s;
}
void display(int s)
{
if(t==1)
{cout <<"no element in tree:"; return;}
for(int i=1;i<40;i++) if(tree[i]==-1)
cout <<" ";
else cout <<tree[i]; return ;
}
int search1(int s,int ch)
{
```

```
if(t==1)
{
cout <<"no element in tree"; return -1;
}
if(tree[s]==-1)
return s/2;
if(tree[s] > ch)
search1(2*s,ch);
else search1(2*s+1,ch);
}
```

OUTPUT

```
1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.EXIT
Enter your choice:3

no element in tree:
0123456789011121314151617181920212223242526272829303132

1. INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.EXIT
Enter your choice:1


Enter the element to insert 10
1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.EXIT
Enter your choice: 4

Enter the element to search: 10
10 is in 1 position

1.INSERT
2.DELETE
```

```
3.DISPLAY
4.SEARCH
5.EXIT
Enter your choice:5
```

RESULT:

Thus, the C++ program to perform Insert, Delete, Search an element into a binary tree has been written and executed successfully.

EX.NO:5 **IMPLEMENTATION OF AVL TREE**
DATE:

AIM:

To Write a C++ Program to implement an AVL Tree.

ALGORITHM:

STEP 1: Insert the node in the AVL tree using the same insertion algorithm of BST.
STEP 2: Once the node is added, the balance factor of each node is updated.
STEP 3: Now check if any node violates the range of the balance factor if the balance factor is violated, then perform rotations using the below case.
STEP 4: For deletion, Find the element in the tree.
STEP 5: Delete the node, as per the BST Deletion.

SOURCE CODE:

```
#include<iostream.h>
#include<cstdio.h>
#include<sstream.h>
#include<algorithm.h>
#define pow2(n) (1 << (n))
using namespace std;
struct avl {
  int d;
  struct avl *l;
  struct avl *r;
}*r;
class avl_tree {
  public:
    int height(avl *);
    int difference(avl *);
    avl *rr_rotat(avl *);
    avl *ll_rotat(avl *);
    avl *lr_rotat(avl*);
    avl *rl_rotat(avl *);
    avl * balance(avl *);
    avl * insert(avl*, int);
    void show(avl*, int);
    void inorder(avl *);
    void preorder(avl *);
    void postorder(avl*);
    avl_tree() {
      r = NULL;
    }
};
int avl_tree::height(avl *t) {
  int h = 0;
  if (t != NULL) {
    int l_height = height(t->l);
    int r_height = height(t->r);
    int max_height = max(l_height, r_height);
    h = max_height + 1;
  }
  return h;
}
int avl_tree::difference(avl *t) {
  int l_height = height(t->l);
  int r_height = height(t->r);
  int b_factor = l_height - r_height;
  return b_factor;
}
avl *avl_tree::rr_rotat(avl *parent) {
  avl *t;
  t = parent->r;
  parent->r = t->l;
  t->l = parent;
  cout<<"Right-Right Rotation";
  return t;
}
avl *avl_tree::ll_rotat(avl *parent) {
  avl *t;
  t = parent->l;
  parent->l = t->r;
  t->r = parent;
  cout<<"Left-Left Rotation";
  return t;
}
avl *avl_tree::lr_rotat(avl *parent) {
  avl *t;
  t = parent->l;
  parent->l = rr_rotat(t);
  cout<<"Left-Right Rotation";
  return ll_rotat(parent);
```

```
}
avl *avl_tree::rl_rotat(avl *parent) {
  avl *t;
  t = parent->r;
  parent->r = ll_rotat(t);
  cout<<"Right-Left Rotation";
  return rr_rotat(parent);
}
avl *avl_tree::balance(avl *t) {
  int bal_factor = difference(t);
  if (bal_factor > 1) {
    if (difference(t->l) > 0)
      t = ll_rotat(t);
    else
      t = lr_rotat(t);
  } else if (bal_factor < -1) {
    if (difference(t->r) > 0)
      t = rl_rotat(t);
    else
      t = rr_rotat(t);
  }
  return t;
}
avl *avl_tree::insert(avl *r, int v) {
  if (r == NULL) {
    r = new avl;
    r->d = v;
    r->l = NULL;
    r->r = NULL;
    return r;
  } else if (v< r->d) {
    r->l = insert(r->l, v);
    r = balance(r);
  } else if (v >= r->d) {
    r->r = insert(r->r, v);
    r = balance(r);
  } return r;
}
void avl_tree::show(avl *p, int l) {
  int i;
  if (p != NULL) {
    show(p->r, l+ 1);
    cout<<" ";
    if (p == r)
      cout << "Root -> ";
    for (i = 0; i < l&& p != r; i++)
      cout << " ";
      cout << p->d;
      show(p->l, l + 1);
  }
}
void avl_tree::inorder(avl *t) {
  if (t == NULL)
    return;
    inorder(t->l);
    cout << t->d << " ";
    inorder(t->r);
}
void avl_tree::preorder(avl *t) {
  if (t == NULL)
    return;
    cout << t->d << " ";
    preorder(t->l);
    preorder(t->r);
}
void avl_tree::postorder(avl *t) {
  if (t == NULL)
    return;
    postorder(t ->l);
    postorder(t ->r);
    cout << t->d << " ";
}
int main() {
  int c, i;
  avl_tree avl;
  while (1) {
    cout << "1.Insert Element into the tree" << endl;
    cout << "2.show Balanced AVL Tree" << endl;
    cout << "3.InOrder traversal" << endl;
    cout << "4.PreOrder traversal" << endl;
    cout << "5.PostOrder traversal" << endl;
    cout << "6.Exit" << endl;
    cout << "Enter your Choice: ";
    cin >> c;
    switch (c) {
      case 1:
          cout << "Enter value to be inserted: ";
          cin >> i;
          r = avl.insert(r, i);
        break;
        case 2:
          if (r == NULL) {
            cout << "Tree is Empty" << endl;
            continue;
          }
          cout << "Balanced AVL Tree:" << endl;
          avl.show(r, 1);
          cout<<endl;
        break;
        case 3:
          cout << "Inorder Traversal:" << endl;
          avl.inorder(r);
          cout << endl;
        break;
        case 4:
          cout << "Preorder Traversal:" << endl;
          avl.preorder(r);
          cout << endl;
        break;
        case 5:
          cout << "Postorder Traversal:" << endl;
          avl.postorder(r);
          cout << endl;
        break;
        case 6:
          exit(1);
        break;
        default:
          cout << "Wrong Choice" << endl;
      }
    }
    return 0;
}
```

OUTPUT

```
 1.Insert Element into the tree
 2.show Balanced AVL Tree
 3.InOrder traversal
 4.PreOrder traversal
 5.PostOrder traversal
```

```
6.Exit
Enter your Choice: 1
Enter value to be inserted: 13
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 1
Enter value to be inserted: 10
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 1
Enter value to be inserted: 15
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 1
Enter value to be inserted: 5
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 1
Enter value to be inserted: 11
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 1
Enter value to be inserted: 4
```

```
Left-Left Rotation1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 1
Enter value to be inserted: 8
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 1
Enter value to be inserted: 16
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 3
Inorder Traversal:
4 5 8 10 11 13 15 16
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 4
Preorder Traversal:
10 5 4 8 13 11 15 16
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 5
Postorder Traversal:
4 8 5 11 16 15 13 10
```

```
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 1
Enter value to be inserted: 14
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 1
Enter value to be inserted: 3
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 1
Enter value to be inserted: 7
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 1
Enter value to be inserted: 9
1.Insert Element into the tree
2.show Balanced AVL Tree
3.InOrder traversal
4.PreOrder traversal
5.PostOrder traversal
6.Exit
Enter your Choice: 1
Enter value to be inserted: 52
Right-Right Rotation
1.Insert Element into the tree
2.show Balanced AVL Tree
```

```
 3.InOrder traversal
 4.PreOrder traversal
 5.PostOrder traversal
 6.Exit
Enter your Choice: 6
```

RESULT:

Thus, the C++ program to perform Insert, Delete, Search an element into a AVL tree has been written and executed successfully

EX.NO:6 **IMPLEMENTATION OF HEAPS**
DATE:

AIM:

To Write a C++ Program to implement heaps.

ALGORITHM:

STEP 1: Let the input array be Initial Array
STEP 2: Create a complete binary tree from the array Complete binary tree
STEP 3: Start from the first index of non-leaf node whose index is given by n/2 - 1 . Start from the first on leaf node
STEP 4: Set current element i as largest.
STEP 5: The index of left child is given by 2i + 1 and the right child is given by 2i+2.
STEP 6: Swap largest with currentElement Swap if necessary
STEP 7: Repeat steps 3-7 until the subtrees are also heapified.

SOURCE CODE:

```
#include <iostream>
#include <cstdlib>
#include <vector>
#include <iterator>
using namespace std;
class BHeap {
  private:
  vector <int> heap;
  int l(int parent);
  int r(int parent);
  int par(int child);
  void heapifyup(int index);
  void heapifydown(int index);
  public:
    BHeap() {}
    void Insert(int element);
    void DeleteMin();
    int ExtractMin();
    void showHeap();
    int Size();
};

int main() {
  BHeap h;
  while (1) {
    cout<<"1.Insert Element"<<endl;
    cout<<"2.Delete Minimum Element"<<endl;
    cout<<"3.Extract Minimum Element"<<endl;
    cout<<"4.Show Heap"<<endl;
    cout<<"5.Exit"<<endl;
    int c, e;
    cout<<"Enter your choice: ";
    cin>>c;
    switch(c) {
      case 1:
        cout<<"Enter the element to be inserted: ";
        cin>>e;
        h.Insert(e);
      break;
      case 2:
        h.DeleteMin();
      break;
      case 3:
        if (h.ExtractMin() == -1) {
          cout<<"Heap is Empty"<<endl;
        }
        else
        cout<<"Minimum Element: "<<h.ExtractMin()<<endl;
      break;
      case 4:
        cout<<"Displaying elements of Hwap: ";
        h.showHeap();
      break;
      case 5:
        exit(1);
        default:
        cout<<"Enter Correct Choice"<<endl;
    }
  }
  return 0;
}
int BHeap::Size() {
  return heap.size();
}
void BHeap::Insert(int ele) {
  heap.push_back(ele);

  heapifyup(heap.size() -1);
```

```
}
void BHeap::DeleteMin() {
  if (heap.size() == 0) {
    cout<<"Heap is Empty"<<endl;
    return;
  }
  heap[0] = heap.at(heap.size() - 1);
  heap.pop_back();
  heapifydown(0);
  cout<<"Element Deleted"<<endl;
}
int BHeap::ExtractMin() {
  if (heap.size() == 0) {
    return -1;
  }
  else
  return heap.front();
}
void BHeap::showHeap() {
  vector <int>::iterator pos = heap.begin();
  cout<<"Heap --> ";
  while (pos != heap.end()) {
    cout<<*pos<<" ";
    pos++;
  }
  cout<<endl;
}
int BHeap::l(int parent) {
  int l = 2 * parent + 1;
  if (l < heap.size())
    return l;
  else
    return -1;
}
int BHeap::r(int parent) {
  int r = 2 * parent + 2;
  if (r < heap.size())
    return r;
  else
    return -1;
}
int BHeap::par(int child) {
  int p = (child - 1)/2;

  if (child == 0)
```

```
    return -1;
  else
    return p;
}
void BHeap::heapifyup(int in) {
  if (in >= 0 && par(in) >= 0 && heap[par(in)] > heap[in]) {
    int temp = heap[in];
    heap[in] = heap[par(in)];
    heap[par(in)] = temp;
    heapifyup(par(in));
  }
}
void BHeap::heapifydown(int in) {
  int child = l(in);
  int child1 = r(in);
  if (child >= 0 && child1 >= 0 && heap[child] > heap[child1]) {
    child = child1;
  }
  if (child > 0 && heap[in] > heap[child]) {
    int t = heap[in];
    heap[in] = heap[child];
    heap[child] = t;
    heapifydown(child);
  }
}
```

OUTPUT

```
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Show Heap
5.Exit
Enter your choice: 1
Enter the element to be inserted: 2
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Show Heap
5.Exit
Enter your choice: 1
Enter the element to be inserted: 3
1.Insert Element
2.Delete Minimum Element

3.Extract Minimum Element
```

```
4.Show Heap
5.Exit

Enter your choice: 1
Enter the element to be inserted: 7

1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Show Heap
5.Exit
Enter your choice: 1
Enter the element to be inserted: 6
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Show Heap
5.Exit
Enter your choice: 4
Displaying elements of Hwap: Heap --> 2 3 7 6
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Show Heap
5.Exit
Enter your choice: 3
Minimum Element: 2
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Show Heap
5.Exit
Enter your choice: 3
Minimum Element: 2
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Show Heap
5.Exit
Enter your choice: 2
Element Deleted
1.Insert Element
2.Delete Minimum Element
```

```
3.Extract Minimum Element
4.Show Heap
5.Exit
Enter your choice: 4

Displaying elements of Hwap: Heap --> 3 6 7
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Show Heap
5.Exit
Enter your
```

RESULT:

Thus, the heap tree program has been written and executed successfully.

EX.NO:7 GRAPH REPRESENTATION AND TRAVERSAL ALGORITHMS

DATE:

AIM:

To Write a C++ Program to perform Graph representation and Graph traversal algorithms.

ALGORITHM:

DFS traversal

STEP 1 - Define a Stack of size total number of vertices in the graph.
STEP 2 - Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.
STEP 3 - Visit any one of the non-visited adjacent vertices of a vertex which is at the top of stack and push it on to the stack.
STEP 4 - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
STEP 5 - When there is no new vertex to visit then use back tracking and pop one vertex from the stack.
STEP 6 - Repeat steps 3, 4 and 5 until stack becomes Empty.
STEP 7 - When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

BFS traversal

STEP 1 - Define a Queue of size total number of vertices in the graph.
STEP 2 - Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.
STEP 3 - Visit all the non-visited adjacent vertices of the vertex which is at front of the Queue and insert them into the Queue.
STEP 4 - When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.
STEP 5 - Repeat steps 3 and 4 until queue becomes empty.
STEP 6 - When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

SOURCE CODE:

Program 1 (BFS)

```
#include <iostream.h>
#include <list.h>
#include <memory.h>
```

```
class Graph
{
   int _V;
   bool _directed;
   std::unique_ptr< std::list<int> > adj;

public:
   Graph(int V, bool directed);
   void AddEdge(int v, int w);
   void BreadthFirstSearch(int s);
};

Graph::Graph(int V, bool directed) : adj(new std::list<int>[V])
{
   _V = V;
   _directed = directed;
}

void Graph::AddEdge(int v, int w)
{
   std::list<int>* adjacency = adj.get();
   adjacency[v].push_back(w);

   if (!_directed)
   {
      adjacency[w].push_back(v);
   }
}

void Graph::BreadthFirstSearch(int s)
{
   bool *visited = new bool[_V];
   for(int i = 0; i < _V; i++)
      visited[i] = false;

   std::list<int> queue;

   visited[s] = true;
   queue.push_back(s);


   std::list<int>::iterator i;

   while(!queue.empty())
   {
      s = queue.front();
      std::cout << s << " ";
      queue.pop_front();


      for(i = (adj.get())[s].begin(); i != (adj.get())[s].end(); ++i)
      {
          if(!visited[*i])
          {
              visited[*i] = true;
              queue.push_back(*i);
          }
      }
   }
}

int main()
{
   Graph g(7, true);
   g.AddEdge(0, 1);
   g.AddEdge(0, 2);
   g.AddEdge(0, 3);
   g.AddEdge(1, 0);
   g.AddEdge(1, 5);
   g.AddEdge(2, 5);
          g.AddEdge(3, 0);
          g.AddEdge(3, 4);
          g.AddEdge(4, 6);
          g.AddEdge(5, 1);
          g.AddEdge(6, 5);

   std::cout << "Breadth First Traversal from vertex 2:\n";
   g.BreadthFirstSearch(2);

   return 0;
}
```

Program 2 (DFS)

```
#include <iostream>
#include <list>
#include <memory>

class Graph
{
private:
    int _V;
    bool _directed;
    std::unique_ptr< std::list<int> > adj;
            void DFSUtil(int v, bool visited[]);

public:
    Graph(int V, bool directed);
    void AddEdge(int v, int w);
    void DepthFirstSearch(int s);
};

Graph::Graph(int V, bool directed) : adj(new std::list<int>[V])
{
    _V = V;
    _directed = directed;
}

void Graph::AddEdge(int v, int w)
{
    std::list<int>* adjacency = adj.get();
    adjacency[v].push_back(w);

    if (!_directed)
    {
        adjacency[w].push_back(v);
    }
}

void Graph::DFSUtil(int v, bool visited[])
{

    visited[v] = true;
    std::cout << v << " ";
    std::list<int>::iterator i;
```

```
  for (i = (adj.get())[v].begin(); i != (adj.get())[v].end(); ++i)
    if (!visited[*i])
      DFSUtil(*i, visited);
}

void Graph::DepthFirstSearch(int v)
{

      std::unique_ptr<bool[]> visited(new bool[_V]);

  for (int i = 0; i < _V; i++)
    visited[i] = false;

      DFSUtil(v, visited.get());
}

int main()
{
  Graph g(7, true);
  g.AddEdge(0, 1);
  g.AddEdge(0, 2);
  g.AddEdge(0, 3);
  g.AddEdge(1, 0);
  g.AddEdge(1, 5);
  g.AddEdge(2, 5);
  g.AddEdge(3, 0);
  g.AddEdge(3, 4);
  g.AddEdge(4, 6);
  g.AddEdge(5, 1);
  g.AddEdge(6, 5);

  std::cout << "Depth First Traversal starting from vertex 2:\n";
      g.DepthFirstSearch(2);

  return 0;
}
```

OUTPUT:

Program 1 (BFS)

```
Breadth first Traversal from vertex 2:
2       5       1       0       3       4       6
```

Program 2 (DFS)

```
Depth first Traversal starting from vertex 2:
2       5       1       0       3       4       6
```

RESULT:

Thus, the programs for Graph Traversal algorithms have been written and executed successfully.

EX: No: 8 **APPLICATIONS OF GRAPHS**

DATE:

AIM:

To write a C++ program to find the shortest distances in a Graph

ALGORITHM:

STEP 1: Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in the shortest-path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.
STEP 2: Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
STEP 3: While *sptSet* doesn't include all vertices
STEP 4: Pick a vertex u which is not there in *sptSet* and has a minimum distance value
STEP 5: Include u to *sptSet*.
STEP 6: Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices.
STEP 7: Stop

SOURCE CODE:

```
#include <iostream.h>
#include <limits.h>

#define V9

int minDistance(int dist[], bool sptSet[])
{

  int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++)
       if (sptSet[v] == false && dist[v] <= min)
           min = dist[v], min_index = v;

   return min_index;
}

void printSolution(int dist[])
{
   cout <<"Vertex \t Distance from Source" << endl;
   for (int i = 0; i < V; i++)
       cout  << i << " \t\t"<<dist[i]<< endl;
}

void dijkstra(int graph[V][V], int src)
{
   int dist[V];

   bool sptSet[V];
   for (int i = 0; i < V; i++)
       dist[i] = INT_MAX, sptSet[i] = false;

  dist[src] = 0;

   for (int count = 0; count < V - 1; count++) {
       int u = minDistance(dist, sptSet);

          sptSet[u] = true;

       for (int v = 0; v < V; v++)

            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
               && dist[u] + graph[u][v] < dist[v])
               dist[v] = dist[u] + graph[u][v];
   }

 printSolution(dist);
}


int main()
{


   int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                       { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                       { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                       { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                       { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                       { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                       { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                       { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                       { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
   dijkstra(graph, 0);

   return 0;
}
  }
```

OUTPUT:

```
Vertex   Distance from Source

0              0

1              4

2              12

3              19

4              21

5              11

6              9

7              8

8              14
```

RESULT:

Thus, the Graph application program has been written and executed successfully.

EX: No: 8(a) **IMPLEMENTATION OF SEARCHING ALGORITHMS**

DATE:

AIM:

To implement a c++ program for searching algorithms.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the class name as fn with data members and member functions.
STEP 3: Read the choice from the user.
STEP 4: Choice=1 then go to the step 5.
STEP 5: The function area() to find area of circle with one integer argument.
STEP 6: Choice=2 then go to the step 7.
STEP 7: The function area() to find area of rectangle with two integer argument.
STEP 8: Choice=3 then go to the step 9.
STEP 9: The function area() to find area of triangle with three arguments, two as Integer and one as float.
STEP 10: Choice=4 then stop the program.

SOURCE CODE:

Program 1 (Linear search)

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
   int myarray[10] = {21,43,23,54,75,13,5,8,25,10};
    int key,loc;
    cout<<"The input array is"<<endl;
    for(int i=0;i<10;i++){
        cout<<myarray[i]<<" ";
    }
    cout<<endl;
    cout<<"Enter the key to be searched : "; cin>>key;
    for (int i = 0; i< 10; i++)
    {
        if(myarray[i] == key)
        {
            loc = i+1;
            break;
        }
        else
        loc = 0;
    }
    if(loc != 0)
    {
        cout<<"Key found at position "<<loc<<" in the array";
    }
    else
    {
        cout<<"Could not find given key in the array";
    }
```

Program 2 (Binary Search)

```
#include <iostream>
#include <string>
using namespace std;
int binarySearch(int myarray[], int beg, int end, int key)
{
    int mid;
    if(end >= beg) {
        mid = (beg + end)/2;
        if(myarray[mid] == key)
        {
            return mid+1;
        }
        else if(myarray[mid] < key) {
            return binarySearch(myarray,mid+1,end,key);
        }
        else {
            return binarySearch(myarray,beg,mid-1,key);
        }
    }
    return -1;
}
int main ()
{
    int myarray[10] = {5,8,10,13,21,23,25,43,54,75};
    int key, location=-1;
    cout<<"The input array is"<<endl;
    for(int i=0;i<10;i++){
        cout<<myarray[i]<<" ";
    }
    cout<<endl;
    cout<<"Enter the key that is to be searched:"; cin>>key;
    location = binarySearch(myarray, 0, 9, key);
    if(location != -1)  {
        cout<<"Key found at location "<<location;
    }
    else   {
        cout<<"Requested key not found";
    }
}
```

OUTPUT:

Program 1

```
The input array is
21 43 23 54 75 13 5 8 25 10
Enter the key to be searched : 3
Could not find given key in the array
The input array is
21 43 23 54 75 13 5 8 25 10
Enter the key to be searched: 75
Key found at position 5 in the array
```

Program 2

```
The input array is
5 8 10 13 21 23 25 43 54 75
Enter the key to be searched
21
the location of the key is 5
```

RESULT:

Thus, the program for searching algorithms has been written and executed successfully.

EX: No: 8(b) IMPLEMENTATION OF SEARCHING AND SORTING ALGORITHMS

DATE:

AIM:

To Write a C++ Program for various sorting algorithms.

ALGORITHM:

STEP 1: Start the program.
STEP 2: Declare the class.
STEP 3: Declare the variables and its member function.
STEP 4: Define the inputs for the sorting process.
STEP 5: Define the sorting functions for performing sorting algorithms for the given inputs
STEP 6: Display the sorted values.
STEP 8: Stop the program.

SOURCE CODE:

1. Bubble sort

```
#include <iostream.h>
using namespace std;
void BubbleSort (int arr[], int n)
{
   int i, j;
   for (i = 0; i < n; ++i)
      {
         for (j = 0; j < n-i-1; ++j)
            {
            if (arr[j] > arr[j+1])
               {
               arr[j] = arr[j]+arr[j+1];
               arr[j+1] = arr[j]-arr[j + 1];
               arr[j] = arr[j]-arr[j + 1];
               }
            }
      }
}


int main()
{
   int n, i;
   cout<<"\nEnter the number of data element to be sorted: ";
   cin>>n;
   int arr[n];
   for(i = 0; i < n; i++)
   {
      cout<<"Enter element "<>arr[i];
   }
   BubbleSort(arr, n);
   cout<<"\nSorted Data ";
   for (i = 0; i < n; i++)
   cout<<"->"<<arr[i];
   return 0;
}
```

2. Selection sort

```
#include<iostream>
using namespace std;
void swapping(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
void display(int *array, int size) {
    for(int i = 0; i<size; i++)
    cout << array[i] << " ";
    cout << endl;
}
void selectionSort(int *array, int size) {
    int i, j, imin;
    for(i = 0; i<size-1; i++)
    {
    imin = i;
    for(j = i+1; j<size; j++)
    {
        if(array[j] < array[imin])
         {   imin = j;
             swap(array[i], array[imin]); }
    }
}
int main()
{
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter elements:" << endl;
    for(int i = 0; i<n; i++)
    {
        cin >> arr[i];
    }
    cout << "Array before Sorting: ";
    display(arr, n);
    selectionSort(arr, n);
    cout << "Array after Sorting: ";
    display(arr, n);
}
```

3. Merge Sort

```
#include<iostream.h>
using namespace std;
void swapping(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
void display(int *array, int size)
 {
    for(int i = 0; i<size; i++)
    cout << array[i] << " ";
    cout << endl;
}
void merge(int *array, int l, int m, int r)
{
    int i, j, k, nl, nr;
    nl = m-l+1; nr = r-m;
    int larr[nl], rarr[nr];
    for(i = 0; i<nl; i++)
    {
        array[i] = array[l+i];
        for(j = 0; j<nr; j++)
        {
        array[j] = array[m+1+j];
        i = 0; j = 0; k = l;
    while(i < nl && j<nr)
    {
      if(larr[i] <= rarr[j])
       {
        array[k] = larr[i];
        i++;
      }
     else{
        array[k] = rarr[j];
        j++;
        }
     k++;
    }
    while(i<nl) {
      array[k] = larr[i];
      i++; k++;
    }
    while(j<nr) {
      array[k] = rarr[j];
      j++; k++;
    }
}
void mergeSort(int *array, int l, int r)
{
  int m;
  if(l < r) {
  int m = l+(r-l)/2;
  mergeSort(array, l, m);
  mergeSort(array, m+1, r);
  merge(array, l, m, r);
  }
}


int main()
 {
  int n;
  cout << "Enter the number of elements: ";
  cin >> n;
  int arr[n];
  cout << "Enter elements:" << endl;
  for(int i = 0; i<n; i++) {
  cin >> arr[i];
  cout << "Array before Sorting: ";
  display(arr, n);
```

```
    mergeSort(arr, 0, n-1);
    cout << "Array after Sorting: ";
    display(arr, n);
}
```

4. Insertion sort

```
#include <bits/stdc++.h>


void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}


void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int arr[] = { 12, 11, 13, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);
    printArray(arr, n);

    return 0;
}
```

OUTPUT:

1. **Bubble Sort**

```
Enter the number of data element to be sorted:3
Enter element 1: 25
Enter element 2: 16
Enter element 3: 63
Sorted Data ->16->25->63
```

2. **Selection Sort**

```
Enter the number of elements: 3
Enter elements:
5
1
3
Array before Sorting: 5 1 3
Array after Sorting: 1 3 5
```

3. Merge Sort

```
Enter the number of elements: 3
Enter elements:
2
16
3
Array before Sorting: 2 16 3
Array after Sorting: 2 3 16
```

4. Insertion Sort

```
Enter the number of elements: 3
Enter elements:
55
15
35
Array before Sorting: 55 15 35
Array after Sorting: 11 35 55
```

RESULT:

Thus, the program for various sorting algorithms has been written and executed successfully.

EX: No: 10 **HASHING – ANY TWO COLLISION TECHNIQUES**
DATE:

AIM:

To write a program to perform hashing techniques to avoid collision.

ALGORITHM:

Linear Probing:

STEP 1: Create hash table and the size of table must be greater than or equal to total number of keys
STEP 2:Insert(k) – Keep probing until an empty slot is found. Once an empty slot is found, insert k.
STEP 3:Search(k) – Keep probing until slot's key doesn't become equal to k or an empty slot is reached.
STEP 4:Delete(k) – Delete operation is interesting. If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted".
STEP 5: To mark a node deleted use **dummy node** with key and value -1.

Quadratic Probing:

STEP 1: Create an array of structure (i.e a hash table).
STEP 2: Take a key and a value to be stored in hash table as input.
STEP 3: Corresponding to the key, an index will be generated i.e every key is stored in a particular array index.
STEP 4: Using the generated index, access the data located in that array index.
STEP 5: In case of absence of data, create one and insert the data item (key and value) into it and increment the size of hash table.
STEP 6: In case the data exists, probe through the subsequent Values

SOURCE CODE:

Program 1 (Linear Probing)

```
#include<iostream>
#include<limits.h>

using namespace std;


void Insert(int ary[],int hFn, int Size){
   int element,pos,n=0;

cout<<"Enter key element to insert\n";
cin>>element;

pos = element%hFn;

while(ary[pos]!= INT_MIN) {
if(ary[pos]== INT_MAX)
            break;

pos = (pos+1)%hFn;
n++;
if(n==Size)
            break;
}

if(n==Size)
        cout<<"Hash table was full of elements\nNo Place to insert this element\n\n";
else
        ary[pos] = element;    //Inserting element
}

void Delete(int ary[],int hFn,int Size){

int element,n=0,pos;

cout<<"Enter element to delete\n";
cin>>element;

pos = element%hFn;

while(n++ != Size){
if(ary[pos]==INT_MIN){
cout<<"Element not found in hash table\n";
break;
}
else if(ary[pos]==element){
ary[pos]=INT_MAX;
cout<<"Element deleted\n\n";
break;
}
else{
pos = (pos+1) % hFn;
}
}
if(--n==Size)
        cout<<"Element not found in hash table\n";
}

void Search(int ary[],int hFn,int Size){
int element,pos,n=0;

cout<<"Enter element you want to search\n";
cin>>element;

pos = element%hFn;

while(n++ != Size){
if(ary[pos]==element){
cout<<"Element found at index "<<pos<<"\n";
break;
}
else
            if(ary[pos]==INT_MAX ||ary[pos]!=INT_MIN)
                pos = (pos+1) %hFn;
}
if(--n==Size)

 cout<<"Element not found in hash table\n";
}

void display(int ary[],int Size){
int i;


cout<<"Index\tValue\n";

for(i=0;i<Size;i++)
        cout<<i<<"\t"<<ary[i]<<"\n";
}

int main(){
int Size,hFn,i,choice;

cout<<"Enter size of hash table\n";
cin>>Size;

int ary[Size];
```

```
   cout<<"Enter hash function [if mod 10 enter 10]\n";
cin>>hFn;

for(i=0;i<Size;i++)
        ary[i]=INT_MIN;

do{
cout<<"Enter your choice\n";
cout<<" 1-> Insert\n 2-> Delete\n 3-> Display\n 4-> Searching\n 0-> Exit\n";
cin>>choice;

switch(choice){
case 1:
Insert(ary,hFn,Size);
break;
case 2:
Delete(ary,hFn,Size);
break;
case 3:
display(ary,Size);
break;
case 4:
Search(ary,hFn,Size);
break;
default:
cout<<"Enter correct choice\n";
break;
}
}while(choice);

return 0;
}
```

Program 2 (Quadratic Probing)

```
#include <iostream>
#include <cstdlib>
#define T_S 10
using namespace std;
enum EntryType {
  Legi, Emp, Del};
  struct HashTableEntry {
    int e;
```

```
enum EntryType info;
  };
  struct HashTable {
    int s;
    HashTableEntry *t;
  };
  bool isPrime (int n) {
  if (n == 2 || n == 3)
    return true;
  if (n == 1 || n % 2 == 0)
    return false;
  for (int i = 3; i * i <= n; i += 2)
    if (n % i == 0)
      return false;
  return true;
}
int nextPrime(int n) {
  if (n <= 0)
    n == 3;
  if (n % 2 == 0)
    n++;
  for (; !isPrime( n ); n += 2);
    return n;
}
int HashFunc(int k, int s) {
  return k % s;
}
HashTable *initiateTable(int s) {
  HashTable *ht;
  if (s < T_S) {
    cout<<"Table Size is Too Small"<<endl;
    return NULL;
  }
  ht= new HashTable;
  if (ht == NULL) {
    cout<<"Out of Space"<<endl;
    return NULL;
  }
  ht->s = nextPrime(s);
  ht->t = new HashTableEntry [ht->s];
  if (ht->t == NULL) {
    cout<<"Table Size is Too Small"<<endl;
    return NULL;
  }
  for (int i = 0; i < ht->s; i++) {
    ht->t[i].info = Emp;
    ht->t[i].e = NULL;
  }
  return ht;
}
int SearchKey(int k, HashTable *ht) {
  int pos = HashFunc(k, ht->s);
  int collisions = 0;
  while (ht->t[pos].info != Emp && ht->t[pos].e != k) {
    pos = pos + 2 * ++collisions -1;
    if (pos >= ht->s)
      pos = pos - ht->s;
  }
  return pos;
}
void Insert(int k, HashTable *ht) {
  int pos = SearchKey(k, ht);
  if (ht->t[pos].info != Legi) {
    ht->t[pos].info = Legi;
    ht->t[pos].e = k;
  }
}
HashTable *Rehash(HashTable *ht) {
  int s = ht->s;
  HashTableEntry *t= ht->t;
  ht= initiateTable(2 * s);
  for (int i = 0; i < s; i++) {
    if (t[i].info == Legi)
      Insert(t[i].e, ht);
  }
  free(t);
  return ht;
}
void display(HashTable *ht) {
  for (int i = 0; i < ht->s; i++) {
    int value = ht->t[i].e;
    if (!value)
      cout<<"Position: "<<i + 1<<" Element: Null"<<endl;
    else
      cout<<"Position: "<<i + 1<<" Element: "<<value<<endl;
  }
}
int main() {
  int v, s, pos, i = 1;
  int c;
  HashTable *ht;
  while(1) {
    cout<<"1.Initialize size of the table"<<endl;
    cout<<"2.Insert element into the table"<<endl;
    cout<<"3.Display Hash Table"<<endl;
    cout<<"4.Rehash The Table"<<endl;
  cout<<"5.Exit"<<endl;
    cout<<"Enter your choice: ";
    cin>>c;
    switch(c) {
      case 1:
        cout<<"Enter size of the Hash Table: ";
        cin>>s;
        ht = initiateTable(s);
        cout<<"Size of Hash Table: "<<nextPrime(s);
      break;
      case 2:
        if (i > ht->s) {
          cout<<"Table is Full, Rehash the table"<<endl;
          continue;
        }
        cout<<"Enter element to be inserted: ";
        cin>>v;
        Insert(v, ht);
        i++;
      break;
      case 3:
        display(ht);
      break;
      case 4:
        ht = Rehash(ht);
      break;
      case 5:
        exit(1);
      default:
        cout<<"\nEnter correct option\n";
    }
  }
  return 0;
}
```

OUTPUT:

Linear Probing

```
Enter size of hash table
10
Enter hash function [if mod 10 enter 10]
10
Enter your choice
 1-> Insert
 2-> Delete
 3->Display
 4->Searching
 0->Exit
1
Enter key element to insert
12
Enter your choice
 1-> Insert
 2-> Delete
 3->Display
 4->Searching
 0->Exit
1
Enter key element to insert
22
Enter your choice
 1-> Insert
 2-> Delete
 3->Display
 4->Searching
 0->Exit
1
Enter key element to insert
32
Enter your choice
 1-> Insert
 2-> Delete
 3->Display
 4->Searching
 0->Exit
3
Index   Value
```

```
0       -2147483648
1       -2147483648
2       12
3       22
4       32
5       -2147483648
6       -2147483648
7       -2147483648
8       -2147483648
9       -2147483648
Enter your choice
 1-> Insert
 2-> Delete
 3->Display
 4->Searching
 0->Exit
2
Enter element to delete
12
Element deleted

Enter your choice
 1-> Insert
 2-> Delete
 3->Display
 4->Searching
 0->Exit
4
Enter element you want to search
32
Element found at index 4
Enter your choice
 1-> Insert
 2-> Delete
 3->Display
 4->Searching
 0->Exit
0
```

Quadratic Probing

```
  1.Initialize size of the table
  2.Insert element into the table
  3.Display Hash Table
  4.Rehash The Table
  5.Exit
```

```
Enter your choice: 1
Enter size of the Hash Table: 4
Table Size is Too Small
Size of Hash Table: 51.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 1
Enter size of the Hash Table: 10
Size of Hash Table: 111.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 2
Enter element to be inserted: 1
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 2
Enter element to be inserted: 2
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 2
Enter element to be inserted: 3
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 2
Enter element to be inserted: 4
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
```

```
Enter your choice: 2

Enter element to be inserted: 5
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 2
Enter element to be inserted: 6
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 2
Enter element to be inserted: 7
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 2
Enter element to be inserted: 8
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 2
Enter element to be inserted: 9
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 2
Enter element to be inserted: 10
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 2
Enter element to be inserted: 11
1.Initialize size of the table

2.Insert element into the table

3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 2
Table is Full, Rehash the table
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 3
Position: 1 Element: 11
Position: 2 Element: 1
Position: 3 Element: 2
Position: 4 Element: 3
Position: 5 Element: 4
Position: 6 Element: 5
Position: 7 Element: 6
Position: 8 Element: 7
Position: 9 Element: 8
Position: 10 Element: 9
Position: 11 Element: 10
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 4
1.Initialize size of the table
2.Insert element into the table
3.Display Hash Table
4.Rehash The Table
5.Exit
Enter your choice: 3
Position: 1 Element: Null
Position: 2 Element: 1
Position: 3 Element: 2
Position: 4 Element: 3
Position: 5 Element: 4
```

```
  Position: 6 Element: 5
  Position: 7 Element: 6
  Position: 8 Element: 7

  Position: 9 Element: 8
  Position: 10 Element: 9

  Position: 11 Element: 10
  Position: 12 Element: 11
  Position: 13 Element: Null
  Position: 14 Element: Null
  Position: 15 Element: Null
  Position: 16 Element: Null
  Position: 17 Element: Null
  Position: 18 Element: Null
  Position: 19 Element: Null
  Position: 20 Element: Null
  Position: 21 Element: Null
  Position: 22 Element: Null
  Position: 23 Element: Null
  1.Initialize size of the table
  2.Insert element into the table
  3.Display Hash Table
  4.Rehash The Table
  5.Exit
  Enter your choice: 2
  Enter element to be inserted: 20
  1.Initialize size of the table
  2.Insert element into the table
  3.Display Hash Table
  4.Rehash The Table
  5.Exit
Enter your choice: 5
```

RESULT:

Thus, the Hashing techniques for collision avoidance program has been written and executed successfully.