Data Structures and Algorithm Laboratory

(210257)

Practical-1(A-1)

Problem Statement:

Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client ‘s telephone number. Make use of two collision handling techniques and compare them using number of comparisons required to find a set of telephone numbers

Code:

```
#include<iostream>
using namespace std;
class hashing
{
   long int mobile,hash_table[10];
   public:
      hashing()
      {
         for(int i=0;i<10;i++)
         {
            hash_table[i]=-1;
         }
      }
      void linear_prob();
      void quadratic_prob();
      void display();
};
void hashing::linear_prob()
{
   int index;
   cout<<"Enter your mobile number:\n";
   cin>>mobile;
   index=mobile%10;
   if(hash_table[index]=-1)
   {
      hash_table[index]=mobile;
```

```
    }
    else
    {
        while(hash_table[index]!=-1)
        {
            cout<<"Testing index"<<index<<endl;
            if(index==9)
            {
                index=0;
            }
            else
            {
                index++;
            }
        }
        hash_table[index]=mobile;
    }
}
void hashing::quadratic_prob()
{
  int index,a;
  cout<<"enter your mobile number";
  cin>>mobile;
  index=mobile%10;
int j=1;
if(hash_table[index]==-1)
{
hash_table[index]=mobile;
}
else
{
a=index%10;
while(j<10)
{
index=(a+(j*j)%10);
if(hash_table[index]==-1)
```

```
{
hash_table[index]=mobile;
break;
}
else
{
j++;
}
}
}
}
void hashing::display()
{
for(int i=0;i<10;i++)
{
cout<<i<<" "<<hash_table[i]<<endl;
}
}
int main()
{
hashing h;
int ch;
do
{
   cout<<"******MENU******"<<endl;
cout<<"1.LINEAR PROBING"<<endl;
cout<<"2.QUADRATIC PROBING"<<endl;
cout<<"3.DISPLAY"<<endl;
cout<<"4.EXIT"<<endl;
cout<<"Enter your choice:";
cin>>ch;
switch(ch)
{
case 1:h.linear_prob();
     break;
case 2:h.quadratic_prob();
```

```
    break;
case 3:h.display();
    break;
case 4: cout<<" Exiting Code";
     break;
}
}while(ch!=4);
return 0;
}
```

OUTPUT:

********MENU********

1.LINEAR PROBING

2.QUADRATIC PROBING

3.DISPLAY

4.EXIT

Enter your choice:1

Enter your mobile number:

9000033221

********MENU********

1.LINEAR PROBING

2.QUADRATIC PROBING

3.DISPLAY

4.EXIT

Enter your choice:2

enter your mobile number9000043211

********MENU********

1.LINEAR PROBING

2.QUADRATIC PROBING

3.DISPLAY

4.EXIT

Enter your choice:3

0 -1

1 9000033221

2 9000043211

3 -1

4 -1

5 -1

6 -1

7 -1

8 -1

9 -1

********MENU********

1.LINEAR PROBING

2.QUADRATIC PROBING

3.DISPLAY

4.EXIT

Enter your choice:4

Practical-2(A-4)

Problem Statement:

To create ADT that implement the "set" concept.
a. Add (newElement) -Place a value into the set
b. Remove (element) Remove the value

c. Contains (element) Return true if element is in collection
d. Size () Return number of values in collection Iterator () Return an iterator used to loop over collection
e. Intersection of two sets
f. Union of two sets
g. Difference between two sets
h. Subset

Code:

```
#include <iostream>

using namespace std;

class Set

{

   int a[100];int cnt;

   int no;

   public:

   Set()

   {

      cnt=0;

   }

   void add();

   void display();

   Set unionop(Set);

   int search(int);

   Set intersection(Set);

   Set Minus(Set);

   void remove();

   void subset(Set);

};


void Set :: add()

{

   cout<<"Enter how many number you want to enter "<<endl;

   cin>>no;

   for (int i=0; i<no; i++)

   {

      cout<<"Enter your number "<<i+1<<" :";

      cin>>a[cnt++];

   }
```

```
}

void Set :: display()
{
   for (int i=0;i<cnt;i++)
   {
      cout<<a[i]<<"  ";
   }
}
Set Set :: unionop(Set B)
{
   Set temp;
   for (int i = 0; i<cnt; i++)
   {
      temp.a[i] = a[i];
      temp.cnt++;
   }
   for (int i = 0; i<B.cnt;i++)
   {
      if(!temp.search(B.a[i]))
      {
         temp.a[temp.cnt++] = B.a[i];
      }
   }

   return temp;
}

int Set :: search(int s)
{
   for (int i = 0; i<cnt;i++)
   {
      if (s == a[i])
      {
         return 1;
```

```
        }

    }
    return 0;
}
Set Set :: intersection(Set B)
{
    Set temp;
    for (int i = 0; i<cnt; i++)
    {
        for (int j = 0; j<B.cnt ; j++)
        {
            if (a[i] == B.a[j])
            {
                temp.a[temp.cnt] = a[i];
                temp.cnt++;
            }
        }
    }
    return temp;
}
Set Set :: Minus (Set B)
{
    Set temp;
    int flag;
    for (int i = 0; i<cnt;i++)
    {
        flag = B.search(a[i]);
        if (flag == 0)
        {
            temp.a[temp.cnt++] = a[i];
        }
    }
    return temp;
}
void Set :: remove()
```

```
{
   int dnumber;
   int k = -1;
   cout<<"Enter number to be deleted :"<<endl;
   cin>>dnumber;

   for(int i = 0; i<cnt;i++)
   {
      if (a[i] == dnumber )
      {
            k =i;
            break;
      }
   }

   for (int j = k; j<cnt;j++)
   {
      a[j] = a[j+1];
   }
   cnt--;
}

void Set :: subset(Set B)
{
   int i;
   for (i=0;i<B.cnt;i++)
   {
      if(!search(B.a[i]))
      {
         cout<<"B is not subset of A"<<endl;
         break;
      }
   }

   if (i == B.cnt)
   {
```

```
        cout<<"B is subset of A"<<endl;
    }
}
int main()
{
    int ch;int snumber;
    Set obj;
    int ans;
    Set B;
    Set C;
    Set D;
    do{
        cout<<"\n-------------------------------------------------"<<endl;
        cout<<"\n1.Insert in the set A"<<endl;
        cout<<"2.Insert in the set B"<<endl;
        cout<<"3.Display"<<endl;
        cout<<"4.Search in the set"<<endl;
        cout<<"5.Union of two set "<<endl;
        cout<<"6.Intersection of two set "<<endl;
        cout<<"7.Minus of two set "<<endl;
        cout<<"8.Remove the element "<<endl;
        cout<<"9.Subset"<<endl;
        cout<<"10.Exit"<<endl;
        cout<<"\nEnter your choice "<<endl;
        cin>>ch;
        cout<<endl;

        switch(ch)
        {
            case 1:
            obj.add();
            break;
            case 2:
            B.add();
            break;
            case 3:
```

```
obj.display();
cout<<endl;
B.display();
break;
  case 4:
cout<<"Enter the number to be searched "<<endl;
cin>>snumber;
ans = (obj.search(snumber) || B.search(snumber));
if (ans == 1)
{
   cout<<"Element found!!!!"<<endl;
   break;
}
else
{
   cout<<"Element not found"<<endl;
   break;
}
case 5:
C = obj.unionop(B);
C.display();
break;
case 6:
D = obj.intersection(B);
D.display();
break;
case 7:
C = obj.Minus(B);
C.display();
break;
case 8:
obj.remove();
B.remove();
break;
case 9:
obj.subset(B);
```

```
            break;
        }
    }while(ch!=10);
    return 0;
}
```

OUTPUT:

1.Insert in the set A

2.Insert in the set B

3.Display

4.Search in the set

5.Union of two set

6.Intersection of two set

7.Minus of two set

8.Remove the element

9.Subset

10.Exit

Enter your choice

1

Enter how many number you want to enter

4

Enter your number 1 :5

Enter your number 2 :6

Enter your number 3 :7

Enter your number 4 :8

1.Insert in the set A

2.Insert in the set B

3.Display

4.Search in the set

5.Union of two set

6.Intersection of two set

7.Minus of two set

8.Remove the element

9.Subset

10.Exit

Enter your choice

2

Enter how many number you want to enter

2

Enter your number 1 :4

Enter your number 2 :5

1.Insert in the set A

2.Insert in the set B

3.Display

4.Search in the set

5.Union of two set

6.Intersection of two set

7.Minus of two set

8.Remove the element

9.Subset

10.Exit

Enter your choice

3

5 6 7 8

4 5

1.Insert in the set A

2.Insert in the set B

3.Display

4.Search in the set

5.Union of two set

6.Intersection of two set

7.Minus of two set

8.Remove the element

9.Subset

10.Exit

Enter your choice

4

Enter the number to be searched

5

Element found!!!!

1.Insert in the set A

2.Insert in the set B

3.Display

4.Search in the set

5.Union of two set

6.Intersection of two set

7.Minus of two set

8.Remove the element

9.Subset

10.Exit

Enter your choice

4

Enter the number to be searched

4

Element found!!!!

1.Insert in the set A

2.Insert in the set B

3.Display

4.Search in the set

5.Union of two set

6.Intersection of two set

7.Minus of two set

8.Remove the element

9.Subset

10.Exit

Enter your choice

5

5 6 7 8 4

1.Insert in the set A

2.Insert in the set B

3.Display

4.Search in the set

5.Union of two set

6.Intersection of two set

7.Minus of two set

8.Remove the element

9.Subset

10.Exit

Enter your choice

6

5

1.Insert in the set A

2.Insert in the set B

3.Display

4.Search in the set

5.Union of two set

6.Intersection of two set

7.Minus of two set

8.Remove the element

9.Subset

10.Exit

Enter your choice

7

6 7 8

1.Insert in the set A

2.Insert in the set B

3.Display

4.Search in the set

5.Union of two set

6.Intersection of two set

7.Minus of two set

8.Remove the element

9.Subset

10.Exit

Enter your choice

8

Enter number to be deleted :

5

Enter number to be deleted :

4

1.Insert in the set A

2.Insert in the set B

3.Display

4.Search in the set

5.Union of two set

6.Intersection of two set

7.Minus of two set

8.Remove the element

9.Subset

10.Exit

Enter your choice

3

6 7 8

5

1.Insert in the set A

2.Insert in the set B

3.Display

4.Search in the set

5.Union of two set

6.Intersection of two set

7.Minus of two set

8.Remove the element

9.Subset

10.Exit

Enter your choice

9

B is not subset of A

1.Insert in the set A

2.Insert in the set B

3.Display

4.Search in the set

5.Union of two set

6.Intersection of two set

7.Minus of two set

8.Remove the element

9.Subset

10.Exit

Enter your choice

10

Practical-3(B-6)

Problem Statement:

Beginning with an empty binary search tree. Construct binary search tree by inserting the values in the order given. After constructing a binary tree -
i. Insert new node

ii. Find number of nodes in longest path from root
iii. Minimum data value found in the tree
iv. Change a tree so that the roles of the left and right pointers are swapped at every node
v. Search a value

Code:

```
#include<iostream>

using namespace std;

class node  // Node declaration for BST.

{

   public:

   int data;

   node * left;

   node * right;

};

class BST

{

   public:

   node * root;

   int cnt;


   BST()

   {

      root = NULL;

      cnt=0;

   }


   void insert();

   void inorder(node *temp);

   void smallest();

   void largest();

   int search(int key);

   void Mirror(node *r);

   int height(node *r);

};

void BST::insert()

{
```

```
node *new_node, *temp;
int flag=0;

new_node = new node();  // allocate the memory.
new_node->left = NULL;
new_node->right = NULL;
cout<<"Enter data : ";
cin>>new_node->data;
if(root == NULL)
{
   root = new_node;
}
else
{
   temp = root;
   while(flag ==0)
   {
      if(new_node->data < temp->data)
      {
         if(temp->left==NULL)
         {
            temp->left = new_node;
            flag=1;
         }
         else
         {
            temp= temp->left;
         }
      }
      else if(new_node->data > temp->data)
      {
         if(temp->right == NULL)
         {
            temp->right = new_node;
            flag =1;
         }
```

```
            else
            {
                temp = temp -> right;
            }
        }
        else
        {
            cout<<"\nData is already exist in the Tree";
            flag++;
        }
    }
  }
}

void BST:: inorder(node * temp)
{
   if(temp != NULL)
   {
      inorder(temp->left);
      cout<<" "<<temp->data;
      cnt++;
      inorder(temp->right);
   }
}
void BST :: smallest()
{
   node *temp;
   temp = root;
   while(temp ->left !=NULL)
   {
      temp = temp ->left;
   }

   cout<<"\nSmallest node in the tree is : "<<temp->data;

}
```

```
void BST :: largest()
{
   node *temp;
   temp = root;
   while(temp ->right !=NULL)
   {
      temp = temp ->right;
   }

   cout<<"\nLargest5 node in the tree is : "<<temp->data;

}

int BST :: search(int key)
{
   node *temp;
   temp = root;

   while(1)
   {
      if(key < temp->data)
      {
         if(temp->left !=NULL)
         {
            temp=temp->left;
         }
         else
         {
            return(0);
         }
      }
      else if(key > temp->data)
      {
         if(temp->right != NULL)
         {
            temp=temp->right;
```

```
            }
            else
            {
                return(0);
            }
        }
        else
        {
            return(1);
        }
    }
}

void BST::Mirror(node *root)
{
    node *temp;
    if(root != NULL)
    {
        temp = root->left;
        root->left = root->right;
        root->right = temp;

        Mirror(root->left);
        Mirror(root->right);

    }
}

int BST::height(node *r)
{
    int Left_Height,Right_Height;
    if(r == NULL)
        return(0);
    if(r->left == NULL && r->right == NULL)
        return(0);
     Left_Height = height(r->left);
```

```
    Right_Height = height(r->right);
    if(Left_Height > Right_Height)
        return(Left_Height +1);
    else
        return(Right_Height +1);

}

int main()
{
    int ch;
    BST B;
    node *test;
    node test2;
    test = new node();
    do{
        cout<<"\n Menu";
        cout<<"\n1. Insert";
        cout<<"\n2. Inorder";
        cout<<"\n3. smallest number";
        cout<<"\n4. Largest number";
        cout<<"\n5. Search";
        cout<<"\n6. Mirror";
        cout<<"\n7. Height of the tree";
        cout<<"\n8. Exit";
        cout<<"\nEnter your choice : ";
        cin>>ch;
        switch(ch)
        {
            case 1 :  //Insert operation.
                B.insert();
            break;
            case 2 :  //Inorder display.
                B.inorder(B.root);
                cout<<"\nCnt = "<<B.cnt;
            break;
```

```
        case 3 :  //smallest number in BST.
            B.smallest();
        break;
        case 4 :  //largest number in BST.
            B.largest();
        break;
        case 5 :  //Search a number in BST.
            int key;
            cout<<"Enter number to Search : ";
            cin>>key;
            if(B.search(key))
            {
                cout<<"number is found";
            }
            else
            {
                cout<<"number is not found";
            }
        break;
        case 6 :  //Mirror image of BST
            cout<<"\nBefore Mirror Image :";
            B.inorder(B.root);
            B.Mirror(B.root);
            cout<<"\nAfter Mirror Image :";
            B.inorder(B.root);
        break;
        case 7 :  //Height of the BST.
            cout<<"Height of the BSt is " <<B.height(B.root);
        break;
        case 8 : cout<<"----Exit----";
            break;
        default :
            cout<<"\nEnter correct choice ";
        break;
    }
}while(ch !=8);
```

```
    return(1);
}
```

OUTPUT:

Menu

1. Insert

2. Inorder

3. smallest number

4. Largest number

5. Search

6. Mirror

7. Height of the tree

8. Exit

Enter your choice : 1

Enter data : 8

Menu

1. Insert

2. Inorder

3. smallest number

4. Largest number

5. Search

6. Mirror

7. Height of the tree

8. Exit

Enter your choice : 1

Enter data : 4

Menu

1. Insert

2. Inorder

3. smallest number

4. Largest number

5. Search

6. Mirror

7. Height of the tree

8. Exit

Enter your choice : 1

Enter data : 9

Menu

1. Insert

2. Inorder

3. smallest number

4. Largest number

5. Search

6. Mirror

7. Height of the tree

8. Exit

Enter your choice : 1

Enter data : 3

Menu

1. Insert

2. Inorder

3. smallest number

4. Largest number

5. Search

6. Mirror

7. Height of the tree

8. Exit

Enter your choice : 2

3 4 8 9

Cnt = 4

Menu

1. Insert

2. Inorder

3. smallest number

4. Largest number

5. Search

6. Mirror

7. Height of the tree

8. Exit

Enter your choice : 3

Smallest node in the tree is : 3

Menu

1. Insert

2. In order

3. smallest number

4. Largest number

5. Search

6. Mirror

7. Height of the tree

8. Exit

Enter your choice : 4

Largest5 node in the tree is : 9

Menu

1. Insert

2. Inorder

3. smallest number

4. Largest number

5. Search

6. Mirror

7. Height of the tree

8. Exit

Enter your choice: 5

Enter number to Search: 9

number is found

Menu

1. Insert

2. Inorder

3. smallest number

4. Largest number

5. Search

6. Mirror

7. Height of the tree

8. Exit

Enter your choice : 5

Enter number to Search : 2

number is not found

Menu

1. Insert

2. Inorder

3. smallest number

4. Largest number

5. Search

6. Mirror

7. Height of the tree

8. Exit

Enter your choice : 6

Before Mirror Image : 3 4 8 9

After Mirror Image : 9 8 4 3

Menu

1. Insert

2. Inorder

3. smallest number

4. Largest number

5. Search

6. Mirror

7. Height of the tree

8. Exit

Enter your choice : 7

Height of the BSt is 2

Menu

1. Insert

2. Inorder

3. smallest number

4. Largest number

5. Search

6. Mirror

7. Height of the tree

8. Exit

Enter your choice : 8

----Exit----

Practical-4(B-7)

Problem Statement:

Construct an expression tree from the given prefix expression eg. +--a*bc/def and traverse it using post order traversal (non recursive) and then delete the entire tree.

Code:

```
#include <iostream>
using namespace std;
#include<string.h>
struct node
{
   char data;
   node *left;
   node *right;
};
class tree
{
   char prefix[20];
   public: node *top;
   void expression(char []);
   void display(node *);
   void non_rec_postorder(node *);
   void del(node *);
};
class stack1
{
   node *data[30];
   int top;
public:
   stack1()
   {
      top=-1;
   }
   int empty()
   {
   if(top==-1)
      return 1;
   return 0;
 }
   void push(node *p)
   {
      data[++top]=p;
   }
   node *pop()
   {
      return(data[top--]);
   }
 };

 void tree::expression(char prefix[])
 {
    char c;
   stack1 s;
   node *t1,*t2;
   int len,i;
   len=strlen(prefix);
   for(i=len-1;i>=0;i--)
   {
      top=new node;
      top->left=NULL;
      top->right=NULL;
      if(isalpha(prefix[i]))
      {
         top->data=prefix[i];
```

```
            s.push(top);
        }
        else if(prefix[i]=='+'||prefix[i]=='*'||prefix[i]=='-'||prefix[i]=='/')
        {
            t2=s.pop();
            t1=s.pop();
            top->data=prefix[i];
            top->left=t2;
            top->right=t1;
            s.push(top);
        }
    }
    top=s.pop();
}
void tree::display(node * root)
{
    if(root!=NULL)
    {
        cout<<" Given Expression is –"<<root->data;
        display(root->left);
        display(root->right);
    }
}
void tree::non_rec_postorder(node *top)
{
    stack1 s1,s2;
    node *T=top;
    cout<< "\n Postfix Expression is -";
    s1.push(T);
    while(!s1.empty())
    {
        T=s1.pop();
        s2.push(T);
        if(T->left!=NULL)
        s1.push(T->left);
        if(T->right!=NULL)
            s1.push(T->right);
    }
    while(!s2.empty())
    {
        top=s2.pop();
        cout<<top->data;
    }
}
void tree::del(node* node)
{
    if (node == NULL)
    return;
    del(node->left);
    del(node->right);
    cout<<"\n Deleting node:"<<node->data;
    free(node);
}
int main()
{
    char expr[20];
    tree t;
    cout<<"Enter prefix Expression: ";
    cin>>expr;
    cout<<expr;
```

```
    t.expression(expr);
    t.non_rec_postorder(t.top);
    t.del(t.top);
 }
```

OUTPUT:

Enter prefix Expression: +--a*bc/def
+--a*bc/def
Postfix Expression is -abc*-de/-f+
Deleting node:a
Deleting node:b
Deleting node:c
Deleting node:*
Deleting node:-
Deleting node:d
Deleting node:e
Deleting node:/
Deleting node:-
Deleting node:f
Deleting node:+

Practical-5(B-10)

Problem Statement:

Consider threading a binary tree using preorder threads rather than inorder threads. Design an algorithm for traversal without using stack and analyze its complexity.

Code:

```
using namespace std;
#include<iostream>

class Tnode
{
   public:
      int data;
      Tnode *left;
      Tnode *right;
      int lbit, rbit;
};

class TBT
{
   Tnode *root,*head;
   public:
   TBT()
   {
      root = NULL;
      head = NULL;
   }
   void createTBT();
   void preorder();

};

void TBT::preorder() // preorder traversal of TBT.
{
   Tnode *temp;
   temp = root;
   int flag =0;

   if(root == NULL)
   {
      cout<<"\nTree is empty";
   }
   else
   {
      while(temp != head)
      {
         if(flag ==0)
         {
            cout<<" "<<temp ->data;
         }
         if(temp->lbit==1 && flag ==0)
         {
            temp = temp ->left;

         }
         else
         {
            if(temp ->rbit == 1)
            {
               temp = temp ->right;
               flag=0;
            }
            else
```

```
            {
               temp = temp->right;
               flag =1;
            }
         }

      }
   }
}


void TBT::createTBT()
{
   int flag = 0;
   char ans;
   Tnode *new_node, *temp;
   head = new Tnode(); // allocation of memory for head.
   head ->data = -1;
   head ->left = head;
   head ->right = head;
   head ->lbit = 0;
   head ->rbit = 0;
   root =  new Tnode(); // allocation of memory for root.
   cout<<"Enter root node : ";
   cin>> root->data;
   head ->left = root;
   head -> lbit = 1;
   root ->left = head;
   root ->right = head;
   root ->lbit = 0;
   root ->rbit = 0;

   do
   {
      new_node = new Tnode();
      cout<<"\nEnter new node : ";
      cin>>new_node ->data;
      new_node ->lbit = 0;
      new_node ->rbit = 0;
      temp = root;
      flag = 0;
      while(flag ==0)  // find proper place for new node.
      {
         if(new_node -> data < temp -> data)
         {
            if(temp ->lbit == 0)
            {
               new_node ->left = temp ->left;
               temp ->left = new_node;
               temp->lbit=1;
               new_node ->right = temp;
               flag = 1;
            }
            else
            {
               temp = temp ->left;
            }
         }
         else if(new_node -> data > temp ->data)
         {
```

```
            if(temp ->rbit ==0)
            {
               new_node ->right = temp ->right;
               temp ->right  = new_node;
               temp ->rbit = 1;
               new_node ->left = temp;
               flag =1;
            }
            else
            {
               temp=temp->right;
            }
         }
         else
         {
            cout<<"\n Data is already exist";
         }

      }

      cout<<"\nDo you want to continue : ";
      cin>>ans;
   }while(ans=='Y' || ans=='y');
}

int main()
{
   TBT T;
   T.createTBT();

 // Preorder Display.
   T.preorder();

   return 0;
}
```

OUTPUT:

```
Enter root node : 10
Enter new node : 5
Do you want to continue : y
Enter new node : 15
Do you want to continue : y
Enter new node : 2
Do you want to continue : y
Enter new node : 8
Do you want to continue : y
Enter new node : 12
Do you want to continue : y
Enter new node : 20
Do you want to continue : n
 10 5 2 8 15 12 20
```

Practical-6(C-13)

Problem Statement:

Represent a given graph using adjacency matrix/list to perform DFS and using adjacency list to perform BFS. Use the map of the area around the college as the graph. Identify the prominent land marks as nodes and perform DFS and BFS on that.

Code:

```
#include<iostream>
using namespace std;
#define SIZE 10
#define MAX 20

class Queue
{
   int front,rear;
   int arr[MAX];

   public:
   Queue()
   {
      front=rear=-1;
   }
   void insert_ele(int x)
   {
      if(!isFull())
      {
         if(front==-1)
            front++;
         rear++;
         arr[rear]=x;
      }
   }
   int delete_ele()
   {
      if(!isEmpty())
      {
         int x=arr[front];
         for(int i=front;i<rear;i++)
         {
            arr[i]=arr[i+1];
         }
         if(front==rear)
            front--;
         rear--;
         return x;
      }
   }
   bool isEmpty()
   {
      if(rear==-1)
         return true;
      else
         return false;
   }
   bool isFull()
   {
      if(rear==MAX-1)
         return true;
      else
         return false;
   }
   void display_queue()
   {
      if(!isEmpty())
```

```
        {
            cout<<"\nQueue : ";
            for(int i=front;i<=rear;i++)
                cout<<arr[i]<<" ";
        }

    }
};

class Stack
{
    int arr[MAX];
    int top;
    public:
    Stack()
    {
        top=-1;
    }
    void push(int x)
    {
        if(!isFull())
        {
            top++;
            arr[top]=x;
        }
    }
    int pop()
    {
        if(!isEmpty())
        {
            int x=arr[top];
            top--;
            return x;
        }
    }
    bool isEmpty()
    {
        if(top==-1)
            return true;
        else
            return false;
    }
    bool isFull()
    {
        if(top==MAX-1)
            return true;
        else
            return false;
    }
    void display_stack()
    {
        cout<<"\nStack : ";
        for(int i=top;i>=0;i--)
            cout<<arr[i]<<" ";
    }
};

class Graph
{
```

```
int i,j;
int visited_arr[20];

public:

int cnt;
int adj_mat[SIZE][SIZE];
int ver_arr[SIZE];
int vertex_count;

Graph()
{
   cnt=0;
}

void DFS();
void BFS();

void display(string s)
{
   cout<<"\n"<<s<<" Traversal : ";
   for(i=0;i<cnt;i++)
   {
      cout<<visited_arr[i]<<" ";
   }
}

int search(int x)
{
   for(i=0;i<cnt;i++)
   {
      if(visited_arr[i]==x)
         return 1;
   }
   return 0;
}

void create_adjmat()
{
   cout<<"\nEnter the total number of nodes in the graph (less than 10) : ";
   cin>>vertex_count;
   cout<<"\nEnter the values of the nodes :\n";
   for(i=1;i<=vertex_count;i++)
   {
      cin>>ver_arr[i];
   }
   cout<<"\nEnter 1 if edge present between nodes else enter 0\n";
   for(i=1;i<=vertex_count;i++)
   {
      for(j=1;j<=vertex_count;j++)
      {
         cout<<"Node "<<ver_arr[i]<<" and Node "<<ver_arr[j]<<" : ";
         cin>>adj_mat[i][j];
      }
   }
}
void display_adjmat()
{
   cout<<"\nAdjacency Matrix \n\n  ";
   for(i=1;i<=vertex_count;i++)
```

```
            cout<<ver_arr[i]<<" ";
        cout<<"\n";
        for(i=1;i<=vertex_count;i++)
        {
            cout<<ver_arr[i]<<" ";
            for(j=1;j<=vertex_count;j++)
            {
                cout<<adj_mat[i][j]<<" ";
            }
            cout<<"\n";
        }
    }
};

void Graph :: DFS()
{
    int curr_ver;
    Stack sobj;
    int i,j,flag=0;
    cout<<"Enter starting vertex : ";
    cin>>curr_ver;
    cnt=0;
    sobj.push(curr_ver);
    visited_arr[cnt++]=curr_ver;
    curr_ver = sobj.pop();
    for(i=curr_ver;i<=vertex_count;)
    {
        if(flag ==0)
        {
            for(j=1;j<=vertex_count;j++)
            {

                if(adj_mat[i][j]==1)
                {
                    sobj.push(j);
                }
            }
            flag=1;
        }
        curr_ver = sobj.pop();

        if(search(curr_ver)==0)
        {
            visited_arr[cnt++]=curr_ver;
            flag=0;
        }

        i=curr_ver;
        if(cnt == vertex_count)
        {
            break;
        }

    }

    display("DFS");
}

void Graph :: BFS()
{
```

```
    int curr_ver;
    Queue qobj;
    int i,j,flag=0;
    cout<<"Enter starting vertex : ";
    cin>>curr_ver;
    cnt=0;
    qobj.insert_ele(curr_ver);
    visited_arr[cnt++]=curr_ver;
    curr_ver = qobj.delete_ele();
    for(i=curr_ver;i<=vertex_count;)
    {
       if(flag ==0)
       {
          for(j=1;j<=vertex_count;j++)
          {

             if(adj_mat[i][j]==1)
             {
                qobj.insert_ele(j);
             }
          }
          flag=1;
       }
       curr_ver = qobj.delete_ele();

       if(search(curr_ver)==0)
       {
          visited_arr[cnt++]=curr_ver;
          flag=0;
       }

       i=curr_ver;
       if(cnt == vertex_count)
       {
          break;
       }

    }

    display("BFS");
}

int main()
{
    Graph gobj;
    int choice;

    do
    {
       cout<<"\n\t\t\tMENU";
       cout<<"\n\t1. Create Matrix\n\t2. Display Matrix\n\t3. Perform DFS Traversal\n\t4. Perform BFS
Traversal\n\t5. Exit\n";
       cout<<"\n Enter your choice : ";
       cin>>choice;

       switch(choice)
       {
          case 1: gobj.create_adjmat();
                  break;
          case 2: gobj.display_adjmat();
```

```
            break;
        case 3: gobj.DFS();
            break;
        case 4: gobj.BFS();
            break;
        case 5: cout<<"Exited Code";
             break;
        default: cout<<"Invalid Option Chosen!";
      }
   }while(choice!=5);
   return 0;
}
```

OUTPUT:

MENU

1. Create Matrix
2. Display Matrix
3. Perform DFS Traversal
4. Perform BFS Traversal
5. Exit

Enter your choice : 1
Enter the total number of nodes in the graph (less than 10) : 3
Enter the values of the nodes :
1
2
3
Enter 1 if edge present between nodes else enter 0
Node 1 and Node 1 : 0
Node 1 and Node 2 : 1
Node 1 and Node 3 : 1
Node 2 and Node 1 : 1
Node 2 and Node 2 : 0
Node 2 and Node 3 : 1
Node 3 and Node 1 : 1
Node 3 and Node 2 : 1
Node 3 and Node 3 : 0
MENU

1. Create Matrix
2. Display Matrix
3. Perform DFS Traversal
4. Perform BFS Traversal
5. Exit

Enter your choice : 2
Adjacency Matrix

1 2 3
1 0 1 1
2 1 0 1
3 1 1 0

MENU

1. Create Matrix
2. Display Matrix
3. Perform DFS Traversal
4. Perform BFS Traversal
5. Exit

Enter your choice : 3
Enter starting vertex : 1
DFS Traversal : 1 3 2

MENU

1. Create Matrix
2. Display Matrix
3. Perform DFS Traversal
4. Perform BFS Traversal
5. Exit

Enter your choice : 4
Enter starting vertex : 1
BFS Traversal : 1 2 3

MENU

1. Create Matrix
2. Display Matrix
3. Perform DFS Traversal
4. Perform BFS Traversal
5. Exit

Enter your choice : 5

Practical-7(C-15)

Problem Statement:
You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of

lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.

Code:

```
#include<iostream>
using namespace std;
class snode
{
  public:       // data structure for sparse matrix.
   char u1,u2;
   int wt;
};
class test
{
              int n,m,x;
              snode arr[10],res[10];
public:
             test()
             {
                          n=0;
                          m=1;
             }


     void inputsparse();
     void displaysparse();
     void bsort();
     void kruskals();
     void dispmst();
};

 // Function to Display result
void test::dispmst()
{
    for(int i=0;i<m;i++)
     {
        cout<<res[i].u1<<" "<<res[i].u2<<" "<<res[i].wt<<endl;
     }
}

// Function to find minimum spanning tree.
 void test::kruskals()
{
             int cnt=0;
             int flag1,flag2,i;
     res[0]=arr[0];
     m=1;
    cnt=1;

             do
             {
         for(i=1; i<n  ; i++) //arr
         {
           flag1=0;
           flag2=0;
                    for(int j=0;j<m;j++)
                     {
                              if((arr[i].u1==res[j].u1 || arr[i].u1==res[j].u2 ) && flag1==0)
                              {
```

```
                        flag1++;
                    }
                    if((arr[i].u2==res[j].u1 || arr[i].u2==res[j].u2 ) && flag2==0)
                    {
                        flag2++;
                    }
                        }

                        if(flag1!=1  ^ flag2 !=1) // ^ Exore operation.
                        {

                                        res[m++]=arr[i];

                        }


            }
            cnt = x-1;
      }while(m!=cnt)       ;
}


// Function tio Read input graph.
 void test::inputsparse()
 {

 cout<<"ENTER NO OF EDGES: ";
 cin>>n;
 cout<<"ENTER NO OF VERTICES: ";
 cin>>x;
  for(int i=0;i<n;i++)
  {
            cout<<"ENTER 1ST VERTEX: ";
            cin>>arr[i].u1;
            cout<<"ENTER 2ND VERTEX: ";
            cin>>arr[i].u2;
            cout<<"ENTER WEIGHT: ";
            cin>>arr[i].wt;
            cout<<endl;
    }
    bsort();
 }

 void test::displaysparse()
{
      for(int i=0;i<n;i++)
    {
       cout<<arr[i].u1<<" "<<arr[i].u2<<" "<<arr[i].wt<<endl;
    }
}


//Sort the given edges of the graph using bubble sort
void test::bsort()
{
  snode temp;
    for(int i=0;i<n-1;i++)
    {
         for(int j=0;j<n-1-i;j++)
         {
```

```
            if(arr[j].wt>arr[j+1].wt)
            {
               temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
             }
         }
    }

}


int main()
{
  test obj;
  obj.inputsparse();
  obj.displaysparse();
  obj.kruskals();
  cout<<"RESULT:"<<endl;
  obj.dispmst();
return 0;
}
```

OUTPUT:

ENTER NO OF EDGES: 3
ENTER NO OF VERTICES: 4
ENTER 1ST VERTEX: 1
ENTER 2ND VERTEX: 2
ENTER WEIGHT: 4
ENTER 1ST VERTEX: 2
ENTER 2ND VERTEX: 3
ENTER WEIGHT: 6
ENTER 1ST VERTEX: 3
ENTER 2ND VERTEX: 4
ENTER WEIGHT: 2
3 4 2
1 2 4
2 3 6
RESULT:
3 4 2
2 3 6
1 2 4

Practical-8(D-18)

Problem Statement:
Given sequence k = k1 <k2 < … <kn of n sorted keys, with a search probability pi for each key ki . Build the Binary search tree that has the least search cost given the access probability for each key?

Code:

```
#include<iostream>
using namespace std;
#define SIZE 10
class OBST
{
int p[SIZE];  // Probabilities with which we search for an element
int q[SIZE]; //Probabilities that an element is not found
int a[SIZE]; //Elements from which OBST is to be built
int w[SIZE][SIZE]; //Weight‘w[i][j]’of a tree having root’r[i][j]’
int c[SIZE][SIZE]; //Cost ‘c[i][j] of a tree having root ‘r[i][j]
int r[SIZE][SIZE]; //represents root
int n;            // number of nodes
public:

void get_data()
{
int i;
cout<<"\n Optimal Binary Search Tree:- \n";
cout<<"\n Enter the number of nodes:-";
cin>>n;
cout<<"\n Enter the data as…..\n";
for(i=1;i<=n;i++)
{
cout<<"\n a["<<i<<"]-";
cin>>a[i];
}
for(i=1;i<=n;i++)
{
cout<<"\n p["<<i<<"]-";
cin>>p[i];
}
for(i=0;i<=n;i++)
{
cout<<"\n q["<<i<<"]-";
cin>>q[i];
}
}
/* This function returns a value in the range ‘r[i][j-1]’ to ‘r[i+1][j]’so that the cost ‘c[i][k-1]+c[k][j]’is minimum
*/

int Min_Value(int i,int j)
{
int m,k;
int minimum=32000;
for(m=r[i][j-1];m<=r[i+1][j];m++)
{
if((c[i][m-1]+c[m][j])<minimum)
{
 minimum=c[i][m-1]+c[m][j];
 k=m;
}
}
return k;
}
/* This function builds the table from all the given probabilities It basically computes C,r,W values */

void build_OBST()
{
```

```
 int i,j,k,l,m;
 for(i=0;i<n;i++)
 {
                    //initialize
w[i][i]=q[i];
r[i][i]=c[i][i]=0;
                    //Optimal trees with one node
w[i][i+1]=q[i]+q[i+1]+p[i+1];
r[i][i+1]=i+1;
c[i][i+1]=q[i]+q[i+1]+p[i+1];
 }
 w[n][n]=q[n];
 r[n][n]=c[n][n]=0;
                    //Find optimal trees with 'm' nodes
 for(m=2;m<=n;m++)
 {
for(i=0;i<=n-m;i++)
{
 j=i+m;
 w[i][j]=w[i][j-1]+p[j]+q[j];
 k=Min_Value(i,j);
 c[i][j]=w[i][j]+c[i][k-1]+c[k][j];
 r[i][j]=k;
}
 }
}
/* This function builds the tree from the tables made by the OBST function */
void build_tree()
{
 int i,j,k;
 int queue[20],front=-1,rear=-1;
 cout<<"The Optimal Binary Search Tree For the Given Node Is….\n";
 cout<<"\n The Root of this OBST is ::"<<r[0][n];
 cout<<"\nThe Cost of this OBST is::"<<c[0][n];
 cout<<"\n\n\t NODE \t LEFT CHILD \t RIGHT CHILD ";
 cout<<"\n";
 queue[++rear]=0;
 queue[++rear]=n;
 while(front!=rear)
 {
     i=queue[++front];
     j=queue[++front];
     k=r[i][j];
    cout<<"\n\t"<<k;
 if(r[i][k-1]!=0)
  {
    cout<<"\t\t"<<r[i][k-1];
    queue[++rear]=i;
    queue[++rear]=k-1;
  }
 else
   cout<<"\t\t";
 if(r[k][j]!=0)
  {
   cout<<"\t"<<r[k][j];
   queue[++rear]=k;
   queue[++rear]=j;
  }
 else
    cout<<"\t";
```

```
 }          //end of while
   cout<<"\n";
}
};     //end of the class
int main()
{
OBST obj;
obj.get_data();
obj.build_OBST();
obj.build_tree();
return 0;
}
```

OUTPUT:

Optimal Binary Search Tree
Enter the number of nodes:- 4
Enter the data as…
a[1]-1
a[2]-2
a[3]-3
a[4]-4
p[1]-3
p[2]-3
p[3]-1
p[4]-1
q[0]-2
q[1]-3
q[2]-1
q[3]-1
q[4]-1
The Optimal Binary Search Tree For the Given Node Is…
The Root of this OBST is ::2
The Cost of this OBST is::32
NODE LEFT CHILD RIGHT CHILD
2 1 3
1
3 4
4

Practical-9(D-19)

Problem Statement:
A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data

sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword.

Code:

```
#include<iostream>
#include<string>
using namespace std;
class dictionary;
class avlnode
{
        string keyword;
        string meaning;
        avlnode *left,*right;
        int bf;
        public:
        avlnode()
        {
                keyword='\0';
                meaning='\0';
                left=right=NULL;
                bf=0;
        }
        avlnode(string k,string m)
        {
                keyword=k;
                meaning=m;
                left=right=NULL;
                bf=0;
        }
friend class dictionary;
};

class dictionary
{
        avlnode *par,*loc;
        public:
        avlnode *root;
        dictionary()
        {
                root=NULL;
                par=loc=NULL;
        }
        void accept();
        void insert(string key,string mean);
        void LLrotation(avlnode*,avlnode*);
        void RRrotation(avlnode*,avlnode*);
        void inorder(avlnode *root);
        void deletekey(string key);
        void descending(avlnode *);
        void search(string);
        void update(string,string);

};
void dictionary::descending(avlnode *root)
{
        if(root)
        {
                descending(root->right);
                cout<<root->keyword<<" "<<root->meaning<<endl;
```

```
                descending(root->left);
        }
}

void dictionary::accept()
{
        string key,mean;
        cout<<"Enter keyword:- "<<endl;
        cin>>key;
        cout<<"Enter meaning:- "<<endl;
        cin>>mean;
        insert(key,mean);
}
void dictionary::LLrotation(avlnode *a,avlnode *b)
{
        cout<<"LL rotation:-"<<endl;
        a->left=b->right;
        b->right=a;
        a->bf=b->bf=0;
}

void dictionary::RRrotation(avlnode *a,avlnode *b)
{
        cout<<"RR rotation:-"<<endl;
        a->right=b->left;
        b->left=a;
        a->bf=b->bf=0;
}

void dictionary::insert(string key,string mean)
{

        if(!root)
        {
                //create new root
                root=new avlnode(key,mean);
                cout<<"ROOT CREATED \n";
                return;
        }


                avlnode *a,*pa,*p,*pp;

                pa=NULL;
                p=a=root;
                pp=NULL;

                while(p)
                {
                        cout<<"In first while \n";
                        if(p->bf)
                        {
                        a=p;
                        pa=pp;
                        }
                        if(key<p->keyword){pp=p;p=p->left;}    //takes the left branch
                        else if(key>p->keyword){pp=p;p=p->right;} //right branch
                        else
                        {
                                //p->meaning=mean;
```

```
                cout<<"Already exist \n";
                return;
        }
}
cout<<"Outside while \n";
avlnode *y=new avlnode(key,mean);
if(key<pp->keyword)
{
        pp->left=y;
}
else
        pp->right=y;
cout<<"KEY INSERTED \n";

int d;
avlnode *b,*c;
//a=pp;
b=c=NULL;
if(key>a->keyword)
{
        cout<<"KEY >A->KEYWORD \n";
        b=p=a->right;
        d=-1;
        cout<<" RIGHT HEAVY \n";
}
else
{
        cout<<"KEY < A->KEYWORD \n";
        b=p=a->left;
        d=1;
        cout<<" LEFT HEAVY \n";
}

while(p!=y)
{
        if(key>p->keyword)
        {
                p->bf=-1;
                p=p->right;

        }
        else
        {
                p->bf=1;
                p=p->left;
        }

}
cout<<" DONE ADJUSTING INTERMEDIATE NODES \n";
if(!(a->bf)||!(a->bf+d))
{
        a->bf+=d;
        return;
}
//else
//{
if(d==1)
{
        //left heavy
        if(b->bf==1)
```

```
		{
			LLrotation(a,b);
			/*a->left=b->right;
			b->right=a;
			a->bf=0;
			b->bf=0;*/
		}
		else //if(b->bf==-1)
		{

			cout<<"LR rotation"<<endl;
			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;
					b->bf=0;
					break;
				}
				case -1:
				{
					a->bf=0;
					b->bf=1;
					break;
				}

				case 0:
				{
					a->bf=0;
					b->bf=0;
					break;
				}

			}
			c->bf=0;
			b=c;   //b is new root


		}


}

if(d==-1)
{
		if(b->bf==-1)
		{

			RRrotation(a,b);
		}
		else
		{
			c=b->left;

			a->right=c->left;
```

```
                                        b->left=c->right;
                                        c->left=a;
                                        c->right=b;
                                        switch(c->bf)
                                        {
                                                case 1:
                                                {
                                                        a->bf=0;
                                                        b->bf=-1;
                                                        break;
                                                }
                                                case -1:
                                                {
                                                        a->bf=1;
                                                        b->bf=0;
                                                        break;
                                                }

                                                case 0:
                                                {
                                                        a->bf=0;
                                                        b->bf=0;
                                                        break;
                                                }

                                        }
                                        c->bf=0;
                                        b=c;  //b is new root

                                }


                        }

                        if(!pa)
                                root=b;
                        else if(a==pa->left)
                                pa->left=b;
                        else
                                pa->right=b;
                        cout<<"AVL tree created!! \n";




}
void dictionary::search(string key)
{
        cout<<"ENTER SEARCH \n";
        loc=NULL;
        par=NULL;
        if(root==NULL)
        {
                cout<<"Tree not created  "<<endl;
                //          root=key;
                loc=NULL;
                par=NULL;
        }
```

```
	avlnode *ptr;
	ptr=root;
	while(ptr!=NULL)
	{
		if(ptr->keyword==key)
		{


			loc=ptr;
			break;
		}
		else if(key<ptr->keyword)
		{
			par=ptr;
			ptr=ptr->left;
		}

		else
		{
			par=ptr;
			ptr=ptr->right;

		}
	}

	if(loc==NULL)
	{
		cout<<"Not found "<<endl;
	}

}

void dictionary::update(string oldkey,string newmean)
{
	search(oldkey);
	loc->meaning=newmean;
	cout<<"UPDATE SUCCESSFULLY \n";
}
void dictionary::deletekey(string key)
{

}
void dictionary::inorder(avlnode *root)
{
	if(root)
	{
		inorder(root->left);
		cout<<root->keyword<<" "<<root->meaning<<endl;
		inorder(root->right);
	}
}

int main()
{
	string k,m;
	dictionary d;
	int ch;
	string key,mean;

	do
```

```
	{
	cout<<"*******MENU*****\n1.Insert \n2.Update \n3.Ascending \n4.Descending \n5.Display
\n6.Quit \n";
	cout<<"Enter yourChoice :-";
	cin>>ch;
	switch(ch)
	{
		case 1:
		{
			d.accept();
			break;
		}
		case 2:
		{
			cout<<"Enter key whose meaning to update:- \n";
			cin>>key;
			cout<<"Enter new meaning:-\n";
			cin>>mean;
			d.update(key,mean);
			break;
		}
		case 3:
			d.inorder(d.root);
			break;

		case 4:
			cout<<"Descending \n";
			d.descending(d.root);
			break;

		case 5:
			d.inorder(d.root);
			break;
		default:
			break;
	}
	}while(ch!=6);



return 0;
}
```

OUTPUT

```
*******MENU*****
1.Insert
2.Update
3.Ascending
4.Descending
5.Display
6.Quit
Enter your Choice :-1
Enter keyword:-
20
Enter meaning:-
1
ROOT CREATED
*******MENU*****
```

1.Insert
2.Update
3.Ascending
4.Descending
5.Display
6.Quit
Enter your Choice :-5
20 1
*******MENU*****
1.Insert
2.Update
3.Ascending
4.Descending
5.Display
6.Quit
Enter your Choice :-2
Enter key whose meaning to update:-
20
Enter new meaning:-
0
ENTER SEARCH
UPDATE SUCCESSFULLY
*******MENU*****
1.Insert
2.Update
3.Ascending
4.Descending
5.Display
6.Quit
Enter your Choice :-5
20 0
*******MENU*****
1.Insert
2.Update
3.Ascending
4.Descending
5.Display
6.Quit
Enter your Choice :-1
Enter keyword:-
10
Enter meaning:-
2
In first while
Outside while
KEY INSERTED
KEY < A->KEYWORD
LEFT HEAVY
DONE ADJUSTING INTERMEDIATE NODES
*******MENU*****
1.Insert
2.Update
3.Ascending
4.Descending
5.Display
6.Quit
Enter your Choice :-5
10 2
20 0
*******MENU*****

1.Insert
2.Update
3.Ascending
4.Descending
5.Display
6.Quit
Enter your Choice :-3
10 2
20 0
*******MENU*****
1.Insert
2.Update
3.Ascending
4.Descending
5.Display
6.Quit
Enter your Choice :-4
Descending
20 0
10 2
*******MENU*****
1.Insert
2.Update
3.Ascending
4.Descending
5.Display
6.Quit
Enter your Choice :-6

Practical-10(E-21)

Problem Statement:

Implement the Heap/Shell sort algorithm implemented in Java demonstrating heap/shell data structure with modularity of programming language

Code:

```
import java.util.Arrays;
import java.util.Scanner;
public class Shellsort
{
void shellSort(int array[], int n)
  {
  for (int interval = n / 2; interval > 0; interval /= 2)
  {
    for (int i = interval; i < n; i += 1)
    {
    int temp = array[i];
    int j;
    for (j = i; j >= interval && array[j - interval] > temp; j -= interval)
    {
      array[j] = array[j - interval];
    }
    array[j] = temp;
    }
  }
  }

  public static void main(String args[])
  {
  Scanner sc=new Scanner(System.in);
  System.out.println("Enter the size of the array: ");
int arr_size = 0;
if (sc.hasNextInt())
{
arr_size = sc.nextInt();
}
int[] arr = new int[arr_size];
System.out.println("Enter the elements of the array: ");
for (int i = 0; i < arr_size; i++)
{
if (sc.hasNextInt())
{
arr[i] = sc.nextInt();
}
}
Shellsort ss = new Shellsort();
  ss.shellSort(arr, arr_size);
  System.out.println("Sorted Array in Ascending Order: ");
System.out.println("The elements of the array are: ");
for (int i = 0; i < arr_size; i++)
{
System.out.print(arr[i] + " ");
}
sc.close();
  }
}
```

OUTPUT:

Enter the size of the array: 4
Enter the elements of the array:
12
9
8
7
Sorted Array in Ascending Order:
The elements of the array are:
7 8 9 12

Problem Statement:
Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to main the data.

Code:

```
#include<iostream>
#include<fstream>
#include<string.h>
using namespace std;

struct stud
{
   int roll;
   char name[10];
   char div;
   char add[10];
}rec;
class student
{
   public:

   void create();
   void display();
   int search();
   void Delete();
};
void student::create()
 {
   char ans;
   ofstream fout;
   fout.open("stud.dat",ios::out|ios::binary);
   do
   {
      cout<<"\n\tEnter Roll No of Student : ";
      cin>>rec.roll;
      cout<<"\n\tEnter a Name of Student : ";
      cin>>rec.name;
      cout<<"\n\tEnter a Division of Student : ";
      cin>>rec.div;
      cout<<"\n\tEnter a Address of Student : ";
      cin>>rec.add;
      fout.write((char *)&rec,sizeof(stud))<<flush;
      cout<<"\n\tDo You Want to Add More Records: ";
      cin>>ans;
   }while(ans=='y'||ans=='Y');
   fout.close();
}
void student::display()
{
   ifstream fin;
   fin.open("stud.dat",ios::in|ios::binary);
   fin.seekg(0,ios::beg);
   cout<<"\n\tThe Content of File are:\n";
   cout<<"\n\tRoll\tName\tDiv\tAddress";
   while(fin.read((char *)&rec,sizeof(stud)))
   {
      if(rec.roll!=-1)
```

```
        cout<<"\n\t"<<rec.roll<<"\t"<<rec.name<<"\t"<<rec.div<<"\t"<<rec.add;

    }
        fin.close();
}

int student::search()
{
    int r,i=0;
    ifstream fin;
    fin.open("stud.dat",ios::in|ios::binary);
    fin.seekg(0,ios::beg);
    cout<<"\n\tEnter a Roll No: ";
    cin>>r;

    while(fin.read((char *)&rec,sizeof(stud)))
    {
        if(rec.roll==r)
        {
            cout<<"\n\tRecord Found...\n";
            cout<<"\n\tRoll\tName\tDiv\tAddress";
            cout<<"\n\t"<<rec.roll<<"\t"<<rec.name<<"\t"<<rec.div<<"\t"<<rec.add;
            return i;
        }
        i++;
    }
    fin.close();
    return 0;
}
void student::Delete()
{
    int pos;
    pos=search();
    fstream f;
    f.open("stud.dat",ios::in|ios::out|ios::binary);
    f.seekg(0,ios::beg);
    if(pos==0)
    {
        cout<<"\n\tRecord Not Found";
        return;
    }
    int offset=pos*sizeof(stud);
    f.seekp(offset);
    rec.roll=-1;
    strcpy(rec.name,"NULL");
    rec.div='N';
    strcpy(rec.add,"NULL");
    f.write((char *)&rec,sizeof(stud));
    f.seekg(0);
    f.close();
    cout<<"\n\tRecord Deleted";
 }
int main()
 {
student obj;
int ch,key;
char ans;
do
 {
cout<<"\n\t***** Student Information *****";
```

```
cout<<"\n\t1. Create\n\t2. Display\n\t3. Delete\n\t4.  Search\n\t5. Exit";
cout<<"\n\t..... Enter Your Choice: ";
cin>>ch;
switch(ch)
 {
case 1: obj.create();
break;
case 2: obj.display();
break;
case 3: obj.Delete();
break;
case 4: key=obj.search();
if(key==-1)
 cout<<"\n\tRecord Not Found...\n";
break;
case 5:
break;
 }
cout<<"\n\t..... Do You Want to Continue in Main Menu: "; cin>>ans;
 }while(ans=='y'||ans=='Y');
return 1;
 }
```

OUTPUT :-

******* Student Information *******
1. Create
2. Display
3. Delete
4. Search
5. Exit
..... Enter Your Choice: 1

Enter Roll No of Student : 45

Enter a Name of Student : Kasturi

Enter a Division of Student : A

Enter a Address of Student : Nashik

Do You Want to Add More Records: y

Enter Roll No of Student : 28

Enter a Name of Student : Samidha

Enter a Division of Student : A

Enter a Address of Student : Nashik

Do You Want to Add More Records: n

..... Do You Want to Continue in Main Menu: y

******* Student Information *******
1. Create
2. Display
3. Delete
4. Search

5. Exit
..... Enter Your Choice: 2

The Content of File are:

Roll	Name	Div	Address
28	Samidha	A	Nashik
45	Kasturi	A	Nashik

..... Do You Want to Continue in Main Menu: y

******* Student Information *******
1. Create
2. Display
3. Delete
4. Search
5. Exit
..... Enter Your Choice: 4

Enter a Roll No: 45

Record Found...

Roll	Name	Div	Address
45	Kasturi	A	Nashik

..... Do You Want to Continue in Main Menu: y

******* Student Information *******
1. Create
2. Display
3. Delete
4. Search
5. Exit
..... Enter Your Choice: 3

Enter a Roll No: 28

Record Found...

Roll	Name	Div	Address
28	Samidha	A	Nashik

Record Deleted
..... Do You Want to Continue in Main Menu: y

******* Student Information *******
1. Create
2. Display
3. Delete
4. Search
5. Exit
..... Enter Your Choice: 2

The Content of File are:

Roll	Name	Div	Address
45	Kasturi	A	Nashik

..... Do You Want to Continue in Main Menu: n

Problem Statement:
Assume we have two input and two output tapes to perform the sorting. The internal memory can hold and sort m records at a time. Write a program in java for external sorting. Find out time complexity.

Code:

```
import java.util.Scanner;
class MergeSort
{
 void merge(int arr[], int p, int q, int r)
   {
    int n1 = q - p + 1;
    int n2 = r - q;
    int L[] = new int[n1];
    int M[] = new int[n2];

    for (int i = 0; i < n1; i++)
      L[i] = arr[p + i];
    for (int j = 0; j < n2; j++)
      M[j] = arr[q + 1 + j];
    int i, j, k;
    i = 0;
    j = 0;
    k = p;
    while (i < n1 && j < n2)
    {
      if (L[i] <= M[j])
      {
        arr[k] = L[i];
        i++;
      } else
      {
        arr[k] = M[j];
        j++;
      }
      k++;
    }


    while (i < n1)
    {
      arr[k] = L[i];
      i++;
      k++;
    }

    while (j < n2)
    {
      arr[k] = M[j];
      j++;
      k++;
    }
  }


  void mergeSort(int arr[], int l, int r)
  {
    if (l < r)
    {
      int m = (l + r) / 2;
```

```
      mergeSort(arr, l, m);
      mergeSort(arr, m + 1, r);
      merge(arr, l, m, r);
    }
  }
  static void printArray(int arr[])
  {
    int n = arr.length;
    for (int i = 0; i < n; ++i)
      System.out.print(arr[i] + " ");
    System.out.println();
  }

  public static void main(String args[])
  {
  Scanner sc = new Scanner(System.in);
// Take the array size from the user
System.out.println("Enter the size of the array: ");
int arr_size = 0;
if (sc.hasNextInt()) {
arr_size = sc.nextInt();
}
int[] arr = new int[arr_size];
System.out.println(
"Enter the elements of the array: ");
for (int i = 0; i < arr_size; i++) {
if (sc.hasNextInt()) {
arr[i] = sc.nextInt();
}
}
    //int arr[] = { 6, 5, 12, 10, 9, 1 };

    MergeSort ob = new MergeSort();
    ob.mergeSort(arr, 0, arr_size - 1);
    System.out.println("Sorted array:");
    printArray(arr);
  }
}
```

OUTPUT :-
Enter the size of the array:
4
Enter the elements of the array:
10
9
8
7
Sorted array:
7 8 9 10

MINI PROJECT

Practical-13.2

Problem Statement:

Design a mini project to implement a Snake and Ladders Game using Python.

Code:

```
import time
import random
import sys

SLEEP_BETWEEN_ACTIONS = 1
MAX_VAL = 100
DICE_FACE = 6

# snake takes you down from 'key' to 'value'
snakes = {
    8: 4,
    18: 1,
    26: 10,
    39: 5,
    51: 6,
    54: 36,
    56: 1,
    60: 23,
    75: 28,
    83: 45,
    85: 59,
    90: 48,
    92: 25,
    97: 87,
    99: 63
}

# ladder takes you up from 'key' to 'value'
ladders = {
    3: 20,
    6: 14,
```

```
    11: 28,
    15: 34,
    17: 74,
    22: 37,
    38: 59,
    49: 67,
    57: 76,
    61: 78,
    73: 86,
    81: 98,
    88: 91
}

player_turn_text = [
    "Your turn.",
    "Go.",
    "Please proceed.",
    "Lets win this.",
    "Are you ready?",
    "",
]

snake_bite = [
    "boohoo",
    "bummer",
    "snake bite",
    "oh no",
    "dang"
]

ladder_jump = [
    "woohoo",
    "woww",
    "nailed it",
    "oh my God...",
    "yaayyy"
```

```
]

def welcome_msg():
    msg = """
    Welcome to Snake and Ladder Game.
    Version: 1.0.0
    Developed by: https://www.pythoncircle.com

    Rules:
      1. Initally both the players are at starting position i.e. 0.
         Take it in turns to roll the dice.
         Move forward the number of spaces shown on the dice.
      2. If you lands at the bottom of a ladder, you can move up to the top of the ladder.
      3. If you lands on the head of a snake, you must slide down to the bottom of the snake.
      4. The first player to get to the FINAL position is the winner.
      5. Hit enter to roll the dice.

    """
    print(msg)


def get_player_names():
    player1_name = None
    while not player1_name:
        player1_name = input("Please enter a valid name for first player: ").strip()

    player2_name = None
    while not player2_name:
        player2_name = input("Please enter a valid name for second player: ").strip()

    print("\nMatch will be played between '" + player1_name + "' and '" + player2_name + "'\n")
    return player1_name, player2_name


def get_dice_value():
    time.sleep(SLEEP_BETWEEN_ACTIONS)
```

```
    dice_value = random.randint(1, DICE_FACE)
    print("Its a " + str(dice_value))
    return dice_value


def got_snake_bite(old_value, current_value, player_name):
    print("\n" + random.choice(snake_bite).upper() + " ~~~~~~~~>")
    print("\n" + player_name + " got a snake bite. Down from " + str(old_value) + " to " + str(current_value))


def got_ladder_jump(old_value, current_value, player_name):
    print("\n" + random.choice(ladder_jump).upper() + " ########")
    print("\n" + player_name + " climbed the ladder from " + str(old_value) + " to " + str(current_value))


def snake_ladder(player_name, current_value, dice_value):
    time.sleep(SLEEP_BETWEEN_ACTIONS)
    old_value = current_value
    current_value = current_value + dice_value

    if current_value > MAX_VAL:
        print("You need " + str(MAX_VAL - old_value) + " to win this game. Keep trying.")
        return old_value

    print("\n" + player_name + " moved from " + str(old_value) + " to " + str(current_value))
    if current_value in snakes:
        final_value = snakes.get(current_value)
        got_snake_bite(current_value, final_value, player_name)

    elif current_value in ladders:
        final_value = ladders.get(current_value)
        got_ladder_jump(current_value, final_value, player_name)

    else:
        final_value = current_value
```

```
    return final_value


def check_win(player_name, position):
    time.sleep(SLEEP_BETWEEN_ACTIONS)
    if MAX_VAL == position:
        print("\n\n\nThats it.\n\n" + player_name + " won the game.")
        print("Congratulations " + player_name)
        print("\nThank you for playing the game. Please visit https://www.pythoncircle.com\n\n")
        sys.exit(1)


def start():
    welcome_msg()
    time.sleep(SLEEP_BETWEEN_ACTIONS)
    player1_name, player2_name = get_player_names()
    time.sleep(SLEEP_BETWEEN_ACTIONS)

    player1_current_position = 0
    player2_current_position = 0

    while True:
        time.sleep(SLEEP_BETWEEN_ACTIONS)
        input_1 = input("\n" + player1_name + ": " + random.choice(player_turn_text) + " Hit the enter to roll dice:
")
        print("\nRolling dice...")
        dice_value = get_dice_value()
        time.sleep(SLEEP_BETWEEN_ACTIONS)
        print(player1_name + " moving....")
        player1_current_position = snake_ladder(player1_name, player1_current_position, dice_value)

        check_win(player1_name, player1_current_position)

        input_2 = input("\n" + player2_name + ": " + random.choice(player_turn_text) + " Hit the enter to roll dice:
")
        print("\nRolling dice...")
        dice_value = get_dice_value()
```

```
        time.sleep(SLEEP_BETWEEN_ACTIONS)
        print(player2_name + " moving....")
        player2_current_position = snake_ladder(player2_name, player2_current_position, dice_value)

        check_win(player2_name, player2_current_position)


if __name__ == "__main__":
    start()
```

OUTPUT:

```
    Welcome to Snake and Ladder Game.
    Version: 1.0.0
    Developed by: https://www.pythoncircle.com

    Rules:
      1. Initally both the players are at starting position i.e. 0.
         Take it in turns to roll the dice.
         Move forward the number of spaces shown on the dice.
      2. If you lands at the bottom of a ladder, you can move up to the top of the ladder.
      3. If you lands on the head of a snake, you must slide down to the bottom of the snake.
      4. The first player to get to the FINAL position is the winner.
      5. Hit enter to roll the dice.


Please enter a valid name for first player: Kasturi
Please enter a valid name for second player: XYZ

Match will be played between 'Kasturi' and 'XYZ'


Kasturi: Your turn. Hit the enter to roll dice:

Rolling dice... Its a 2
Kasturi moving....
```

Kasturi moved from 0 to 2

XYZ: Are you ready? Hit the enter to roll dice:

Rolling dice...

Its a 1

XYZ moving....

XYZ moved from 0 to 1

Kasturi: Your turn. Hit the enter to roll dice:

Rolling dice...

Its a 4

Kasturi moving....

Kasturi moved from 2 to 6

YAAYYY ########

Kasturi climbed the ladder from 6 to 14

XYZ: Your turn. Hit the enter to roll dice:

Rolling dice...

Its a 6

XYZ moving....

XYZ moved from 1 to 7

Kasturi: Hit the enter to roll dice: