# Practicals

**1. Write C++ programs to implement the following using an array**

*Ans :*

Stack ADT

```
#include<iostream>
#include<conio.h>
using namespace std;
class Stack
{
int stk[10];
int top;
public:
Stack()
  {
top=-1;
  }
//This funtion is used to push an elemnt into stack
void push(int x)
  {
if(isFull())
    {
cout<<"Stack is full";
return;
    }
top++;
stk[top]=x;
cout<<"Inserted Element:"<<" "<<x;
  }
  //This functionn is used to check if the stack is full
bool isFull()
  {
int size=sizeof(stk)/sizeof(*stk);
```

```
if(top == size-1)
    {
return true;
    }
else
    {
return false;
    }
  }

  //This function is used to check if the stack is empty
bool isEmpty()
  {
if(top==-1)
    {
return true;
    }
else
    {
return false;
    }
  }

  //This function is used to pop an element of stack
void pop()
  {
if(isEmpty())
    {
cout<<"Stack is Empty";
return;
    }
cout<<"Deleted element is:" <<" "<<stk[top--];
  }
  //This function is used to display the elements of the stack
void display()
```

```
  {
if(top==-1)
    {
cout<<" Stack is Empty!!";
return;
    }
for(int i=top;i>=0;i--)
    {
cout<<stk[i] <<" ";
    }
  }
};

void main()
{
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 you want to push";
cin>> ch;
st.push(ch);
break;
case 2:  st.pop();
break;
case 3:  st.display();
break;
case 4:  exit(0);
        }
    }
}
```

Sample Output:

```
1.Push 2.Pop 3.Display 4.Exit
Enter ur choice 1
Enter the element you want to push 11
Inserted Element: 11
1.Push 2.Pop 3.Display 4.Exit
Enter ur choice 1
Enter the element you want to push 22
Inserted Element: 22
1.Push 2.Pop 3.Display 4.Exit
Enter ur choice 1
Enter the element you want to push 33
Inserted Element: 33
1.Push 2.Pop 3.Display 4.Exit
Enter ur choice 1
Enter the element you want to push 44
Inserted Element: 44
1.Push 2.Pop 3.Display 4.Exit
Enter ur choice 1
Enter the element you want to push 55
Inserted Element: 55
1.Push 2.Pop 3.Display 4.Exit
Enter ur choice 3
55 44 33 22 11
```

b) Queue ADT

```
#include <iostream>
using namespace std;
const int Max=10;
void Qinsert(int Q[],int &R,int F)
{
if ((R+1)%Max!=F)
   {
      R=(R+1)%Max;
cout<<"Data:";
cin>>Q[R];
```

```
  }
else
cout<<"Queue is Full!"<<endl;
}
void Qdelete(int Q[],int R,int &F)
{
if (R!=F)
  {
     F=(F+1)%Max;
cout<<Q[F]<<" Deleted!";
  }
else
cout<<"Queue is empty"<<endl;
}
void Qdisplay(int Q[],int R,int F)
{
int Cn=F;
while (Cn!=R)
  {
     Cn=(Cn+1)%Max;
cout<<Q[Cn]<<endl;
  }
}
int main()
{  //Initialisation Steps
int Que[Max],Rear=0,Front=0;
char Ch;
do
  {
cout<<"\n I:Insert/D:Delete/S:Show/Q:Quit ";
cin>>Ch;
switch(Ch)
     {
case 'I':Qinsert(Que,Rear,Front);
break;
```

```
case 'D':Qdelete(Que,Rear,Front);
break;
case 'S':Qdisplay(Que,Rear,Front);
break;
      }
}while (Ch!='Q');
return 0 ;
}
Output:
      I:Insert/D:Delete/S:Show/Q:Quit I
Data:10
      I:Insert/D:Delete/S:Show/Q:Quit I
Data:20
      I:Insert/D:Delete/S:Show/Q:Quit D
10 Deleted!
      I:Insert/D:Delete/S:Show/Q:Quit I
Data:30
      I:Insert/D:Delete/S:Show/Q:Quit I
Data:40
      I:Insert/D:Delete/S:Show/Q:Quit S
20
30
40
      I:Insert/D:Delete/S:Show/Q:Quit Q
...Program finished with exit code 0
Press ENTER to exit console.
```

**2. Write a C++ program to implement Circular queue using array.**

***Ans :***

```
/* Circular Queue */
#include<iostream>
#include<climits>
#define MAX 50
using namespace std;
// gloabal variables - queue_array, front, rear
```

```
int queue[MAX];
int front=-1,rear=-1;
// Utility to Enqueue an element to the Queue
void enqueue(int num)
{
    if((rear==MAX-1 && front==0) || (front==rear+1))
    {
        cout<<"Queue is Full !!\n";
        return;
    }
    if(rear == MAX-1) // front != 0
        rear = -1;
    rear++;
    queue[rear] = num;
    if(front == -1)
        front=0;
}
// Utility to Dequeue an element from Queue
void dequeue()
{
    if(front==-1)
    {
        cout<<"Queue is Empty !!\n";
        return;
    }
    int number;
    number = queue[front];
    queue[front] = INT_MAX;

    if(front == rear)
    {
        front = -1;
        rear = -1;
    }
    else if(front == MAX-1) //rear != MAX-1
```

```
            front = 0;
        else
            front++;

        cout<<"Dequeued element = "<<number<<"\n";
    }
    // Utility to display Front of Queue
    void queue_front()
    {
        if(front == -1)
        {
            cout<<"Queue is Empty !!\n";
            return;
        }
        cout<<"Front of the queue = "<<queue[front]<<"\n";
    }
    // Utility to display Rear of Queue
    void queue_rear()
    {
        if(rear == -1)
        {
            cout<<"Queue is Empty !!\n";
            return;
        }
        cout<<"Rear of the queue = "<<queue[rear]<<"\n";
}
// Utility to display entire Queue
void disp_queue()
{
    if(front == -1)
    {
        cout<<"Queue is Empty !!\n";
        return;
    }
    int i;
```

```
    for(i=0;i<MAX;i++)
    {
        if(queue[i] == INT_MAX)
            continue;
        cout<<queue[i]<<" ";
    }
    cout<<"\n";
}
// Driver Code
int main()
{
    int optn,number;

    for(int i=0;i<MAX;i++)
        queue[i] = INT_MAX;
    while(1)
    {
        cout<<"\nOperations available : \n";
        cout<<"1. ENQUEUE\n2. DEQUEUE\n3. Front\n4. Rear\n5. DISPLAY queue\n6. Exit\n\n";
        cout<<"Enter your choice : ";
        cin>>optn;
            switch(optn)
        {
            case(1):
                cout<<"Enter the number : ";
                cin>>number;
                enqueue(number);
                break;
            case(2):
                dequeue();
                break;
            case(3):
                queue_front();
                break;
```

```
                case(4):
                        queue_rear();
                        break;
                case(5):
                        disp_queue();
                        break;
                case(6):
                        exit(0);
            }
        }
        return 0;
    }
```

Output :

Operations available :

1. ENQUEUE
2. DEQUEUE
3. Front
4. Rear
5. DISPLAY queue
6. Exit

Enter your choice : 1

Enter the number : 12

Operations available :

1. ENQUEUE
2. DEQUEUE
3. Front
4. Rear
5. DISPLAY queue
6. Exit

Enter your choice : 1

Enter the number : 13

Operations available :

1. ENQUEUE
2. DEQUEUE
3. Front

```
4. Rear
5. DISPLAY queue
6. Exit
Enter your choice : 1
Enter the number : 14
Operations available :
1. ENQUEUE
2. DEQUEUE
3. Front
4. Rear
5. DISPLAY queue
6. Exit
Enter your choice : 5
12 13 14
```

**3. Write C++ programs to implement the following using a single linked list.**

*Ans :*

Stack ADT

```
#include <iostream>
#include <string.h>
usingnamespace std;
struct Node{
int stu_no;
char stu_name[50];
int p;
   Node *next;
};
Node *top;
class stack{
public:
void push(int n,char name[],int perc);
void pop();
void display();
};
void stack :: push(int n,char name[],int perc)
```

```
{
struct Node *newNode=new Node;
  //fill data part
 newNode->stu_no=n;
 newNode->p=perc;
 strcpy(newNode->stu_name,name);
  //link part
 newNode->next=top;
  //make newnode as top/head
 top=newNode;
}
void stack ::pop()
{
if(top==NULL){
cout<<"List is empty!"<<endl;
return;
  }
  cout<<top->stu_name<<" is removed."<<endl;
top=top->next;
}
void stack:: display()
{
if(top==NULL){
cout<<"List is empty!"<<endl;
return;
  }
struct Node *temp=top;
while(temp!=NULL){
cout<<temp->stu_no<<" ";
cout<<temp->stu_name<<" ";
cout<<temp->p<<" ";
cout<<endl;
temp=temp->next;
}
```

```
cout<<endl;
}
int main(){

stack s;
char ch;
do{
int n;
cout<<"ENTER CHOICE\n"<<"1.Push\n"<<"2.Pop\n"<<"3.Display\n";
cout<<"Make a choice: ";
cin>>n;
switch(n){
case1:
        Node n;
cout<<"Enter details of the element to be pushed : \n";
cout<<"Roll Number : ";
cin>>n.stu_no;
cout<<"Enter Name: ";
std::cin.ignore(1);
cin.getline(n.stu_name,50);
cout<<"Enter Percentage: ";
cin>>n.p;
        //push data into the stack
s.push(n.stu_no,n.stu_name,n.p);
break;
case2 :
        //pop data from stack
s.pop();
break;
case3 :
        //display data
s.display();
break;
default :
```

```
cout<<"Invalid Choice\n";
  }
cout<<"Do you want to continue ? : ";
cin>>ch;
}while(ch=='Y'||ch=='y');
return 0;
}
```

OUTPUT

R CHOICE

1. Push

2. Pop

3.Display

Make a choice: 1

Enter details of the element to be pushed :

Roll Number : 101

Enter Name: PRIYA

Enter Percentage: 99

Do you want to continue ? : y

ENTER CHOICE

1.Push

2.Pop

3.Display

Make a choice: 1

Enter details of the element to be pushed :

Roll Number : 102

Enter Name: Mahak

Enter Percentage: 95

Do you want to continue ? : y

ENTER CHOICE

1.Push

2.Pop

3.Display

Make a choice: 3

102 Mahak 95

101 PRIYA 99

b) Queue ADT

```
#include<iostream>
usingnamespace std;
struct Node{
int data;
   Node *next;
};
class Queue{
   public:
   Node *front,*rear;
Queue(){front=rear=NULL;}
void insert(int n);
void deleteitem();
void display();
   ~Queue();
};
void Queue::insert(int n){
   Node *temp=new Node;
if(temp==NULL){
cout<<"Overflow"<<endl;
return;
   }
temp->data=n;
temp->next=NULL;
   //for first node
if(front==NULL){
front=rear=temp;
   }
else{
rear->next=temp;
rear=temp;
   }
    cout<<n<<" has been inserted successfully."<<endl;
}
```

```
void Queue::display(){
if(front==NULL){
cout<<"Underflow."<<endl;
return;
  }
   Node *temp=front;
   //will check until NULL is not found
while(temp){
cout<<temp->data<<" ";
temp=temp->next;
  }
cout<<endl;
}
void Queue :: deleteitem()
  {
if (front==NULL){
cout<<"underflow"<<endl;
return;
  }
cout<<front->data<<" is being deleted "<<endl;
if(front==rear)//if only one node is there
front=rear=NULL;
else
front=front->next;
}
Queue ::~Queue()
{
while(front!=NULL)
  {
    Node *temp=front;
front=front->next;
delete temp;
  }
rear=NULL;
}
```

```
int main(){
   Queue Q;
Q.display();
Q.insert(10);
Q.insert(24);
Q.insert(28);
Q.insert(32);
Q.insert(30);
Q.display();
Q.deleteitem();
Q.deleteitem();
Q.deleteitem();
Q.deleteitem();
Q.deleteitem();
return 0;
}
```

Output

```
Underflow.
  10 has been inserted successfully.
  24 has been inserted successfully.
  28 has been inserted successfully.
  32 has been inserted successfully.
  30 has been inserted successfully.
  10 24 28 32 30
  10 is being deleted
  24 is being deleted
  28 is being deleted
  32 is being deleted
  30 is being deleted
```

**4. Write a C++ program to implement Circular queue using Single linked list.**

*Ans :*

```
#include <iostream>;
using namespace std;
class Queue {
  public:
```

```
    // Initialize front and rear
    int rear, front;
    // Circular Queue
    int size;
    int *circular_queue;
    Queue(int sz) {
      front = rear = -1;
      size = sz;
      circular_queue = new int[sz];
    }
    void enQueue(int elem);
    int deQueue();
    void displayQueue();
};
/* Function to create Circular queue */
void Queue::enQueue(int elem)
{
  if ((front == 0 &amp;amp;&amp;amp; rear == size-1) || (rear == (front-1)%(size-1))) {
    cout&amp;lt;&amp;lt;"\nQueue is Full";
    return;
  }
  else if (front == -1) {     /* Insert First Element */
    front = rear = 0;
    circular_queue[rear] = elem;
  }
  else if (rear == size-1 &amp;amp;&amp;amp; front != 0) {
    rear = 0;
    circular_queue[rear] = elem;
  }
  else {
    rear++;
    circular_queue[rear] = elem;
  }
}
```

```
// Function to delete element from Circular Queue
int Queue::deQueue()
{
  if (front == -1) {
    cout&lt;&lt;"\nQueue is Empty";
    return -1;
  }
  int data = circular_queue[front];
  circular_queue[front] = -1;
  if (front == rear) {
    front = -1;
    rear = -1;
  }
  else if (front == size-1)
    front = 0;
  else
    front++;
  return data;
}
//display elements of Circular Queue
void Queue::displayQueue()
{
  if (front == -1) {
    cout&lt;&lt;"\nQueue is Empty"&lt;&lt;endl;
    return;
  }
  cout&lt;&lt;"\nCircular Queue elements: ";
  if (rear &gt;= front) {
    for (int i = front; i &lt;= rear; i++)
      cout&lt;&lt;circular_queue[i]&lt;&lt;" ";
  }
  Else {
    for (int i = front; i &lt; size; i++)
      cout&lt;&lt;circular_queue[i]&lt;&lt;" ";
    for (int i = 0; i &lt;= rear; i++)
```

```
            cout&lt;&lt;circular_queue[i]&lt;&lt;" ";
        }
    }
    //main program
    int main()
    {
        Queue pq(5);
        // Insert elements in Circular Queue
        pq.enQueue(2);
        pq.enQueue(4);
        pq.enQueue(6);
        pq.enQueue(8);
        // Display elements present in Circular Queue
        pq.displayQueue();
        // Delete elements from Circular Queue
        cout&lt;&lt;"\nElement Dequeued = "&lt;&lt;pq.deQueue();
        cout&lt;&lt;"\nElement Dequeued = "&lt;&lt;pq.deQueue();
        pq.displayQueue();
        pq.enQueue(10);
        pq.enQueue(12);
        pq.enQueue(14);
        pq.displayQueue();
        pq.enQueue(10);
        return 0;
    }
```

Output:

```
Circular Queue elements: 2 4 6 8
Element Dequeued = 2
Element Dequeued = 4
Circular Queue elements: 6 8
Circular Queue elements: 6 8 10 12 14
Queue is Full
```

**5. Write a C++ program to implement the double ended queue ADT using double linked list.**

*Ans :*

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
class node
{
public:
   int data;
   class node *next;
   class node *prev;
};
class dqueue: public node
{
   node *head,*tail;
   int top1,top2;
public:
   dqueue()
   {
      top1=0;
      top2=0;
      head=NULL;
      tail=NULL;
   }
   void push(int x)
   {
      node *temp;
      int ch;
      if(top1+top2 >=5)
      {
         cout <<"dqueue overflow";
         return ;
```

```
    }
    if( top1+top2 == 0)
    {
      head = new node;
      head->data=x;
      head->next=NULL;
      head->prev=NULL;
     tail=head;
     top1++;
  }
 else
 {
    cout <<" Add element 1.FIRST 2.LAST\n Enter Your Choice:";
   cin >> ch;
   if(ch==1)
   {
     top1++;
     temp=new node;
     temp->data=x;
     temp->next=head;
     temp->prev=NULL;
     head->prev=temp;
    head=temp;
 }
 else
 {
   top2++;
   temp=new node;
   temp->data=x;
   temp->next=NULL;
   temp->prev=tail;
   tail->next=temp;
  tail=temp;
}
```

```
  }
}
void pop()
{
  int ch;
  cout <<"Delete 1.First Node 2.Last Node\n Enter Your Choice:";
  cin >>ch;
  if(top1 + top2 <=0)
  {
     cout <<"\nDqueue under flow";
     return;
  }
  if(ch==1)
  {
     head=head->next;
     head->prev=NULL;
     top1--;
  }
  else
  {
     top2--;
     tail=tail->prev;
     tail->next=NULL;
  }
}
void display()
{
  int ch;
  node *temp;
  cout <<"Display From 1.STARTING 2.ENDING\n Enter Your Choice";
  cin >>ch;
  if(top1+top2 <=0)
  {
     cout <<"under flow";
```

```
            return ;
          }
          if (ch==1)
          {
            temp=head;
            while(temp!=NULL)
            {
              cout << temp->data <<" ";
              temp=temp->next;
            }
          }
          else
          {
            temp=tail;
            while( temp!=NULL)
            {
              cout <<temp->data << " ";
              temp=temp->prev;
            }
        }
    }
};
int main()
{
  dqueue d1;
  int ch;
  while (1)
  {
    cout <<"1.INSERT 2.DELETE 3.DISPLAY 4.EXIT\n Enter Your Choice:";
   cin >>ch;
   switch(ch)
   {
   case 1:
     cout <<"Enter the Element: ";
```

```
        cin >> ch;
        d1.push(ch);
        break;
      case 2:
        d1.pop();
        break;
      case 3:
        d1.display();
        break;
      case 4:
        exit(1);
      }
    }<br>  return 0;
}
```

OUTPUT:

```
1.INSERT 2.DELETE 3.DISPLAY 4.EXIT
 Enter Your Choice:1
Enter the Element: 10
1.INSERT 2.DELETE 3.DISPLAY 4.EXIT
 Enter ur choice:1
Enter the Element: 15
 Add element 1.FIRST 2.LAST
 Enter Your Choice:2
1.INSERT 2.DELETE 3.DISPLAY 4.EXIT
 Enter Your Choice:3
Display From 1.STARTING 2.ENDING
 Enter Your Choice:1
10 15 1.INSERT 2.DELETE 3.DISPLAY 4.EXIT
 Enter Your Choice:4
```

**6. Write a C++ program to solve tower of Hanoi problem recursively**

*Ans :*

```
#include<iostream>
usingnamespace std;
//tower of HANOI function implementation
```

```
void TOH(int n,char Sour, char Aux,char Des)
{
    if(n==1)
    {
        cout<<"Move Disk "<<n<<" from "<<Sour<<" to "<<Des<<endl;
        return;
    }
    TOH(n-1,Sour,Des,Aux);
    cout<<"Move Disk "<<n<<" from "<<Sour<<" to "<<Des<<endl;
    TOH(n-1,Aux,Sour,Des);
}
//main program
int main()
{
    int n;
    cout<<"Enter no. of disks:";
    cin>>n;
    //calling the TOH
    TOH(n,'A','B','C');
    return 0;
}
```

Output

```
Enter no. of disks:3
Move Disk 1 from A to C
Move Disk 2 from A to B
Move Disk 1 from C to B
Move Disk 3 from A to C
Move Disk 1 from B to A
Move Disk 2 from B to C
Move Disk 1 from A to C
```

**7. Write C++ program to perform the following operations:**

*Ans :*

Insert an element into a binary search tree.

```
#include<iostream>
```

```
usingnamespace std;
template<classt>
classbst
{
struct NODE
{
t data;
 NODE* left, *right;
};
public:
NODE* createnewnode(t data)
{
  NODE* createnewnode = newNODE();
createnewnode->data = data;
createnewnode->left = createnewnode->right = NULL;
      return createnewnode;
}
NODE* Insert(NODE* root,t data)
{
      if(root == NULL)
      {
      root = createnewnode(data);
      }
      elseif(data <= root->data)
      {
      root->left = Insert(root->left,data);
      }
      else
      {
      root->right = Insert(root->right,data);
      }
      return root;
}
voidinorder(NODE* root )
```

```
        if(root != NULL)
        {
        inorder(root->left);
        cout<<root->data<<" ";
        inorder(root->right);
        }
    }
    bst()
    {
    NODE* root = NULL;
    t a,b,c,d,e,f;
    cout<<endl<<"Enter the data"<<endl;
    cin>>a;
    cin>>b;
    cin>>c;
    cin>>d;
    cin>>e;
    cin>>f;
        root = Insert(root,a);
        root = Insert(root,b);
        root = Insert(root,c);
        root = Insert(root,d);
        root = Insert(root,e);
        root = Insert(root,f);
        cout<<"***THE BINARY SEARCH TREE*** "<<endl;
        cout<<"Inorder:";
        inorder(root);
        cout<<endl;
        insertion(root);
}
NODE* insertion(NODE* root)
{
    t n;
```

```
        cout<<"Enter the ELEMENT to be inserted in the BST"<<endl;
        cin>>n;
root = Insert(root,n);
cout<<"***THE BINARY SEARCH TREE(2)***"<<endl;
cout<<"Inorder:";
        inorder(root);

}
};
intmain()
{
        bst<int>a;
        cout<<endl<<"----------------------------------------"<<endl;
        bst<char>b;
        return0;

}
25
31
12
15
16
13
***THE BINARY SEARCH TREE***
Inorder:12 13 15 16 25 31
Enter the ELEMENT to be inserted in the BST
17
***THE BINARY SEARCH TREE(2)***
Inorder:12 13 15 16 17 25 31
Enter the data
g
u
q
i
***THE BINARY SEARCH TREE***
Inorder: a g i q t u
```

```
Enter the ELEMENT to be inserted in the BST
y
***THE BINARY SEARCH TREE(2)***
Inorder: a g i q t u y
Process exited after 39.67 seconds with return value 0
Press any key to continue . . .
```

30

b) Delete an element from binary search tree.

```
#include<iostream>
usingnamespace std;
bool c=false;
struct node
{
      int data;
      node* left;
      node* right;
};
struct node* getNode(int data)
{
      node* newNode=new node();
      newNode->data=data;
      newNode->left=NULL;
      newNode->right=NULL;
      return newNode;
}
void inorder(struct node* root)
{
if (root != NULL)
   {
inorder(root->left);
cout<<root->data<<" ";
inorder(root->right);
   }
}
```

```
{
while(root->left!=NULL)
  {
root=root->left;
  }
return root;
}
struct node* Insert(struct node* root, int data)
{
    if (root == NULL)
        return getNode(data);
    if (data < root->data)
        root->left  = Insert(root->left, data);
    elseif (data > root->data)
        root->right = Insert(root->right, data);
    return root;
}
bool Search(node*root,int value)
{
    if(root==NULL)
        returnfalse;
    elseif(root->data == value)
    {
        returntrue;
    }
    elseif(value < root-> data)
        Search(root->left,value);
    elseif(value > root->data)
        Search(root->right,value);
}
node* Delete( node* root,int value)
{
    c=Search(root,value);
    if(root==NULL)
```

```
        return root;
    elseif(value< root->data)
    {
        root->left= Delete(root->left,value);
    }
    elseif(value> root->data)
    {
        root->right= Delete(root->right,value);
    }
    // Node deletion
    else
    {
        //case 1: Leaf Node
        if(root->left==NULL&&root->right==NULL)
        {
            delete root;
            root=NULL;
            return root;
        }
        //case 2: one child
        elseif(root->left==NULL)
        {
            struct node* temp=root;
            root=root->right;
            delete temp;
            return root;
    }
    elseif(root->right==NULL)
    {
        struct node* temp=root;
        root=root->left;
        delete temp;
        return root;
}
```

```
        //case 3: 2 child
        else
        {
            struct node*temp=findMin(root->right);
            root->data=temp->data;
            root->right=Delete(root->right,temp->data);
        }
    }
    return root;
}
int main()
{
    node* root=NULL;
    root=Insert(root,20);
    Insert(root,15);
    Insert(root,25);
    Insert(root,18);
    Insert(root,10);
    Insert(root,16);
    Insert(root,19);
    Insert(root,17);
    cout<<"Before Deletion: "<<endl;
    cout<<"Inorder: ";
    inorder(root);
    cout<<endl<<endl;
    Delete(root,15);
    if(c)
    {
        cout<<"Node Deleted"<<endl;
        cout<<"\nAfter Deletion: "<<endl;
        cout<<"Inorder: ";
        inorder(root);
        cout<<endl;
    }
    else
```

```
        cout<<"Node Not Found"<<endl;

    return 0;
}
```

34

Output

Before Deletion:

Inorder: 10 15 16 17 18 19 20 25

Node Deleted

After Deletion:

Inorder: 10 16 17 18 19 20 25

c) Search for a key in a binary search tree.

```
#include <iostream>
using namespace std;
// Data structure to store a BST node
struct Node
{
    int data;
    Node* left = nullptr, *right = nullptr;
    Node() {}
    Node(int data): data(data) {}
};
// Recursive function to insert a key into a BST
Node* insert(Node* root, int key)
{
    // if the root is null, create a new node and return it
    if (root == nullptr) {
        return new Node(key);
    }
    // if the given key is less than the root node, recur for the left subtree
    if (key < root->data) {
        root->left = insert(root->left, key);
    }
    // if the given key is more than the root node, recur for the right subtree
    else {
        root->right = insert(root->right, key);
```

35

```
    }
    return root;
}
// Recursive function to search in a given BST
void search(Node* root, int key, Node* parent)
{
    // if the key is not present in the key
    if (root == nullptr)
    {
        cout << "Key not found";
        return;
    }
    // if the key is found
    if (root->data == key)
    {
        if (parent == nullptr) {
            cout << "The node with key " << key << " is root node";
        }
        else if (key < parent->data)
        {
            cout << "The given key is the left node of the node with key "
                << parent->data;
        }
        else {
            cout << "The given key is the right node of the node with key "
                << parent->data;
        }
        return;
    }
    // if the given key is less than the root node, recur for the left subtree;
    // otherwise, recur for the right subtree
    if (key < root->data) {
        search(root->left, key, root);
    } else {
```

```
        search(root->right, key, root);
    }
}
int main()
{
    int keys[] = { 15, 10, 20, 8, 12, 16, 25 };
    Node* root = nullptr;
    for (int key: keys) {
        root = insert(root, key);
    }
    search(root, 25, nullptr);
    return 0;
}
```

Output:

The given key is the right node of the node with key 20

**8. Write C++ programs for the implementation tree traversal technique BFS.**

***Ans :***

```
#include <iostream>
#include <conio.h>
usingnamespace std;
int c =0, t =0;
struct node_info
{
int no;
int st_time;
}*q =NULL, *r =NULL, *x =NULL;
struct node
{
   node_info *pt;
node*next;
}*front =NULL, *rear =NULL, *p =NULL, *np =NULL;
void push(node_info *ptr)
{
np=new node;
```

```
np->pt = ptr;
np->next =NULL;
if(front ==NULL)
{
front= rear = np;
rear->next =NULL;
}
else
{
rear->next = np;
rear= np;
rear->next =NULL;
}
}
node_info *remove()
{
if(front ==NULL)
{
cout<<"empty queue\n";
}
else
{
    p = front;
    x = p->pt;
front= front->next;
delete(p);
return(x);
}
}
void bfs(int*v,int am[][7],int i)
{
if(c ==0)
{
    q =new node_info;
    q->no = i;
```

```
        q->st_time = t++;
 cout<<"time of visitation for node "<<q->no<<":"<<q->st_time<<"\n\n";
 v[i]=1;
 push(q);
 }
 c++;
 for(int j =0; j <7; j++)
 {
 if(am[i][j]==0||(am[i][j]==1&& v[j]==1))
 continue;
 elseif(am[i][j]==1&& v[j]==0)
 {
        r =new node_info;
 r->no = j;
 r->st_time = t++;
 cout<<"time of visitation for node "<<r->no<<":"<<r->st_time<<"\n\n";
 v[j]=1;
 push(r);
 }
 }
 remove();
 if(c <=6&& front !=NULL)
 bfs(v, am, remove()->no);
 }
 int main()
 {
int v[7], am[7][7];
for(int i =0; i <7; i++)
v[i]=0;
for(int i =0; i <7; i++)
{
cout<<"enter the values for adjacency matrix row:"<<i+1<<endl;
for(int j =0; j <7; j++)
```

```
        q->st_time = t++;
cout<<"time of visitation for node "<<q->no<<":"<<q->st_time<<"\n\n";
v[i]=1;
push(q);
}
c++;
for(int j =0; j <7; j++)
{
if(am[i][j]==0||(am[i][j]==1&& v[j]==1))
continue;
elseif(am[i][j]==1&& v[j]==0)
{
        r =new node_info;
r->no = j;
r->st_time = t++;
cout<<"time of visitation for node "<<r->no<<":"<<r->st_time<<"\n\n";
v[j]=1;
push(r);
}
}
remove();
if(c <=6&& front !=NULL)
bfs(v, am, remove()->no);
}
int main()
{
int v[7], am[7][7];
for(int i =0; i <7; i++)
v[i]=0;
for(int i =0; i <7; i++)
{
cout<<"enter the values for adjacency matrix row:"<<i+1<<endl;
for(int j =0; j <7; j++)
```

```
{
cin>>am[i][j];
}
}
bfs(v, am, 0);
getch();
}
```

Output

enter the values for adjacency matrix row:1

0
1
1
0
0
1
1

enter the values for adjacency matrix row:2

1
0
0
0
0
0
0

enter the values for adjacency matrix row:3

1
0
0
0
0
0
1

time of visitation for node 0:0

time of visitation for node 1:1

time of visitation for node 2:2

**9. Write a C++ program that uses recursive functions to traverse a binary search tree.**

*Ans :*

a) Pre-order

b) In-order

c) Post-order

```
// C++ program for different tree traversals
#include <iostream>
using namespace std;
/* A binary tree node has data, pointer to left childand a pointer to right child */
struct Node {
  int data;
  struct Node *left, *right;
  Node(int data)
  {
    this->data = data;
    left = right = NULL;
  }
};
/* Given a binary tree, print its nodes according to the
"bottom-up" postorder traversal. */
void printPostorder(struct Node* node)
{
  if (node == NULL)
    return;
  // first recur on left subtree
  printPostorder(node->left);
  // then recur on right subtree
  printPostorder(node->right);
  // now deal with the node
  cout << node->data << " ";
}
/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct Node* node)
{
```

```
    if (node == NULL)
        return;
    /* first recur on left child */
    printInorder(node->left);
    /* then print the data of node */
    cout << node->data << " ";
    /* now recur on right child */
    printInorder(node->right);

}
/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct Node* node)
{
    if (node == NULL)
        return;
    /* first print data of node */
    cout << node->data << " ";
    /* then recur on left subtree */
    printPreorder(node->left);
    /* now recur on right subtree */
    printPreorder(node->right);
}
/* Driver program to test above functions*/
int main()
{
    struct Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    cout << "\nPreorder traversal of binary tree is \n";
    printPreorder(root);
    cout << "\nInorder traversal of binary tree is \n";
    printInorder(root);
    cout << "\nPostorder traversal of binary tree is \n";
    printPostorder(root);
```

```
    return 0;
}
```

Output:

Preorder traversal of binary tree is

1 2 4 5 3

Inorder traversal of binary tree is

4 2 5 1 3

Postorder traversal of binary tree is

4 5 2 3 1



**10. Write a C++ program to find height of a tree.**

*Ans :*

```
#include <iostream>
using namespace std;
// Data structure to store a Binary Tree node
struct Node
{
int key;
    Node *left, *right;

Node(int key)
  {
this->key = key;
this->left = this->right = NULL;
  }
};
// Recursive function to calculate height of given binary tree
int height(Node* root)
{
// Base case: empty tree has height 0
if (root == NULL)
return 0;
// recur for left and right subtree and consider maximum depth
return 1 + max(height(root->left), height(root->right));
}
```

```
// main function
int main()
{
   Node* root = NULL;
root = new Node(10);
root->left = new Node(1);
root->right = new Node(25);
root->left->left = new Node(9);
root->left->right = new Node(10);
root->right->left = new Node(13);
root->right->right = new Node(20);
cout<< "The height of the binary tree is " << height(root);
return 0;
}
```

Output :

The height of binary tree is 3.

**11 Write a C++ program to find MIN and MAX element of a BST.**

*Ans :*

```
#include<iostream>
using namespace std;
struct BSTNode
{
int data;
BSTNode *left;
   BSTNode *right;
};
 BSTNode* getnewnode(int data)
{
 BSTNode* temp=new BSTNode();
temp->data=data;
    temp->left=temp->right=NULL;
    return temp;
}
BSTNode* insertnode(struct BSTNode *root,int data)
```

```
{
if(root==NULL)
{
    root=getnewnode(data);
}
else if(data<=root->data)
{
root->left=insertnode(root->left,data);
}
else
{
root->right=insertnode(root->right,data);
}
return root;
}
bool search(BSTNode *root,int data)
{
if(root==NULL) return false;
else if(root->data==data) return true;
else if(data<root->data)
return search(root->left,data);
else
return search(root->right,data);
}
void preorder(BSTNode* root)
{
if(root==NULL)
return ;
else
{
cout<<root->data<<" ";
preorder(root->left);
preorder(root->right);
```

```
}
void inorder(BSTNode* root)
{
if(root==NULL)
return ;
else
{
inorder(root->left);
cout<<root->data<<" ";
inorder(root->right);
}
}
void postorder(BSTNode* root)
{
if(root==NULL)
return ;
else
{
    postorder(root->left);
postorder(root->right);
    cout<<root->data<<" ";
}
}
int findmin(BSTNode* root) //itterative method
{
if(root==NULL)
{
cout<<"\nerror: tree is empty ";
return -1;
}
BSTNode* current=root;
while(current->left)
current=current->left;
return current->data;
```

```
}
int findmax(BSTNode* root)   //itterative method find the maximum element
{
if(root==NULL)
{
cout<<"\n error: tree is empty ";
return -1;
}
BSTNode* current=root;
while(current->right)
current=current->right;
return current->data;
}
int recursive_findMin(BSTNode* root)
{
if(root==NULL)
{
cout<<"\n error: tree is empty ";
return -1;
}
else if(root->left==NULL)
return root->data;
   return recursive_findMin(root->left);
}
int recursive_findMax(BSTNode *root)
{
if(root==NULL)
{
cout<<"\n error: tree is empty ";
return -1;
}
else if(root->right==NULL)
return root->data;
  return recursive_findMax(root->right);
```

```
}
int findHeight(BSTNode* root)
{
if(root==NULL)
return -1;
return max(findHeight(root->left),findHeight(root->right))+1;
}
int main()
{
int x;
struct BSTNode* root=NULL;
root=insertnode(root,10);
   root=insertnode(root,15);
root=insertnode(root,9);
root=insertnode(root,8);
   root=insertnode(root,7);
root=insertnode(root,6);
cout<<"enter the data to search ";
cin>>x;
if(search(root,x)==true)
   cout<<"\n data is found ";
else
cout<<"\n data is not found";
cout<<"\npreorder traversal ";
preorder(root);
cout<<"\ninorder traversal ";
inorder(root);
cout<<"\npostorder traversal ";
postorder(root);
cout<<"\n height of tree= "<<findHeight(root);
 cout<<"\n minimum element by itterative method: "<<findmin(root);
 cout<<"\n maximum element by itterative method: "<<findmax(root);
 cout<<"\n maximum element by recursive method : "<< recursive_findMax(root);
 return 0;
}
```

Output:-

enter the data to search 15

data Is found

preorder traversal 10 9 8 7 6 15

inorder traversal 6 7 8 9 10 15

postorder traversal 6 7 8 9 15 10

height of tree= 4

minimum element by itterative method: 6

maximum element by itterative method: 15

minimum element by recursive method : 6

maximum element by recursive method : 15

Process exited after 9.164 seconds with return value 0

Press any key to continue ...



**12 Write a C++ program to find Inorder Successor of a given node.**

*Ans :*

```
#include <iostream>
using namespace std;
// Data structure to store a BST node
struct Node
{
  int data;
  Node* left = nullptr, *right = nullptr;
  Node() {}
  Node(int data): data(data) {}
};
// Recursive function to insert a key into a BST
Node* insert(Node* root, int key)
{
  // if the root is null, create a new node and return it
  if (root == nullptr) {
    return new Node(key);
  }
  // if the given key is less than the root node, recur for the left subtree
  if (key < root->data) {
```

```
        root->left = insert(root->left, key);
    }
    // if the given key is more than the root node, recur for the right subtree
    else {
        root->right = insert(root->right, key);
    }
    return root;
}
// Helper function to find minimum value node in a given BST
Node* findMinimum(Node* root)
{
    while (root->left) {
        root = root->left;
    }
    return root;
}
// Recursive function to find an inorder successor for the given key in a BST.
// Note that successor `succ` is passed by reference to the function
Node* findSuccessor(Node* root, Node* succ, int key)
{
    // base case
    if (root == nullptr) {
        return succ;
    }
    // if a node with the desired value is found, the successor is the minimum value
    // node in its right subtree (if any)
    if (root->data == key)
    {
        if (root->right != nullptr) {
            return findMinimum(root->right);
        }
    }
    // if the given key is less than the root node, recur for the left subtree
    else if (key < root->data)
```

```
    {
        // update successor to the current node before recursing in the left subtree
        succ = root;
        return findSuccessor(root->left, succ, key);
    }
    // if the given key is more than the root node, recur for the right subtree
    else {
        return findSuccessor(root->right, succ, key);
    }
    return succ;
}
int main()
{
    int keys[] = { 15, 10, 20, 8, 12, 16, 25 };

    /* Construct the following tree
              15
             /  \
            /    \
           10     20
          /\     / \
         /  \   /   \
        8   12 16   25
    */
    Node* root = nullptr;
    for (int key: keys) {
        root = insert(root, key);
    }

    // find inorder successor for each key
    for (int key: keys)
    {
        Node* succ = findSuccessor(root, nullptr, key);
        if (succ != nullptr) {
```

```
        cout << "The successor of node " << key << " is " << succ->data;
    }
    else {
        cout << "No Successor exists for node " << key;
    }
    cout << endl;
  }
  return 0;
}
```

Output:

```
The successor of node 15 is 16
The successor of node 10 is 12
The successor of node 20 is 25
The successor of node 8 is 10
The successor of node 12 is 15
The successor of node 16 is 20
No Successor exists for node 25
```

**13. Write C++ programs to perform the following operations on B-Trees and AVL Trees.**

*Ans :*

a) Insertion b) Deletion

```
#include<iostream>
#include<stdlib.h>
using namespace std;
#define TRUE 1
#define FALSE 0
#define NULL 0
class AVL;
class AVLNODE
{
  friend class AVL;
private:
  int data;
  AVLNODE *left,*right;
  int bf;
```

```
};
class AVL
{
private:
   AVLNODE *root;
public:
   AVLNODE *loc,*par;
   AVL()
   {
      root=NULL;
   }
   int insert(int);
   void displayitem();
   void display(AVLNODE *);
   void removeitem(int);
   void remove1(AVLNODE *,AVLNODE *,int);
   void remove2(AVLNODE *,AVLNODE *,int);
   void search(int x);
   void search1(AVLNODE *,int);
};
int AVL::insert(int x)
{
   AVLNODE *a,*b,*c,*f,*p,*q,*y,*clchild,*crchild;
   int found,unbalanced;
  int d;
  if(!root) //special case empty tree
  {
    y=new AVLNODE;
    y->data=x;
    root=y;
    root->bf=0;
    root->left=root->right=NULL;
   return TRUE;
}
```

```
//phase 1:locate insertion point for x.a keeps track of the most
// recent node with balance factor +/-1,and f is the parent of a
// q follows p through the tree.
  f=NULL;
  a=p=root;
  q=NULL;
  found=FALSE;
  while(p&&!found)
  {
    //search for insertion point for x
    if(p->bf)
    {
      a=p;
      f=q;
    }
    if(x<p->data) //take left branch
    {
      q=p;
      p=p->left;
    }
    else if(x>p->data)
    {
      q=p;
      p=p->right;
    }
    else
    {
      y=p;
      found=TRUE;
    }
  } //end while
//phase 2:insert and rebalance.x is not in the tree and
// may be inserted as the appropriate child of q.
  if(!found)
```

```
{
  y = new AVLNODE;
  y->data=x;
  y->left=y->right=NULL;
  y->bf=0;
  if(x<q->data) //insert as left child
    q->left=y;
  else
    q->right=y; //insert as right child
 //adjust balance factors of nodes on path from a to q
 //note that by the definition of a,all nodes on this
 //path must have balance factors of 0 and so will change
 //to +/- d=+1 implies that x is inserted in the left
 // subtree of a d=-1 implies
 //to that x inserted in the right subtree of a.
   if(x>a->data)
   {
     p=a->right;
     b=p;
     d=-1;
   }
   else
   {
     p=a->left;
     b=p;
     d=1;
  }
  while(p!=y)
    if(x>p->data) //height of right increases by 1
    {
      p->bf=-1;
      p=p->right;
  }
  else //height of left increases by 1
```

```
    {
      p->bf=1;
      p=p->left;
    }
//is tree unbalanced
  unbalanced=TRUE;
  if(!(a->bf)||!(a->bf+d))
  {
    //tree still balanced
    a->bf+=d;
    unbalanced=FALSE;
  }
  if(unbalanced) //tree unbalanced,determine rotation type
  {
    if(d==1)
    {
      //left imbalance
      if(b->bf==1) //rotation type LL
      {
        a->left=b->right;
        b->right=a;
        a->bf=0;
        b->bf=0;
      }
      else //rotation type LR
      {
        c=b->right;
        b->right=c->left;
        a->left=c->right;
        c->left=b;
        c->right=a;
        switch(c->bf)
        {
        case 1:
          a->bf=-1; //LR(b)
```

```
            b->bf=0;
            break;
        case -1:
            b->bf=1; //LR(c)
            a->bf=0;
            break;
        case 0:
            b->bf=0; //LR(a)
            a->bf=0;
            break;
        }
        c->bf=0;
        b=c; //b is the new root
    } //end of LR
} //end of left imbalance
else //right imbalance
{
    if(b->bf==-1) //rotation type RR
    {
        a->right=b->left;
        b->left=a;
        a->bf=0;
        b->bf=0;
    }
    else //rotation type LR
    {
        c=b->right;
        b->right=c->left;
        a->right=c->left;
        c->right=b;
        c->left=a;
        switch(c->bf)
        {
        case 1:
```

```
            a->bf=-1; //LR(b)
            b->bf=0;
            break;
          case -1:
            b->bf=1; //LR(c)
            a->bf=0;
            break;
          case 0:
            b->bf=0; //LR(a)
            a->bf=0;
            break;
          }
          c->bf=0;
          b=c; //b is the new root
        } //end of LR
      }
//subtree with root b has been rebalanced and is the new subtree
      if(!f)
        root=b;
      else if(a==f->left)
        f->left=b;
      else if(a==f->right)
        f->right=b;
    } //end of if unbalanced
    return TRUE;
  } //end of if(!found)
  return FALSE;
} //end of AVL INSERTION
void AVL::displayitem()
{
  display(root);
}
void AVL::display(AVLNODE *temp)
{
```

```
    if(temp==NULL)
      return;
    cout<<temp->data<<" ";
    display(temp->left);
    display(temp->right);
  }
  void AVL::removeitem(int x)
  {
    search(x);
    if(loc==NULL)
    {
      cout<<"\nitem is not in tree :(";
      return;
    }
    if(loc->right!=NULL&&loc->left!=NULL)
      remove1(loc,par,x);
    else
      remove2(loc,par,x);
  }
  void AVL::remove1(AVLNODE *l,AVLNODE *p,int x)
  {
    AVLNODE *ptr,*save,*suc,*psuc;
    ptr=l->right;
    save=l;
    while(ptr->left!=NULL)
    {
      save=ptr;
      ptr=ptr->left;
    }
    suc=ptr;
    psuc=save;
    remove2(suc,psuc,x);
  if(p!=NULL)
    if(l==p->left)
```

```
        p->left=suc;
    else
        p->right=suc;
  else
    root=l;
  suc->left=l->left;
  suc->right=l->right;
  return;
}
void AVL::remove2(AVLNODE *s,AVLNODE *p,int x)
{
  AVLNODE *child;
  if(s->left==NULL && s->right==NULL)
    child=NULL;
  else if(s->left!=NULL)
    child=s->left;
  else
    child=s->right;
  if(p!=NULL)
    if(s==p->left)
      p->left=child;
    else
      p->right=child;
  else
    root=child;
}
void AVL::search(int x)
{
  search1(root,x);
}
void AVL::search1(AVLNODE *temp,int x)
{
  AVLNODE *ptr,*save;
  int flag;
```

```
if(temp==NULL)
{
    cout<<"\nthe tree is empty";
    return;
}
if(temp->data==x)
{
    cout<<"\nThe item is root and is found :)";
    par=NULL;
    loc=temp;
    par->left=NULL;
    par->right=NULL;
    return;
}
if( x < temp->data)
{
    ptr=temp->left;
    save=temp;
}
else
{
    ptr=temp->right;
    save=temp;
}
while(ptr!=NULL)
{
    if(x==ptr->data)
    {
        flag=1;
        cout<<"\nItemfound :)";
        loc=ptr;
        par=save;
    }
    if(x<ptr->data)
```

```
        ptr=ptr->left;
      else
        ptr=ptr->right;
    }
    if(flag!=1)
    {
      cout<<"Item is not there in tree :(";
      loc=NULL;
      par=NULL;
      cout<<loc;
      cout<<par;
    }
}
int main()
{
  AVL a;
  int x,y,c;
  char ch;
  do
  {
    cout<<"\n1.Insert";
    cout<<"\n2.Display";
    cout<<"\n3.Delete";
    cout<<"\n4.Search";
    cout<<"\n5.Exit";
    cout<<"\nEnter your choice of operation on AVL Tree :";
    cin>>c;
    switch(c)
    {
    case 1:
      cout<<"\nEnter an Element to be inserted into Tree :";
      cin>>x;
      a.insert(x);
      break;
```

```
        case 2:
            a.displayitem();
            break;
        case 3:
            cout<<"\nEnter an item for Deletion :";
            cin>>y;
            a.removeitem(y);
            break;
        case 4:
            cout<<"\nEnter an element to perform Search :";
            cin>>c;
            a.search(c);
            break;
        case 5:
            exit(0);
            break;
        default :
            cout<<"\nInvalid option try again";
        }
        cout<<"\nDo u want to continue (y/n) :";
        cin>>ch;
    }
    while(ch=='y'||ch=='Y');<br>    return 0;
}
```



OUTPUT:

1.Insert
2.Display
3.Delete
4.Search
5.Exit

Enter your choice of operation on AVL Tree :1

Enter an Element to be inserted into Tree :10

Do u want to continue (y/n) :y

1.Insert

2.Display
3.Delete
4.Search
5.Exit
Enter your choice of operation on AVL Tree :1
Enter an Element to be inserted into Tree :14
Do u want to continue (y/n) :y
1.Insert
2.Display
3.Delete
4.Search
5.Exit
Enter your choice of operation on AVL Tree :1
Enter an Element to be inserted into Tree :9
Do u want to continue (y/n) :y
1.Insert
2.Display
3.Delete
4.Search
5.Exit
Enter your choice of operation on AVL Tree :2
10 9 14

**14 Write C++ programs for sorting a given list of elements in ascending order using the following sorting methods.**

*Ans :*

a) Quick sort

```
#include<iostream>
#include<cstdlib>
usingnamespace std;
void swap(int*a,int*b){
  int temp;
  temp=*a;
  *a =*b;
  *b = temp;
}
```

```
intPartition(int a[],int l,int h){
  int pivot, index, i;
  index= l;
  pivot= h;
  for(i = l; i < h; i++){
    if(a[i]< a[pivot]){
      swap(&a[i],&a[index]);
      index++;
    }
  }
  swap(&a[pivot],&a[index]);
  return index;
}
intRandomPivotPartition(int a[],int l,int h){
  int pvt, n, temp;
  n =rand();
  pvt= l + n%(h-l+1);
  swap(&a[h],&a[pvt]);
  returnPartition(a, l, h);
}
intQuickSort(int a[],int l,int h){
  int pindex;
  if(l < h){
    pindex=RandomPivotPartition(a, l, h);
    QuickSort(a, l, pindex-1);
    QuickSort(a, pindex+1, h);
  }
  return0;
}
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 "<<i+1<<": ";
  cin>>arr[i];
}
QuickSort(arr,0, n-1);
cout<<"\nSorted Data ";
for(i =0; i < n; i++)
  cout<<"->"<<arr[i];
return0;
}
```



Output

Enter the number of data element to be sorted: 4 Enter element 1: 3

Enter element 2: 4

Enter element 3: 7

Enter element 4: 6

Sorted Data ->3->4->6->7

b) Merge sort

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
#define MAX_SIZE 5
usingnamespacestd;
voidmerge_sort(int, int);
voidmerge_array(int, int, int, int);
int arr_sort[MAX_SIZE];
intmain(){
int i;
cout<<"Simple C++ Merge Sort Example - Functions and Array\n";
cout<<"\nEnter "<< MAX_SIZE <<" Elements for Sorting : "<<endl;
for (i = 0; i < MAX_SIZE; i++)
cin>> arr_sort[i];
cout<<"\nYour Data  :";
for (i = 0; i < MAX_SIZE; i++) {
cout<<"\t"<< arr_sort[i];
```

```
    }
    merge_sort(0, MAX_SIZE - 1);
  cout<<"\n\nSorted Data :";
  for (i = 0; i < MAX_SIZE; i++) {
  cout<<"\t"<< arr_sort[i];
    }
  getch();
  }
  voidmerge_sort(int i, int j){
  int m;
  if (i < j) {
        m = (i + j) / 2;
        merge_sort(i, m);
        merge_sort(m + 1, j);
  // Merging two arrays
        merge_array(i, m, m + 1, j);
    }
  }
  voidmerge_array(int a, int b, int c, int d){
  int t[50];
  int i = a, j = c, k = 0;
  while (i <= b && j <= d) {
  if (arr_sort[i] < arr_sort[j])
  t[k++] = arr_sort[i++];
  else
  t[k++] = arr_sort[j++];
    }
 //collect remaining elements
 while (i <= b)
 t[k++] = arr_sort[i++];
 while (j <= d)
 t[k++] = arr_sort[j++];
for (i = a, j = 0; i <= d; i++, j++)
      arr_sort[i] = t[j];
```

```
}
```

Output

```
Enter 5 Elements for Sorting
67
57
45
32
13
Your Data  :  6757453213
Sorted Data :1332455767
```



**15. Write a C++ program to find optimal ordering of matrix multiplication.**

*Ans :*

```
#include<stdio.h>
#include<limits.h>
// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
int MatrixChainMultiplication(int p[], int n)
{
   int m[n][n];
   int i, j, k, L, q;
   for (i=1; i<n; i++)
      m[i][i] = 0;    //number of multiplications are 0(zero) when there is only one matrix
   //Here L is chain length. It varies from length 2 to length n.
   for (L=2; L<n; L++)
   {
      for (i=1; i<n-L+1; i++)
      {
         j = i+L-1;
         m[i][j] = INT_MAX;  //assigning to maximum value
         for (k=i; k<=j-1; k++)
         {
            q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
            if (q < m[i][j])
            {
               m[i][j] = q;    //if number of multiplications found less that number will be updated.
```

```
            }
        }
    }
}
    return m[1][n-1];  //returning the final answer which is M[1][n]
}
int main()
{
    int n,i;
    printf("Enter number of matrices\n");
    scanf("%d",&n);
    n++;
    int arr[n];
    printf("Enter dimensions \n");
    for(i=0;i<n;i++)
    {
        printf("Enter d%d :: ",i);
        scanf("%d",&arr[i]);
    }
    int size = sizeof(arr)/sizeof(arr[0]);
    printf("Minimum number of multiplications is %d ", MatrixChainMultiplication(arr, size));
    return 0;
}
```

Output

```
Enter number of matrices
4
Enter dimensions
Enter d0 :: 10
Enter d1 :: 100
Enter d2 :: 20
Enter d3 :: 5
Enter d4 :: 80
Minimum number of multiplications is 19000
```

**16. Write a C++ program that uses dynamic programming algorithm to solve the optimal binary search tree problem**

*Ans :*

```
#include <iostream>
#include <climits>
using namespace std;
// Find optimal cost to construct a binary search tree from keys
// `i` to `j`, where each key `k` occurs `freq[k]` number of times
int findOptimalCost(int freq[], int i, int j, int level)
{
  // base case
  if (j < i) {
    return 0;
  }
  int optimalCost = INT_MAX;
  // consider each key as a root and recursively find an optimal solution
  for (int k = i; k <= j; k++)
  {
    // recursively find the optimal cost of the left subtree
    int leftOptimalCost = findOptimalCost(freq, i, k - 1, level + 1);
    // recursively find the optimal cost of the right subtree
    int rightOptimalCost = findOptimalCost(freq, k + 1, j, level + 1);
    // current node's cost is `freq[k]×level`
    int cost = freq[k] * level + leftOptimalCost + rightOptimalCost;
    // update the optimal cost
    optimalCost = min (optimalCost, cost);
  }
  // Return minimum value
  return optimalCost;
}
int main()
{
  int freq[] = { 25, 10, 20 };
  int n = sizeof(freq) / sizeof(freq[0]);
  cout << "The optimal cost of constructing BST is "
```

```
    << findOptimalCost(freq, 0, n - 1, 1);
    return 0;
}
```

Output:

The optimal cost of constructing BST is 95

**17. Write a C++ program to implement Hash Table**

*Ans :*

71

```
#include <iostream>
#include<list>
using namespace std;
class hashclass
{
   int bucket; //no of buckets you want
   list<int>*hashtable;//container
   public:
   hashclass(int a)//constructor
   {
     bucket=a;
     hashtable=new list<int>[bucket];
   }
   void insert_element(int key)//function used to insert elements to the hashtable
   {
//to get the hash index of key
      int indexkey=hashFunction(key);
      hashtable[indexkey].push_back(key);
   }
   void delete_element(int key)//function used to delete elements from the hashtable
   {
      int indexkey=hashFunction(key);
      list<int>::iterator i=hashtable[indexkey].begin();
      for(;i!=hashtable[indexkey].end();i++)
      {
        if(*i==key)
          break;
```

```
    }
    if(i!=hashtable[indexkey].end())
    {
        hashtable[indexkey].erase(i);
    }
}
int hashFunction(int a)//function used to map values to key
{
    return (a%bucket);
}
void display_table()//used to display hashtable values
{

    for(int i=0;i<bucket;i++)
    {
        cout<<i;
        list<int>::iterator j=hashtable[i].begin();
    for(;j!=hashtable[i].end();j++)
        {
            cout<<"---->"<<*j;
        }
        cout<<endl;
    }
}
void search_element(int key)//used to search element
{
    int a=0;//will be 1 if element exist otherwise 0
    int indexkey=hashFunction(key);
        list<int>::iterator i=hashtable[indexkey].begin();
    for(;i!=hashtable[indexkey].end();i++)
    {
        if(*i==key)
        {
            a=1;
            break;
```

```
        }
    }
        if(a==1)
        {
            cout<<"The element you wanted to search is present in the hashtable."<<endl;
        }
        else
        {
            cout<<"Element not present"<<endl;
        }
    }
    ~hashclass()//destructor
    {
       delete[]hashtable;
    }
};
int main()
{
int bucketn,ch,element;
cout<<"enter the no of buckets"<<endl;
cin>>bucketn;
hashclass hashelement(bucketn);
while(1)
{
    cout<<"1.insert element to the hashtable"<<endl;
    cout<<"2.search element from the hashtable"<<endl;
    cout<<"3.delete element from the hashtable"<<endl;
    cout<<"4.display elements of the hashtable"<<endl;
    cout<<"5.exit"<<endl;
    cout<<"enter your choice"<<endl;
   cin>>ch;
   switch(ch)
   {
      case 1:cout<<"enter the element"<<endl;
      cin>>element;
```

```
    hashelement.insert_element(element);
    break;
    case 2:cout<<"enter the element you want to search"<<endl;
    cin>>element;
    hashelement.search_element(element);
    break;
    case 3:cout<<"enter the element to be deleted"<<endl;
    cin>>element;
    hashelement.delete_element(element);
    break;
    case 4:hashelement.display_table();
    break;
    case 5:return 0;
    default:cout<<"enter a valid value"<<endl;
}
}
 return 0;
}
```

SAMPLE OUTPUT:

```
enter the no of buckets
4
1.insert element to the hashtable
2.search element from the hashtable
3.delete element from the hashtable
4.display elements of the hashtable
5.exit
enter your choice
1
enter the element
13
1.insert element to the hashtable
2.search element from the hashtable
3.delete element from the hashtable
4.display elements of the hashtable
5.exit
```

```
enter your choice
1
enter the element
16
1.insert element to the hashtable
2.search element from the hashtable
3.delete element from the hashtable
4.display elements of the hashtable
5.exit
```

**18. Write C++ programs to perform the following on Heap**

*Ans :*

a) Build Heap

```
#include <iostream>
using namespace std;
int n; // size of array
// To heapify a subtree rooted with node i which is
// an index in arr[]. N is size of heap
// we are using an array to create min heap
void heapify(int arr[], int i)
{
   int smallest = i; // The node which will be heapified
   int l = 2 * i + 1; // left child node
   int r = 2 * i + 2; // right child node
   // Check if left child is smaller than its parent
   if (l < n && arr[l] < arr[smallest])
      smallest = l;
   // Check if right child is smaller than smallest
   if (r < n && arr[r] < arr[smallest])
      smallest = r;
   // If smallest is not parent
   if (smallest != i) {
      swap(arr[i], arr[smallest]);
      // Recursively heapify the affected sub-tree
      heapify(arr, smallest);
```

```
  }
}
// Function to build a Max-Heap from the given array
void buildHeap(int arr[])
{
  // Perform level order traversal
  // from last non-leaf node and heapify each node
  for (int i = n; i >= 0; i--) {
    heapify(arr, i);
  }
}
// Driver Code
int main()
{
  // Sample array
  int arr[] = { 1, 5, 6, 8, 9, 7, 3};
  // calculating size of array
  n = sizeof(arr) / sizeof(arr[0]);
  cout<< "Array representation before buildHeap is: "<<endl;
  for (int i = 0; i < n; ++i)
    cout<< arr[i] << " ";
  cout<<endl;
  // call the buildheap method on the array
  buildHeap(arr);
 cout<< "Array representation after buildHeap is: "<<endl;
// print the heap
  for (int i = 0; i < n; ++i)
    cout<< arr[i] << " ";
    return 0;
}
```

Output

1.32s

Array representation before buildHeap is:

1 5 6 8 9 7 3

Array representation after buildHeap is:

1 5 3 8 9 7 6

b) Insertion

```
#include<iostream>
using namespace std;
/*Heapify using Bottom up approach*/
void heapify(int a[],int n,int i)
{
    int parent;
    parent = (i-1)/2; //for finding the parent of child
    if(parent>=0)     //Check until index < 0
    {
        if(a[parent]<a[i])
        {
            swap(a[parent],a[i]);
            heapify(a,n,parent); //recursive heapify function
        }
    }
}
/*Inserting the New Element into the Heap*/
void insert(int a[],int &n,int val)
{
    /*Increase the Size of the Array by 1*/
    n=n+1;
    /*Insert the new element at the end of the Array*/
    a[n-1]=val;
    /*Heapify function*/
    heapify(a,n,n-1);
}
/*Printing the Heap*/
void print(int a[],int n)
{
    cout<<"\nThe Array Representation of Heap is\n";
    for(int i=0;i<n;i++)
    {
        cout<<a[i]<<" ";
```

```
    }
}
/*Driver Function*/
int main()
{
    /*Initial Max Heap is */
    int a[100]={10,5,3,2,4};
    int n = 5;
    /*The Element to be insert is 15*/
    int val = 15;
    /*Printing the Array*/
    print(a,n);
    /*Insert Function*/
    insert(a,n,val); //here we are passing the 'n' value by reference
    /*Printing the Array*/
    print(a,n);
    return 0;
}
```

OUTPUT

The Array Representation of Heap is

10 5 3 2 4

The Array Representation of Heap is //After inserting new element 15 is

15 5 10 2 4 3

c) Deletion

```
#include<iostream>
using namespace std;
/*Heapify the heap using top down approach*/
void heapify(int a[],int n,int i)
{
    int parent = i;
//checking whether the parent index is valid in the array or not
    if(i<n)
    {
        //formula for finding left subtree of parent
```

```
            int left = 2*i+1;
            //formula for finding Right subtree of parent
            int right = 2*i+2;
            //checking whether the children index is valid in the array or not
            if(left<n || right<n)
            {
                1//checking whether parent is less than children or not
                if(a[parent]<a[left] || a[parent]<a[right])
                {
 //finding the largest children
                    if(a[left]<a[right])
                    {
 //swaping the parent with largest children
                        swap(a[right],a[parent]);
                        heapify(a,n,right);  //Recursively heapify the heap
                    }
                    else
                    {
                        swap(a[left],a[parent]);
                        heapify(a,n,right);
                    }
                }
            }
        }
 }
 // Function to delete the root from Heap
 void DeleteElement(int a[],int &n)
 {
        swap(a[0],a[n-1]); // swaping root and the last element
        n=n-1; // Decrease size of heap by 1
        heapify(a,n,0); //heapify the root node
}
/*Printing the Heap*/
void printArray(int a[],int n)
```

```
{
    for(int i=0;i<n;i++)
    {
        cout<<a[i]<<" ";
    }
}
/*Driver Function*/
int main()
{
    /*Initial Max Heap is */
    int a[] = {10,5,3,2,4};
    int n=5;
    cout<<"Before Deletion"<<endl;
    /*Printing the Array*/
    printArray(a,n);
    /*Deleting the root*/
    DeleteElement(a,n);
    cout<<"\nAfter Deletion"<<endl;
    /*Printing the Array*/
    printArray(a,n);
    return 0;
}
```

OUTPUT

Before Deletion

10 5 3 2 4

After Deletion

5 4 3 2

**19. Write C++ programs to perform following operations on Skip List**

*Ans :*

a) Insertion b) Deletion

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <cstring>
```

```
#define MAX_LEVEL 6
constfloat P =0.5;
usingnamespace std;
/*
 * Skip Node Declaration
 */
struct snode
{
int value;
snode**forw;
snode(int level, int&value)
{
forw=new snode *[level +1];
memset(forw, 0, sizeof(snode*)*(level +1));
this->value = value;
}
   ~snode()
{
delete[] forw;
}
};
/*
 * Skip List Declaration
 */
struct skiplist
{
snode*header;
int value;
int level;
skiplist()
{
header=new snode(MAX_LEVEL, value);
level=0;
}
```

```
~skiplist()
{
delete header;
}
void display();
bool contains(int&);
void insert_element(int&);
void delete_element(int&);
};
/*
 * Main: Contains Menu
 */
int main()
{
skiplist ss;
int choice, n;
while(1)
{
cout<<endl<<"----------------------"<<endl;
cout<<endl<<"Operations on Skip list"<<endl;
cout<<endl<<"----------------------"<<endl;
cout<<"1.Insert Element"<<endl;
cout<<"2.Delete Element"<<endl;
cout<<"3.Search Element"<<endl;
cout<<"4.Display List "<<endl;
cout<<"5.Exit "<<endl;
cout<<"Enter your choice : ";
cin>>choice;
switch(choice)
{
case1:
cout<<"Enter the element to be inserted: ";
cin>>n;
        ss.insert_element(n);
```

```
if(ss.contains(n))
cout<<"Element Inserted"<<endl;
break;
case2:
cout<<"Enter the element to be deleted: ";
cin>>n;
if(!ss.contains(n))
{
cout<<"Element not found"<<endl;
break;
}
        ss.delete_element(n);
if(!ss.contains(n))
cout<<"Element Deleted"<<endl;
break;
case3:
cout<<"Enter the element to be searched: ";
cin>>n;
if(ss.contains(n))
cout<<"Element "<<n<<" is in the list"<<endl;
else
cout<<"Element not found"<<endl;
case4:
cout<<"The List is: ";
ss.display();
break;
case5:
exit(1);
break;
default:
cout<<"Wrong Choice"<<endl;
}
}
return0;
```

```
}
/*
 * Random Value Generator
 */
float frand()
{
return(float)rand()/RAND_MAX;
}
/*
 * Random Level Generator
 */
int random_level()
{
staticbool first =true;
if(first)
{
srand((unsigned)time(NULL));
first=false;
}
int lvl =(int)(log(frand())/log(1.-P));
return lvl < MAX_LEVEL ?lvl : MAX_LEVEL;
}
/*
 * Insert Element in Skip List
 */
void skiplist::insert_element(int&value)
{
snode*x = header;
snode*update[MAX_LEVEL +1];
memset(update, 0, sizeof(snode*)*(MAX_LEVEL +1));
for(int i = level;i >=0;i--)
{
while(x->forw[i]!=NULL&& x->forw[i]->value < value)
{
```

```
            x = x->forw[i];
        }
        update[i]= x;
        }
            x = x->forw[0];
        if(x ==NULL|| x->value != value)
        {
        int lvl = random_level();
        if(lvl > level)
        {
        for(int i = level +1;i <= lvl;i++)
        {
        update[i]= header;
        }
        level= lvl;
        }
            x =newsnode(lvl, value);
       for(int i =0;i <= lvl;i++)
       {
       x->forw[i]= update[i]->forw[i];
       update[i]->forw[i]= x;
       }
       }
       }
      /*
       * Delete Element from Skip List
       */
     void skiplist::delete_element(int&value)
     {
     snode*x = header;
     snode*update[MAX_LEVEL +1];
     memset(update, 0, sizeof(snode*)*(MAX_LEVEL +1));
    for(int i = level;i >=0;i--)
    {
```

```
while(x->forw[i]!=NULL&& x->forw[i]->value < value)
{
        x = x->forw[i];
}
update[i]= x;
}
   x = x->forw[0];
if(x->value == value)
{
for(int i =0;i <= level;i++)
{
if(update[i]->forw[i]!= x)
break;
update[i]->forw[i]= x->forw[i];
}
delete x;
while(level >0&& header->forw[level]==NULL)
{
level--;
}
}
}
/*
 * Display Elements of Skip List
 */
void skiplist::display()
{
const snode *x = header->forw[0];
while(x !=NULL)
{
cout<< x->value;
        x = x->forw[0];
if(x !=NULL)
cout<<" - ";
```

```
}
cout<<endl;
}
/*
 * Search Elemets in Skip List
 */
bool skiplist::contains(int&s_value)
{
snode*x = header;
for(int i = level;i >=0;i--)
{
while(x->forw[i]!=NULL&& x->forw[i]->value < s_value)
{
        x = x->forw[i];
}
}
   x = x->forw[0];
return x !=NULL&& x->value == s_value;
}
```

87

OUTPUT

```
---------------------------------
Operations on Skip list
---------------------------------
1.Insert Element
2.Delete Element
3.Search Element
4.Display List
5.Exit
Enter your choice :1
Enter the element to be inserted: 7
Element Inserted
---------------------------------
Operations on Skip list
---------------------------------
```

```
1.Insert Element
2.Delete Element
3.Search Element
4.Display List
5.Exit
Enter your choice :1
Enter the element to be inserted: 9
Element Inserted
---------------------------
Operations on Skip list
---------------------------
1.Insert Element
2.Delete Element
3.Search Element
4.Display List
5.Exit
Enter your choice :4
The List is: 7 - 9
```

**20. Write a C++ Program to Create a Graph using Adjacency Matrix Representation.**

*Ans :*

```
#include <iostream>
using namespace std;
// stores adjacency list items
struct adjNode {
   int val, cost;
   adjNode* next;
};
// structure to store edges
struct graphEdge {
   int start_ver, end_ver, weight;
};
class DiaGraph{
   // insert new nodes into adjacency list from given graph
   adjNode* getAdjListNode(int value, int weight, adjNode* head)  {
```

```
        adjNode* newNode = new adjNode;
        newNode->val = value;
        newNode->cost = weight;

        newNode->next = head;   // point new node to current head
        return newNode;
    }
    int N;  // number of nodes in the graph
public:
    adjNode **head;              //adjacency list as array of pointers
    // Constructor
    DiaGraph(graphEdge edges[], int n, int N) {
        // allocate new node
        head = new adjNode*[N]();
        this->N = N;
        // initialize head pointer for all vertices
        for (int i = 0; i < N; ++i)
            head[i] = nullptr;
        // construct directed graph by adding edges to it
        for (unsigned i = 0; i < n; i++) {
            int start_ver = edges[i].start_ver;
            int end_ver = edges[i].end_ver;
            int weight = edges[i].weight;
            // insert in the beginning
            adjNode* newNode = getAdjListNode(end_ver, weight, head[start_ver]);
                    // point head pointer to new node
            head[start_ver] = newNode;
            }
}
    // Destructor
    ~DiaGraph() {
for (int i = 0; i < N; i++)
    delete[] head[i];
    delete[] head;
```

```
    }
};
// print all adjacent vertices of given vertex
void display_AdjList(adjNode* ptr, int i)
{
    while (ptr != nullptr) {
        cout << "(" << i << ", " << ptr->val
            << ", " << ptr->cost << ") ";
        ptr = ptr->next;
    }
    cout << endl;
}
// graph implementation
int main()
{
    // graph edges array.
    graphEdge edges[] = {
        // (x, y, w) -> edge from x to y with weight w
        {0,1,2},{0,2,4},{1,4,3},{2,3,2},{3,1,4},{4,3,3}
    };
    int N = 6;      // Number of vertices in the graph
    // calculate number of edges
    int n = sizeof(edges)/sizeof(edges[0]);
    // construct graph
    DiaGraph diagraph(edges, n, N);
    // print adjacency list representation of graph
    cout<<"Graph adjacency list "<<endl<<"(start_vertex, end_vertex, weight):"<<endl;
    for (int i = 0; i < N; i++)
    {
        // display adjacent vertices of vertex i
        display_AdjList(diagraph.head[i], i);
    }
    return 0;
}
```

Output:

Graph adjacency list

(start_vertex, end_vertex, weight):

(0, 2, 4) (0, 1, 2)

(1, 4, 3)

(2, 3, 2)

(3, 1, 4)

(4, 3, 3)

**21. Write a C++ program to implement graph traversal techniques**

*Ans :*

a) BFS

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
int cost[10][10],i,j,k,n,qu[10],front,rare,v,visit[10],visited[10];
int main()
{
   int m;
   cout <<"Enter no of vertices:";
   cin >> n;
   cout <<"Enter no of edges:";
   cin >> m;
   cout <<"\nEDGES \n";
   for(k=1; k<=m; k++)
   {
      cin >>i>>j;
      cost[i][j]=1;
   }
   cout <<"Enter initial vertex to traverse from:";
   cin >>v;
   cout <<"Visitied vertices:";
   cout <<v<<" ";
   visited[v]=1;
   k=1;
```

```
while(k<n)
{
   for(j=1; j<=n; j++)
      if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
      {
         visit[j]=1;
         qu[rare++]=j;
      }
     v=qu[front++];
     cout<<v <<" ";
     k++;
     visit[v]=0;
     visited[v]=1;
  }
  return 0;
}
```



OUTPUT

```
Enter no of vertices:4
Enter no of edges:4
EDGES
1 2
1 3
2 4
3 4
Enter initial vertex to traverse from:1
Visited vertices:1 2 3 4
12
```

b) DFS

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
int cost[10][10],i,j,k,n,stk[10],top,v,visit[10],visited[10];
int main()
{
```

```
    int m;
    cout <<"Enter no of vertices:";
    cin >> n;
    cout <<"Enter no of edges:";
    cin >> m;
    cout <<"\nEDGES \n";
    for(k=1; k<=m; k++)
    {
      cin >>i>>j;
      cost[i][j]=1;
    }
    cout <<"Enter initial vertex to traverse from:";
    cin >>v;
    cout <<"DFS ORDER OF VISITED VERTICES:";
    cout << v <<" ";
    visited[v]=1;
    k=1;
    while(k<n)
    {
      for(j=n; j>=1; j--)
        if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
        {
          visit[j]=1;
          stk[top]=j;
          top++;
        }
      v=stk[--top];
      cout<<v << " ";
      k++;
      visit[v]=0;
      visited[v]=1;
  }
  return 0;
}
```

OUTPUT

```
Enter no of vertices:4
Enter no of edges:4
EDGES
1 2
1 3
2 4
3 4
Enter initial vertex to traverse from:1
DFS ORDER OF VISITED VERTICES:1 2 4 3
```

94

**22. Write a C++ program to Heap sort using tree structure.**

*Ans :*

```
#include <iostream>
using namespace std;
// function to heapify the tree
void heapify(int arr[], int n, int root)
{
  int largest = root; // root is the largest element
  int l = 2*root + 1; // left = 2*root + 1
  int r = 2*root + 2; // right = 2*root + 2
  // If left child is larger than root
  if (l < n && arr[l] > arr[largest])
  largest = l;
  // If right child is larger than largest so far
  if (r < n && arr[r] > arr[largest])
  largest = r;
  // If largest is not root
  if (largest != root)
   {
    //swap root and largest
    swap(arr[root], arr[largest]);
    // Recursively heapify the sub-tree
    heapify(arr, n, largest);
   }
}
// implementing heap sort
void heapSort(int arr[], int n)
```

```
{
  // build heap
  for (int i = n / 2 - 1; i >= 0; i--)
  heapify(arr, n, i);
  // extracting elements from heap one by one
  for (int i=n-1; i>=0; i--)
  {
    // Move current root to end
    swap(arr[0], arr[i]);
    // again call max heapify on the reduced heap
    heapify(arr, i, 0);
  }
}
/* print contents of array - utility function */
void displayArray(int arr[], int n)
{
  for (int i=0; i<n; ++i)
  cout << arr[i] << " ";
  cout << "\n";
}
// main program
int main()
{
  int heap_arr[] = {4,17,3,12,9,6};
  int n = sizeof(heap_arr)/sizeof(heap_arr[0]);
  cout<<"Input array"<<endl;
  displayArray(heap_arr,n);
  heapSort(heap_arr, n);
  cout << "Sorted array"<<endl;
  displayArray(heap_arr, n);
}
```

Output:

Input array

4 17 3 12 9 6

Sorted array

3 4 6 9 12 17