GROUP A

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

```
#include<iostream>
#include<cstring>
#define max 10
using namespace std;
struct client
{
        long int iPhno;
        char name[20];


};
class hashtable
{
        client ht[max];

        public:
                hashtable()      //constructor
                {
                        for(int i=0;i<max;i++)
                        {
                                ht[i].iPhno=0;


                        }
                }
                void insert();
                void display();
                int search(int);
                int del(int);
                int hash(long int);
};
void hashtable::insert()
{
        client c;
        int iPos;
        char cAns;
        do
        {
                cout<<"\n Enter Phone Number:";
                cin>>c.iPhno;
                iPos=hash(c.iPhno);

                if(ht[iPos].iPhno==0)
                {
                        ht[iPos]=c;
                }
                else
                {
                        for(int i=iPos+1;i%max!=iPos;i++)
                        {
                                ht[i]=c;
                                break;
                        }
                }
                cout<<"\n Add More:";
                cin>>cAns;
        }while(cAns=='y' || cAns=='Y');
```

```
}
int hashtable::hash(long int key)
{
        return(key%max);
}
void hashtable::display()
{
        cout<<"\nSrno\tPhone number\n";
        for(int i=0;i<max;i++)
        {
                cout<<i<<"\t"<<ht[i].iPhno<<endl;
        }
}
int hashtable::search(int x)
{
        int iFlag=0;
        cout<<"Enter Phone number to be searched:";
        cin>>x;
        for(int i=0;i<max;i++)
        {
                if(ht[i].iPhno==x)
                {
                        cout<<"\n Phone Number Found at position "<<i;
                        iFlag=1;
                }
        }
        if(iFlag==0)
        cout<<"\n Phone Number Not Found";
}//end of search
```

```
int hashtable::del(int s)
{
        int iF=0;
        cout<<"\n Enter phone number to be deleted:";
        cin>>s;
        for(int i=0;i<max;i++)
        {
                if(ht[i].iPhno==s)
                {

                        ht[i].iPhno=0;
                        cout<<"\n Phone number found and deleted";
                        iF=1;
                }
        }
        if(iF==0)
        cout<<"\n Phone number not found";

}

int main()
{
        int y,s,iCh;
        hashtable h;
        do
        {
                cout<<"\n1.INSERT\n2.DISPLAY\n3.SEARCH\n4.DELETE\n5.EXIT\n\n";
                cout<<"Enter your choice:";
                cin>>iCh;
                cout<<"\n";
                switch(iCh)
                {
                        case 1:
                                h.insert();
                                cout<<"\n";
                                break;

                        case 2:
                                h.display();
                                cout<<"\n";
                                break;

                        case 3:
                                h.search(y);
                                cout<<"\n";
                                break;

                        case 4:
                                h.del(s);
                                cout<<"\n";
                                break;

                        case 5:
                                break;
                }
        }while(iCh!=5);
return 0;
}
```

OUTPUT

```
1.INSERT
2.DISPLAY
3.SEARCH
4.DELETE
5.EXIT
Enter your choice:1
Enter Phone Number:9989765433
Add More:9080808909
1.INSERT
2.DISPLAY
3.SEARCH
4.DELETE
5.EXIT
Enter your choice:
1.INSERT
2.DISPLAY
3.SEARCH
4.DELETE
5.EXIT
Enter your choice
```

GROUP A

Implement all the functions of a dictionary (ADT) using hashing and handle collisions using chaining with / without replacement. Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, Keys must be unique Standard Operations: Insert(key, value), Find(key), Delete(key)

```
#include<iostream>
#include<cstdlib>
#include<string>
#include<cstdio>
using namespace std;
const int TABLE_SIZE = 128;
class HashNode
{
 public:
 int key;
 int value;
 HashNode* next;
 HashNode(int key, int value)
 {
 this->key = key;
 this->value = value;
 this->next = NULL;
 }
};
class HashMap
{
 private:
 HashNode** htable;
 public:
 HashMap()
 {
```

```
htable = new HashNode*[TABLE_SIZE];
for (int i = 0; i < TABLE_SIZE; i++)
htable[i] = NULL;
}
~HashMap()
{
for (int i = 0; i < TABLE_SIZE; ++i)
{
HashNode* entry = htable[i];
while (entry != NULL)
{
HashNode* prev = entry;
entry = entry->next;
delete prev;
}
}
delete[] htable;
}
int HashFunc(int key)
{
return key % TABLE_SIZE;
}
void Insert(int key, int value)
{
int hash_val = HashFunc(key);
HashNode* prev = NULL;
HashNode* entry = htable[hash_val];
while (entry != NULL)
{
prev = entry;
entry = entry->next;
}
if (entry == NULL)
{
entry = new HashNode(key, value);
if (prev == NULL)
{
htable[hash_val] = entry;
}
else
{
prev->next = entry;
}
}
else
{
entry->value = value;
}
}
void Remove(int key)
{
int hash_val = HashFunc(key);
HashNode* entry = htable[hash_val];
HashNode* prev = NULL;
if (entry == NULL || entry->key != key)
{
cout<<"No Element found at key "<<key<<endl;
return;
}
while (entry->next != NULL)
```

```
 {
 prev = entry;
 entry = entry->next;
 }
 if (prev != NULL)
 {
 prev->next = entry->next;
 }
 delete entry;
 cout<<"Element Deleted"<<endl;
 }
 int Search(int key)
 {
 bool flag = false;
 int hash_val = HashFunc(key);
 HashNode* entry = htable[hash_val];
 while (entry != NULL)
 {
 if (entry->key == key)
 {
 cout<<entry->value<<" ";
 flag = true;
 }
 entry = entry->next;
 }
 if (!flag)
 return -1;
 }
};
int main()
```

```
{
HashMap hash;
int key, value;
int choice;
while (1)
{
cout<<"\n Operations on Hash Table"<<endl;
cout<<"1.Insert element into the table"<<endl;
cout<<"2.Search element from the key"<<endl;
cout<<"3.Delete element at a key"<<endl;
cout<<"4.Exit"<<endl;
cout<<"Enter your choice: ";
cin>>choice;
switch(choice)
{
case 1:
cout<<"Enter key at which element to be inserted: ";
cin>>key;
cout<<"Enter element to be inserted: ";
cin>>value;
hash.Insert(key, value);
break;
case 2:
cout<<"Enter key of the element to be searched: ";
cin>>key;
cout<<"Element at key "<<key<<" : ";
if (hash.Search(key) == -1)
{
cout<<"No element found at key "<<key<<endl;
continue;
}
break;
case 3:
cout<<"Enter key of the element to be deleted: ";
cin>>key;
hash.Remove(key);
break;
case 4:
exit(1);
default:
cout<<"\nEnter correct option\n";
}
}
return 0;
}
```

OUTPUT

```
Operations on Hash Table
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 1
Enter key at which element to be inserted: 53
Enter element to be inserted: 78
Operations on Hash Table
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice:
```

GROUP B

A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method.

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

struct node // Node Declaration
{
   string label;
   //char label[10];
   int ch_count;
   struct node *child[10];
} * root;

class GT // Class Declaration
{
public:
   void create_tree();
   void display(node *r1);

   GT()
   {
      root = NULL;
   }
};

void GT::create_tree()
{
   int tbooks, tchapters, i, j, k;
   root = new node;
   cout << "Enter name of book : ";
   cin.get();
   getline(cin, root->label);
   cout << "Enter number of chapters in book : ";
   cin >> tchapters;
   root->ch_count = tchapters;
   for (i = 0; i < tchapters; i++)
   {
      root->child[i] = new node;
      cout << "Enter the name of Chapter " << i + 1 << " : ";
      cin.get();
      getline(cin, root->child[i]->label);
      cout << "Enter number of sections in Chapter : " << root->child[i]->label << " : ";
      cin >> root->child[i]->ch_count;
      for (j = 0; j < root->child[i]->ch_count; j++)
      {
          root->child[i]->child[j] = new node;
          cout << "Enter Name of Section " << j + 1 << " : ";
          cin.get();
          getline(cin, root->child[i]->child[j]->label);
      }
   }
}

void GT::display(node *r1)
{
   int i, j, k, tchapters;
   if (r1 != NULL)
```

```
        {
            cout << "\n-----Book Hierarchy---";
            cout << "\n Book title : " << r1->label;
            tchapters = r1->ch_count;
            for (i = 0; i < tchapters; i++)
            {

                cout << "\nChapter " << i + 1;
                cout << " : " << r1->child[i]->label;
                cout << "\nSections : ";
                for (j = 0; j < r1->child[i]->ch_count; j++)
                {
                    cout << "\n"<< r1->child[i]->child[j]->label;
                }
            }
        }
        cout << endl;
    }

    int main()
    {
        int choice;
        GT gt;
        while (1)
        {
            cout << "Book Tree Creation" << endl;
            cout << "1.Create" << endl;
            cout << "2.Display" << endl;
            cout << "3.Quit" << endl;
            cout << "Enter your choice : ";
        cin >> choice;
        switch (choice)
        {
        case 1:
           gt.create_tree();
        case 2:
           gt.display(root);
           break;
        case 3:
           cout << "Thanks for using this program!!!";
           exit(1);
        default:
           cout << "Wrong choice!!!" << endl;
        }
    }
    return 0;
}
```

OUTPUT

```
Book Tree Creation
1.Create
2.Display
3.Quit
Enter your choice : 1
Enter name of book : data structure and algorithm
Enter number of chapters in book : 3
Enter the name of Chapter 1 : hash
Enter number of sections in Chapter : hash : 2
Enter Name of Section 1 : hashing mathods
Enter Name of Section 2 : collosion
Enter the name of Chapter 2 : tree
Enter number of sections in Chapter : ree : 2
Enter Name of Section 1 : types of trees
Enter Name of Section 2 : algorithms
Enter the name of Chapter 3 : graph
Enter number of sections in Chapter : raph : 2
Enter Name of Section 1 : types of graph
Enter Name of Section 2 : spanning tree
-----Book Hierarchy---
 Book title : data structure and algorithm
Chapter 1 : hash
Sections :
hashing mathods
ollosion
Chapter 2 : ree
Sections :
types of trees
lgorithms
Chapter 3 : raph
Sections :
types of graph
panning tree
-----------------
Book Tree Creation
-----------------
1.Create
2.Display
3.Quit
Enter your choice : 2
-----Book Hierarchy---
 Book title : data structure and algorithm
```

```
Chapter 1 : hash
Sections :
hashing mathods
ollosion
Chapter 2 : ree
Sections :
types of trees
lgorithms
Chapter 3 : raph
Sections :
types of graph
panning tree
-----------------
Book Tree Creation
-----------------
1.Create
2.Display
3.Quit
Enter your choice :
```

<u>**GROUP B**</u>

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

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

struct Bstnode
{
 int data;
 Bstnode *left = NULL;
 Bstnode *right = NULL;

};

class Btree
{

  int n;
  int x;
  int flag;

public:
  Bstnode * root;
 Btree()
 {
  root = NULL;
 }

 Bstnode *GetNewNode(int in_data)
 {
```

```
 Bstnode * ptr = new Bstnode();
 ptr->data = in_data;
 ptr->left = NULL;
 ptr->right = NULL;
 return ptr;
}

Bstnode *insert( Bstnode *temp , int in_data)
{
 if( temp == NULL )
 {
  temp = GetNewNode(in_data);
 }
 else if( temp->data > in_data)
 {
  temp->left = insert(temp->left , in_data);
 }
 else
 {
  temp->right = insert( temp->right , in_data);
 }
 return temp;
}

void input()
{
 cout<<"ENTER NUMBER OF ELEMENTS IN THE BST : ";
 cin>>n;
 for(int i = 0 ; i < n ; i++)
 {
  cout<<"NUMBER = ";
  cin>>x;
  root = insert(root , x);
 }
}

int search(Bstnode *temp ,int in_data)
{
 if( temp != NULL)
 {
  if(temp->data == in_data)
  {
   cout<<":-- RECORD FOUND --:"<<endl;
   return 1;
  }
  else if(in_data < temp->data)
  {
   this->search(temp->left, in_data);
  }
  else if(in_data > temp->data)
  {
   this->search(temp->left , in_data);
  }
 }
 else
 {
   return 0;
 }
}
 void minvalue(Bstnode *temp)
```

```
{
 while(temp->left != NULL)
 {
  temp = temp->left;
 }
 cout<<"MINIMUM VALUE = "<<temp->data<<endl;
}

void mirror(Bstnode *temp)
{
 if(temp == NULL)
 {
  return;
 }
 else
 {
  Bstnode *ptr;
  mirror(temp->left);
  mirror(temp->right);
  ptr = temp->left;
  temp->left = temp->right;
  temp->right = ptr;
 }
}

void display()
{
 cout<<endl<<"--- INORDER TRAVERSAL ---"<<endl;
 inorder(root);
 cout<<endl;
 cout<<endl<<"--- POSTORDER TRAVERSAL ---"<<endl;
 postorder(root);
 cout<<endl;
 cout<<endl<<"--- PREORDER TRAVERSAL ---"<<endl;
 preorder(root);
 cout<<endl;

}

void inorder(Bstnode *temp)
{
 if(temp != NULL)
 {
  inorder(temp->left);
  cout<<temp->data<<"  ";
  inorder(temp->right);
 }
}

void postorder(Bstnode *temp)
{
 if(temp != NULL)
 {
  postorder(temp->left);
  postorder(temp->right);
  cout<<temp->data<<" ";
 }
}

void preorder(Bstnode *temp)
```

```
 {
  if(temp != NULL)
  {
   cout<<temp->data<<" ";
   preorder(temp->left);
   preorder(temp->right);
  }
 }

 int depth(Bstnode *temp)
 {
  if(temp == NULL)
   return 0;
  return (max((depth(temp->left)),(depth(temp->right))) +1);
 }
};

int main()
{
 Btree obj;
 obj.input();
 obj.display();
 int a = 0;
 a = obj.search(obj.root,10);
 if( a == 0)
 {
  cout<<"ELEMENT NOT FOUND"<<endl;
 }
 else
  cout<<"ELEMENT FOUND"<<endl;
 cout<<endl<<a<<endl;
 obj.minvalue(obj.root);
 obj.mirror(obj.root);
 obj.inorder(obj.root);
 cout<<endl<<obj.depth(obj.root);
 return 0;
}
```

OUTPUT

```
ENTER NUMBER OF ELEMENTS IN THE BST : 5
NUMBER = 12
NUMBER = 45
NUMBER = 76
NUMBER = 98
NUMBER = 90
--- INORDER TRAVERSAL ---
12  45  76  90  98

--- POSTORDER TRAVERSAL ---
90 98 76 45 12

--- PREORDER TRAVERSAL ---
12 45 76 98 90
ELEMENT NOT FOUND


0
MINIMUM VALUE = 12
98  90  76  45  12
5
```

GROUP B

Construct an expression tree from the given prefix expression eg. +--a*bc/def and

traverse it using postordertraversal(non recursive) and then delete the entire tree.

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

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 << root->data;
        display(root->left);
        display(root->right);
    }
}
void tree::non_rec_postorder(node *top)
{
    stack1 s1, s2;
    node *T = top;
    cout << "\n";
    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 <<endl<<"Deleting node : " << node->data<<endl;
    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 : +ab
+ab
ab+
Deleting node : a
Deleting node : b
Deleting node : +
```

GROUP C

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.

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

int cost[10][10], i, j, k, n, qu[10], front, rear, v, visit[10], visited[10];
int stk[10], top, visit1[10], visited1[10];

int main()
{
   int m;
   cout << "Enter number of vertices : ";
   cin >> n;
   cout << "Enter number of edges : ";
   cin >> m;

   cout << "\nEDGES :\n";
   for (k = 1; k <= m; k++)
   {
      cin >> i >> j;
      cost[i][j] = 1;
      cost[j][i] = 1;
   }
   cout << "The adjacency matrix of the graph is : " << endl;
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         cout << " " << cost[i][j];
```

```
        }
        cout << endl;
    }

    cout << "Enter initial vertex : ";
    cin >> v;
    cout << "The BFS of the Graph is\n";
    cout << v<<endl;
    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[rear++] = j;
            }
        v = qu[front++];
        cout << v << " ";
        k++;
        visit[v] = 0;
        visited[v] = 1;
    }

    cout <<endl<<"Enter initial vertex : ";
    cin >> v;
    cout << "The DFS of the Graph is\n";
    cout << v<<endl;
    visited[v] = 1;
    k = 1;
    while (k < n)
    {
        for (j = n; j >= 1; j--)
            if (cost[v][j] != 0 && visited1[j] != 1 && visit1[j] != 1)
            {
                visit1[j] = 1;
                stk[top] = j;
                top++;
            }
        v = stk[--top];
        cout << v << " ";
        k++;
        visit1[v] = 0;
        visited1[v] = 1;
    }

    return 0;
}
```

OUTPUT

```
Enter number of vertices : 4
Enter number of edges : 4
EDGES :
ab
The adjacency matrix of the graph is :
 1 0 0 0
 0 0 0 0
 0 0 0 0
 0 0 0 0
Enter initial vertex : The BFS of the Graph is
```

```
0

0 0 0

Enter initial vertex : The DFS of the Graph is

0

0 0 0
```

GROUP C

There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight take to reach city B from A, or the amount of fuel used for the journey. Represent this as a

graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Check whether the graph is connected or not. Justify the storage representation used

```
#include <iostream>

#include <queue>

using namespace std;


int adj_mat[50][50] = {0, 0};

int visited[50] = {0};


void dfs(int s, int n, string arr[])

{

    visited[s] = 1;

    cout << arr[s] << " ";

    for (int i = 0; i < n; i++)

    {

        if (adj_mat[s][i] && !visited[i])

            dfs(i, n, arr);

    }

}


void bfs(int s, int n, string arr[])

{

    bool visited[n];

    for (int i = 0; i < n; i++)

        visited[i] = false;

    int v;

    queue<int> bfsq;

    if (!visited[s])

    {

        cout << arr[s] << " ";
        bfsq.push(s);
        visited[s] = true;
        while (!bfsq.empty())
        {
            v = bfsq.front();
            for (int i = 0; i < n; i++)
            {
                if (adj_mat[v][i] && !visited[i])
                {
                    cout << arr[i] << " ";
                    visited[i] = true;
                    bfsq.push(i);
                }
            }
            bfsq.pop();
        }
    }
}

int main()
{
    cout << "Enter no. of cities: ";
    int n, u;
    cin >> n;
    string cities[n];
    for (int i = 0; i < n; i++)
    {
        cout << "Enter city #" << i << " (Airport Code): ";
        cin >> cities[i];
    }
```

```
cout << "\nYour cities are: " << endl;
for (int i = 0; i < n; i++)
   cout << "city #" << i << ": " << cities[i] << endl;
for (int i = 0; i < n; i++)
{
   for (int j = i + 1; j < n; j++)
   {
      cout << "Enter distance between " << cities[i] << " and " << cities[j] << " : ";
      cin >> adj_mat[i][j];
      adj_mat[j][i] = adj_mat[i][j];
   }
}
cout << endl;
for (int i = 0; i < n; i++)
   cout << "\t" << cities[i] << "\t";
for (int i = 0; i < n; i++)
{
   cout << "\n"
        << cities[i];
   for (int j = 0; j < n; j++)
      cout << "\t" << adj_mat[i][j] << "\t";
   cout << endl;
}
cout << "Enter Starting Vertex: ";
cin >> u;
cout << "DFS: ";
dfs(u, n, cities);
cout << endl;
cout << "BFS: ";
```

```
    bfs(u, n, cities);
    return 0;
}
```

OUTPUT

```
Enter no. of cities: 3
Enter city #0 (Airport Code): 45
Enter city #1 (Airport Code): 46
Enter city #2 (Airport Code): 47
Your cities are:
city #0: 45
city #1: 46
city #2: 47
Enter distance between 45 and 46 : 76
Enter distance between 45 and 47 : 87
Enter distance between 46 and 47 : 12
45              46              47
45      0               76              87


46      76              0               12


47      87              12              0
Enter Starting Vertex: 45
DFS:
BFS:  47
dash: 2: 47: not found
```

GROUP D

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?

```
#include<iostream>
using namespace std;
void con_obst(void);
void print(int,int);
float a[20],b[20],wt[20][20],c[20][20];
int r[20][20],n;
int main()
  {
        int i;
        cout<<"\n****** PROGRAM FOR OBST ******\n";
        cout<<"\nEnter the no. of nodes : ";
        cin>>n;cout<<"\nEnter the probability for successful search :: ";
        for(i=1;i<=n;i++)
          {
                cout<<"p["<<i<<"]";
                cin>>a[i];
          }
        cout<<"\nEnter the probability for unsuccessful search :: ";
        for(i=0;i<=n;i++)
          {
                cout<<"q["<<i<<"]";
                cin>>b[i];
          }
        con_obst();
        print(0,n);
        cout<<endl;
}
void con_obst(void)
{
```

```
int i,j,k,l,min;
for(i=0;i<n;i++)
  { //Initialisation
        c[i][i]=0.0;
        r[i][i]=0;
        wt[i][i]=b[i];
        // for j-i=1 can be j=i+1
        wt[i][i+1]=b[i]+b[i+1]+a[i+1];
        c[i][i+1]=b[i]+b[i+1]+a[i+1];
        r[i][i+1]=i+1;
  }
c[n][n]=0.0;
r[n][n]=0;
wt[n][n]=b[n];
//for j-i=2,3,4....,n
for(i=2;i<=n;i++)
  {
        for(j=0;j<=n-i;j++)
          {
                wt[j][j+i]=b[j+i]+a[j+i]+wt[j][j+i-1];
                c[j][j+i]=9999;
                for(l=j+1;l<=j+i;l++)
                  {
                        if(c[j][j+i]>(c[j][l-1]+c[l][j+i]))
                          {
                                c[j][j+i]=c[j][l-1]+c[l][j+i];
                                r[j][j+i]=l;
                          }
                  }
                c[j][j+i]+=wt[j][j+i];
```

```
                        }
                    cout<<endl;
            }
        cout<<"\n\nOptimal BST is :: ";
        cout<<"\nw[0]["<<n<<"] :: "<<wt[0][n];
        cout<<"\nc[0]["<<n<<"] :: "<<c[0][n];
        cout<<"\nr[0]["<<n<<"] :: "<<r[0][n];
 }
void print(int l1,int r1)
 {
        if(l1>=r1)
                return;
        if(r[l1][r[l1][r1]-1]!=0)
                cout<<"\n Left child of "<<r[l1][r1]<<" :: "<<r[l1][r[l1][r1]-1];
        if(r[r[l1][r1]][r1]!=0)
                cout<<"\n Right child of "<<r[l1][r1]<<" :: "<<r[r[l1][r1]][r1];
        print(l1,r[l1][r1]-1);
        print(r[l1][r1],r1);
        return;
}
```

OUTPUT

```
Enter the no. of nodes : 6
Enter the probability for successful search ::
p[1]2
p[2]3
p[3]3
p[4]1
p[5]2
p[6]3
Enter the probability for unsuccessful search ::
```

```
q[0]2
q[1]1
q[2]2
q[3]2
q[4]5
q[5]6
q[6]7
Optimal BST is ::
w[0][6] :: 39
c[0][6] :: 99
r[0][6] :: 5
 Left child of 5 :: 3
 Right child of 5 :: 6
 Left child of 3 :: 2
 Right child of 3 :: 4
 Left child of 2 :: 1
```

GROUP D

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

```
#include<iostream>
#include<stdio.h>
#include<string.h>
using namespace std;
class Tree
  {
    typedef struct node
          {
                   char key[10];
                   char meaning[10];
                   struct node *left;
                   struct node * right;
          }btree;
   public:
     btree *New,*root;
     Tree();
     void create();
     void insert(btree *root,btree *New);
          void inorder();
          void inorder_rec(btree *root);
          void postorder();
          void postorder_rec(btree *root);
  };
Tree::Tree()
  {
   root=NULL;
  }
```

```
void Tree::inorder()
  {
          inorder_rec(root);
  }
void Tree::inorder_rec(btree *root)
  {
          if(root!=NULL)
          {
                  inorder_rec(root->left);
                  cout<<"\n\t"<<root->key<<"\t"<<root->meaning;
                  inorder_rec(root->right);
    }
  }
void Tree::postorder()
  {
          postorder_rec(root);
  }
void Tree::postorder_rec(btree *root)
  {
          if(root!=NULL)
          {


                  postorder_rec(root->right);
                  cout<<"\n\t"<<root->key<<"\t"<<root->meaning;
                  postorder_rec(root->left);
    }
  }
void Tree::create()
  {
    New=new btree;
    New->left=New->right=NULL;
    cout<<"\n\tEnter the Keyword: ";
    cin>>New->key;
    cout<<"\n\tEnter the Meaning of "<<New->key<<" : ";
    cin>>New->meaning;
    if(root==NULL)
          {
                  root=New;
          }
    else
       {
                  insert(root,New);
       }
  }
void Tree::insert(btree *root,btree *New)
  {
    if(strcmp(root->key,New->key)>0)
       {
            if(root->left==NULL)
                  root->left=New;
        else
         insert(root->left,New);
             } else
       {
            if(root->right==NULL)
                   root->right=New;
         else
          insert(root->right,New);
             }
  }
```

```
 main()
 {
   Tree tr;
   int ch;
         char ans;
   do
     {
           cout<<"\n\t***** BST Operations *****";
           cout<<"\n\t1. Create\n\t2. Display\n\t3. Exit";
            cout<<"\n\t.....Enter Your Choice: ";
            cin>>ch;
            switch(ch)
                  {
                        case 1:

                              do
                              {
                                          tr.create();
                                          cout<<"......Do You Want To Continue: ";
                                          cin>>ans;
                              }while(ans=='y'||ans=='Y');
                              break;
                        case 2: cout<<"\n\t\t1. Ascending\n\t\t2. Descending\n\t\t.....Enter Your
Choice: ";
                              cin>>ch;
                              cout<<"\n\tKeyword\tMeaning";

                              switch(ch)
                              {
                                    case 1:
                                          tr.inorder();
```

```
                                        break;
                                    case 2:
                                        tr.postorder();
                                        break;
                              }
                              break;
                    case 3:
                              break;
          }
     cout<<"\n\t\t..... Do You Want to Continue: ";
     cin>>ans;
     }while(ans=='y'||ans=='Y');
 }
```

OUTPUT

```
***** BST Operations *****
     1. Create
     2. Display
     3. Exit
     .....Enter Your Choice: 1
     Enter the Keyword: AB
     Enter the Meaning of AB : TREE
     ......Do You Want To Continue: y
     Enter the Keyword: inorder
     Enter the Meaning of inorder : 1
     ......Do You Want To Continue:
```

<u>GROUP E</u>

Consider a scenario for Hospital to cater services to different kinds of patients as Serious (top priority), b) non-serious (medium priority), c) General Checkup (Least priority).

Implement the priority queue to cater services to the patients.

```
#include <iostream>
#include <queue>
#include <string>
struct Patient {
   std::string name;
   int priority;
   bool operator<(const Patient& other) const {
      return priority > other.priority; // Higher priority patients are served first
   }
};
int main() {
   // Priority queue to store patients
   std::priority_queue<Patient> hospitalQueue;
   Patient patient1 = { "John", 2 };  // Non-serious
   Patient patient2 = { "Mary", 1 };  // Serious
   Patient patient3 = { "Alice", 3 }; // General Checkup
   hospitalQueue.push(patient1);
   hospitalQueue.push(patient2);
   hospitalQueue.push(patient3);
   while (!hospitalQueue.empty()) {
      Patient servedPatient = hospitalQueue.top();
      hospitalQueue.pop();
      std::cout << "Serving patient: " << servedPatient.name << std::endl;
   }
 return 0;
}
```

OUTPUT

```
Serving patient: Mary
Serving patient: John
Serving patient: Alice
```

<u>GROUP F</u>

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.

```
#include<iostream>
#include<fstream>
#include<cstring>
using namespace std;
class tel
 {
 public:
 int rollNo,roll1;
 char name[10];
 char div;
 char address[20];
 void accept()
 {
  cout<<"\n\tEnter Roll Number : ";
  cin>>rollNo;
  cout<<"\n\tEnter the Name : ";
  cin>>name;
  cout<<"\n\tEnter the Division:";
  cin>>div;
  cout<<"\n\tEnter the Address:";
  cin>>address;
 }
        void accept2()
        {
               cout<<"\n\tEnter the Roll No. to modify : ";
               cin>>rollNo;
        }
        void accept3()
        {
              cout<<"\n\tEnter the name to modify : ";
              cin>>name;
        }
        int getRollNo()
        {
         return rollNo;
        }
  void show()
  {
 cout<<"\n\t"<<rollNo<<"\t\t"<<name<<"\t\t"<<div<<"\t\t"<<address;
  }
};
int main()
{
 int
i,n,ch,ch1,rec,start,count,add,n1,add2,start2,n2,y,a,b,on,oname,add3,start3,n3,y1,add4,start4,n
4;
 char name[20],name2[20];
 tel t1;
 count=0;
 fstream g,f;
 do
 {
  cout<<"\n1.Insert and overwrite\n2.Show\n3.Search & Edit(number)\n4.Search &
Edit(name)\n5.Search & Edit(onlynumber)\n6.Search & edit(only name)\n 7.Delete a Student
Record\n 8.Exit\n\tEnter the Choice\t:";
  cin>>ch;
  switch(ch)
  {
  case 1:
```

```
 f.open("StuRecord.txt",ios::out);
 x:t1.accept();
 f.write((char*) &t1,(sizeof(t1)));
 cout<<"\nDo you want to enter more records?\n1.Yes\n2.No";
 cin>>ch1;
  if(ch1==1)
   goto x;
  else
  {
   f.close();
   break;
  }

case 2:
 f.open("StuRecord.txt",ios::in);
 f.read((char*) &t1,(sizeof(t1)));
 //cout<<"\n\tRoll No.\t\tName \t\t Division \t\t Address";
 while(f)
 {
  t1.show();
  f.read((char*) &t1,(sizeof(t1)));
 }
 f.close();
 break;
case 3:
 cout<<"\nEnter the roll number you want to find";
 cin>>rec;
 f.open("StuRecord.txt",ios::in|ios::out);
 f.read((char*)&t1,(sizeof(t1)));
 while(f)
```

```
{
 if(rec==t1.rollNo)
 {
  cout<<"\nRecord found";
  add=f.tellg();
  f.seekg(0,ios::beg);
        start=f.tellg();
  n1=(add-start)/(sizeof(t1));
  f.seekp((n1-1)*sizeof(t1),ios::beg);
  t1.accept();
  f.write((char*) &t1,(sizeof(t1)));
  f.close();
  count++;
  break;
 }
 f.read((char*)&t1,(sizeof(t1)));
    }
if(count==0)
       cout<<"\nRecord not found";
f.close();
break;

case 4:
 cout<<"\nEnter the name you want to find and edit";
 cin>>name;
f.open("StuRecord.txt",ios::in|ios::out);
f.read((char*)&t1,(sizeof(t1)));
while(f)
{
 y=(strcmp(name,t1.name));
 if(y==0)
 {
  cout<<"\nName found";
  add2=f.tellg();
  f.seekg(0,ios::beg);
  start2=f.tellg();
  n2=(add2-start2)/(sizeof(t1));
  f.seekp((n2-1)*sizeof(t1),ios::beg);
  t1.accept();
  f.write((char*) &t1,(sizeof(t1)));
  f.close();
  break;
 }
      f.read((char*)&t1,(sizeof(t1)));
}
break;
   case 5:
        cout<<"\n\tEnter the roll number you want to modify";
        cin>>on;
        f.open("StuRecord.txt",ios::in|ios::out);
        f.read((char*) &t1,(sizeof(t1)));
        while(f)
        {
          if(on==t1.rollNo)
          {
            cout<<"\n\tNumber found";
            add3=f.tellg();
            f.seekg(0,ios::beg);
            start3=f.tellg();
            n3=(add3-start3)/(sizeof(t1));
            f.seekp((n3-1)*(sizeof(t1)),ios::beg);
            t1.accept2();
            f.write((char*)&t1,(sizeof(t1)));
            f.close();
            break;
           }
           f.read((char*)&t1,(sizeof(t1)));
         }
         break;
    case 6:
         cout<<"\nEnter the name you want to find and edit";
  cin>>name2;
 f.open("StuRecord.txt",ios::in|ios::out);
 f.read((char*)&t1,(sizeof(t1)));
 while(f)
 {
  y1=(strcmp(name2,t1.name));
  if(y1==0)
  {
   cout<<"\nName found";
   add4=f.tellg();
   f.seekg(0,ios::beg);
   start4=f.tellg();
   n4=(add4-start4)/(sizeof(t1));
   f.seekp((n4-1)*sizeof(t1),ios::beg);
   t1.accept3();
   f.write((char*) &t1,(sizeof(t1)));
   f.close();
   break;
  }
```

```
         f.read((char*)&t1,(sizeof(t1)));
    }
    break;
     case 7:
       int roll;
       cout<<"Please Enter the Roll No. of Student Whose Info You Want to Delete: ";
      cin>>roll;
      f.open("StuRecord.txt",ios::in);
      g.open("temp.txt",ios::out);
      f.read((char *)&t1,sizeof(t1));
      while(!f.eof())
      {
         if (t1.getRollNo() != roll)
            g.write((char *)&t1,sizeof(t1));
          f.read((char *)&t1,sizeof(t1));
      }
     cout << "The record with the roll no. " << roll << " has been deleted " << endl;
      f.close();
      g.close();
      remove("StuRecord.txt");
      rename("temp.txt","StuRecord.txt");
       break;
     case 8:
        cout<<"\n\tThank you";
        break;



         }
   }while(ch!=8);
 }
```

OUTPUT

```
1.Insert and overwrite
2.Show
3.Search & Edit(number)
4.Search & Edit(name)
5.Search & Edit(onlynumber)
6.Search & edit(only name)
 7.Delete a Student Record
 8.Exit
        Enter the Choice      :1
        Enter Roll Number : 4
        Enter the Name : nikita
        Enter the Division:c
        Enter the Address:shirwal
        Do you want to enter more records?
1.Yes
2.No2
1.Insert and overwrite
2.Show
3.Search & Edit(number)
4.Search & Edit(name)
5.Search & Edit(onlynumber)
6.Search & edit(only name)
 7.Delete a Student Record
 8.Exit
        Enter the Choice      :
```

GROUP F

Company maintains employee information as employee ID, name, designation and salary. Allow user to add, delete information of employee. Display information of particular

employee. If employee does not exist an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data.

```
#include <iostream>
#include <fstream>
#include <sstream>

using namespace std;

const string DATA_FILE = "employee_data.txt";

struct Employee {
    int employee_id;
    string name;
    string designation;
    double salary;
};

void addEmployee() {
    ofstream file(DATA_FILE, ios::app);
    Employee employee;

    cout << "Enter Employee ID: ";
    cin >> employee.employee_id;
    cout << "Enter Name: ";
    cin.ignore();
    getline(cin, employee.name);
    cout << "Enter Designation: ";
    getline(cin, employee.designation);
    cout << "Enter Salary: ";
    cin >> employee.salary;
    file << employee.employee_id << ',' << employee.name << ',' << employee.designation <<
',' << employee.salary << '\n';

    file.close();
    cout << "Employee added successfully!\n";
}

void deleteEmployee() {
    int employee_id;
    cout << "Enter Employee ID to delete: ";
    cin >> employee_id;

    ifstream inputFile(DATA_FILE);
    ofstream tempFile("temp.txt");
    bool found = false;
    string line;

    while (getline(inputFile, line)) {
        stringstream ss(line);
        string field;
        getline(ss, field, ',');
        int id = stoi(field);

        if (id == employee_id) {
            found = true;
        } else {
            tempFile << line << '\n';
        }
    }
    inputFile.close();
    tempFile.close();

    if (found) {
        remove(DATA_FILE.c_str());
        rename("temp.txt", DATA_FILE.c_str());
        cout << "Employee deleted successfully!\n";
    } else {
        remove("temp.txt");
        cout << "Employee not found.\n";
    }
}

void displayEmployee() {
    int employee_id;
    cout << "Enter Employee ID: ";
    cin >> employee_id;

    ifstream file(DATA_FILE);
    string line;
    bool found = false;

    while (getline(file, line)) {
        stringstream ss(line);
        string field;
        getline(ss, field, ',');
        int id = stoi(field);

        if (id == employee_id) {
            found = true;
            cout << "Employee ID: " << field << '\n';
            getline(ss, field, ',');
            cout << "Name: " << field << '\n';
            getline(ss, field, ',');
            cout << "Designation: " << field << '\n';
            getline(ss, field, ',');
            cout << "Salary: " << field << '\n';

            break;
        }
    }

    file.close();

    if (!found) {
        cout << "Employee not found.\n";
    }
}

int main() {
    int choice;

    do {
        cout << "\nEmployee Management System\n";
        cout << "--------------------------\n";
        cout << "1. Add Employee\n";
        cout << "2. Delete Employee\n";
        cout << "3. Display Employee\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                addEmployee();
                break;
            case 2:
                deleteEmployee();
                break;
            case 3:
                displayEmployee();
                break;
            case 4:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice. Please try again.\n";
                break;
        }
    } while (choice != 4);

    return 0;
}
```

OUTPUT

```
Employee Management System
1. Add Employee
```

```
2. Delete Employee

3. Display Employee

4. Exit
Enter your choice: 1

Enter Employee ID: 1

Enter Name: Nikita Yadav

Enter Designation: HR

Enter Salary: 90000

Employee added successfully!


Employee Management System

1. Add Employee

2. Delete Employee

3. Display Employee

4. Exit

Enter your choice:
```