#include <iostream>

using namespace std;

#include <string.h>

int const size = 3;

struct student

{

    int rno;

    char name[20];

    float SGPA;

};

void accept(struct student list[size]);

void display(struct student list[size]);

void insertSort(struct student list[size]);

void binarysearch(struct student list[size]);

void accept(struct student list[size])

{

    int i;

    for (i = 0; i < size; i++)

    {

        cout << "Enter rollno,name & SGPA:\n";

        cin >> list[i].rno >> list[i].name >> list[i].SGPA;

    }

}

void display(struct student list[size])

{

    int i;

    cout << "\n Roll no \t Name \t SGPA \n";

    for (i = 0; i < size; i++)

    {

        cout << "\n"<< list[i].rno << "\t" << list[i].name << "\t" << list[i].SGPA;

    }

}

void insertSort(struct student list[size])

{

    int k, j;

    struct student temp;

    for (k = 1; k < size; k++)

    {

        temp = list[k];

        j = k - 1;

        while (strcmp(list[j].name, temp.name) > 0 && j >= 0)

        {

            list[j + 1] = list[j];

            --j;

        }

        list[j + 1] = temp;

    }

}

void binarysearch(struct student list[size])

{

    int k, lower, upper, mid;

    char search[80];

    cout << "\n Enter name of the students you want to search\n";

    cin >> search;

    lower = 0;

    upper = size - 1;

    mid = (lower + upper) / 2;

    while (lower <= upper)

    {

        if (strcmp(list[mid].name, search) < 0)

            lower = mid + 1;

        else if (strcmp(list[mid].name, search) == 0)

        {

            cout << "\n"<< list[mid].rno << "\t" << list[mid].name << "\t" << list[mid].SGPA;

            break;

        }

        else

            upper = mid - 1;

        mid = (lower + upper) / 2;

    }

    if (lower > upper)

        cout << search << "not found in the list";

}

main()

{

    int ch;

    struct student data[20];

    accept(data);

    do

    {

        cout << "\n 1:Insertion Sort";

        cout << "\n 2:Binary Search";

        cout << "\n 3:Exit";

        cout << "\n Select your choice:";

        cin >> ch;

        switch (ch)

        {

        case 1:

            insertSort(data);

            display(data);

            break;

        case 2:

            binarysearch(data);

            break;

        case 3:

            cout<<"Exited Successfully";

            break;

        default:

            cout << "Invalid choice....";

        }

    } while (ch!=3);

}

#2

#include <iostream>

using namespace std;

#include <string.h>

int const size = 3;

struct student

{

    int rno;

    char name[20];

    float SGPA;

};

void accept(struct student list[size]);

void display(struct student list[80]);

void bubbleSort(struct student list[size]);

void search(struct student list[size]);

void accept(struct student list[size])

{

    int i;

    for (i = 0; i < size; i++)

    {

        cout << "Enter rollno,name & SGPA:";

        cin >> list[i].rno >> list[i].name >> list[i].SGPA;

    }

}

void display(struct student list[80])

{

    int i;

    cout << "\n Roll no \t Name \t SGPA \n";

    for (i = 0; i < size; i++)

    {

        cout << "\n"<< list[i].rno << "\t" << list[i].name << "\t" << list[i].SGPA;

    }

}

void bubbleSort(struct student list[size])

{

    int i, j;

    struct student temp;

    for (i = 0; i < size - 1; i++)

    {

        for (j = 0; j < (size - 1 - i); j++)

        {

            if (list[j].rno > list[j + 1].rno)

            {

                temp = list[j];

                list[j] = list[j + 1];

                list[j + 1] = temp;

            }

        }

    }

}

void search(struct student list[size])

{

    float SGPA;

    int i;

    cout << "\n Enter SGPA";

    cin >> SGPA;

    cout << "\n Rollno \t Name \t SGPA \n";

    for (int i = 0; i < size; i++)

    {

        if (SGPA == list[i].SGPA)

        {

            cout << "\n"<< list[i].rno << "\t" << list[i].name << "\t" << list[i].SGPA;

        }

        else{

            cout<<"record not found";

            break;

        }

    }

}

main()

{

    int ch, i;

    struct student data[20];

    accept(data);

    do

    {

        cout << "\n 1:Bubble Sort";

        cout << "\n 2:Search";

        cout << "\n 3:Exit";

        cout << "\n Select your choice:";

        cin >> ch;

        switch (ch)

        {

        case 1:

            bubbleSort(data);

            display(data);

            break;

        case 2:

            search(data);

            break;

        case 3:

            cout<<"Exited Successfully";

            break;

        default:

            cout << "Invalid choice....";

        }

    } while (ch!=3);

}

**3. Stack**

**#include<iostream>**

**#include<ctype.h>**

**#include<string.h>**

**using namespace std;**

**class Stack**

**{**

**//Structure for Expression**

**struct Stk**

**{**

**float Operator;**

**Stk \*Next;**

**Stk(){ Next=NULL;}**

**};**

**Stk \*Top;**

**public:**

**Stack(){Top=NULL;}**

**int Empty();**

**void Push(float Opr);**

**float Pop();**

**};**

**int Stack::Empty()**

**{**

**if(Top==NULL)**

**return 1;**

**return 0;**

**}**

**void Stack::Push(float Opr)**

**{**

**Stk \*Node;**

**Node=new Stk;**

**Node->Operator=Opr;**

**Node->Next=Top;**

**Top=Node;**

**}**

**float Stack::Pop()**

**{**

**Stk \*Temp=Top;**

**float Opr;**

**Top=Top->Next;**

**Opr=Temp->Operator;**

**delete Temp;**

**return Opr;**

**}**

**//Stack class End**

**//Function return Operater Priority**

**int Priority(char Op)**

**{**

**if(Op=='^')**

**return 2;**

**if(Op=='+' || Op=='-')**

**return 0;**

**else return 1;**

**}**

**//Return the result of given operation**

**float Operation(char Op,float A,float B)**

**{**

**int I=0;**

**float P=1;**

**if(Op=='\*') P=A\*B;**

**else if(Op=='/') P=A/B;**

**else if(Op=='+') P=A+B;**

**else if(Op=='-') P=A-B;**

**else while(I++<B) P=P\*A;**

**return P;**

**}**

**void infixTOpostfix(char str[20])**

**{**

**char Opr,post[20];**

**int i,j=0;**

**Stack S;**

**for(i=0;str[i]!='\0';i++)**

**{**

**if(isalnum(str[i])) post[j++]=str[i];**

**else**

**{**

**if(str[i]== ')')**

**{**

**Opr=S.Pop();**

**while(Opr!='(')**

**{ post[j++]=Opr; Opr=S.Pop(); }//while**

**}**

**else { if(str[i]=='(');**

**else while(!S.Empty())**

**{**

**Opr=S.Pop();**

**if(Opr!='('&&Priority(Opr)>= Priority(str[i]))**

**post[j++]=Opr;**

**else**

**{S.Push(Opr);**

**break;}**

**}//while**

**S.Push(str[i]);**

**}**

**}**

**}//for**

**while(!S.Empty())**

**post[j++]=S.Pop();**

**post[j]='\0';**

**cout<<post;**

**}**

**void infixTOprefix(char str[20])**

**{**

**char Opr,pre[20];**

**int i,j=0;**

**Stack S;**

**for(i=strlen(str)-1;i>=0;i--)**

**{**

**if(isalnum(str[i])) pre[j++]=str[i];**

**else**

**{**

**if(str[i]== '(')**

**{**

**Opr=S.Pop();**

**while(Opr!=')')**

**{ pre[j++]=Opr; Opr=S.Pop(); }//while**

**}**

**else { if(str[i]==')');**

**else while(!S.Empty())**

**{**

**Opr=S.Pop();**

**if(Opr!=')'&&Priority(Opr)>Priority(str[i]))**

**pre[j++]=Opr;**

**else**

**{S.Push(Opr);**

**break;}**

**}//while**

**S.Push(str[i]);**

**}**

**}**

**}//for**

**while(!S.Empty())**

**pre[j++]=S.Pop();**

**pre[j]='\0';**

**for(j--;j>=0;j--)**

**cout<<pre[j];**

**}**

**float Postfix\_Evaluation(char String[20])**

**{**

**int I=0;**

**float Operand1,Operand2,Result;**

**Stack S;**

**while(String[I]!='\0')**

**{**

**if(String[I]>='0' &&String[I]<='9')**

**S.Push(String[I]-48);**

**else**

**{**

**Operand2=S.Pop();**

**Operand1=S.Pop();**

**Result=Operation(String[I],Operand1,Operand2);**

**S.Push(Result);**

**}**

**I++;**

**}**

**return S.Pop();**

**}**

**//PreFix Expression Evaluation**

**float Prefix\_Evaluation(char String[20])**

**{**

**int I=strlen(String)-1;**

**float Operand1,Operand2,Result;**

**Stack S;**

**while(I>=0)**

**{**

**if(String[I]>='0' &&String[I]<='9')**

**S.Push(String[I]-48);**

**else**

**{**

**Operand1=S.Pop();**

**Operand2=S.Pop();**

**Result=Operation(String[I],Operand1,Operand2);**

**S.Push(Result);**

**}**

**I--;**

**}**

**return S.Pop();**

**}**

**int main()**

**{**

**int Choice;**

**char Expression[25],Answer;**

**do**

**{**

**cout<<"\n1:Infix to Prefix\n2:Infix to Postfix\n3:PostfixEvaluation\n4:Prefix Evaluation";**

**cout<<"\nEnter your Choice: ";**

**cin>>Choice;**

**switch(Choice)**

**{**

**case 1:**

**cout<<"\nEnter infix Expression";**

**cin>>Expression;**

**infixTOprefix(Expression);**

**break;**

**case 2:**

**cout<<"\nEnter infix Expression";**

**cin>>Expression;**

**infixTOpostfix(Expression);**

**break;**

**case 3:**

**cout<<"\nEnter Postfix Expression";**

**cin>>Expression;**

**cout<<"\nEvaluated Result :"**

**<<Postfix\_Evaluation(Expression);**

**break;**

**case 4:**

**cout<<"\nEnter Prefix Expression";**

**cin>>Expression;**

**cout<<"\nEvaluated Result "**

**<<Prefix\_Evaluation(Expression);**

**break;**

**}**

**cout<<"\nContinue(y/n)...";**

**cin>>Answer;**

**}while(Answer=='y'||Answer=='Y');**

**return 0;**

**}**

**4. Circular Queue**

**#include <iostream>**

**#define SIZE 5 /\* Size of Circular Queue \*/**

**using namespace std;**

**class Queue {**

**private:**

**int items[SIZE], front, rear;**

**public:**

**Queue() {**

**front = -1;**

**rear = -1;**

**}**

**// Check if the queue is full**

**bool isFull() {**

**if (front == 0 && rear == SIZE - 1) {**

**return true;**

**}**

**if (front == rear + 1) {**

**return true;**

**}**

**return false;**

**}**

**// Check if the queue is empty**

**bool isEmpty() {**

**if (front == -1)**

**return true;**

**else**

**return false;**

**}**

**// Adding an element**

**void enQueue() {**

**int element;**

**if (isFull()) {**

**cout << "Queue is full";**

**} else {**

**if (front == -1) front = 0;**

**rear = (rear + 1) % SIZE;**

**cout<<"Enter the element to be inserted: ";**

**cin>>element;**

**items[rear] = element;**

**cout << endl**

**<< "Inserted " << element << endl;**

**}**

**}**

**// Removing an element**

**int deQueue() {**

**int element;**

**if (isEmpty()) {**

**cout << "Queue is empty" << endl;**

**return (-1);**

**} else {**

**element = items[front];**

**if (front == rear) {**

**front = -1;**

**rear = -1;**

**}**

**// Q has only one element,**

**// so we reset the queue after deleting it.**

**else {**

**front = (front + 1) % SIZE;**

**}**

**return (element);**

**}**

**}**

**void display() {**

**// Function to display status of Circular Queue**

**int i;**

**if (isEmpty()) {**

**cout << endl**

**<< "Empty Queue" << endl;**

**} else {**

**cout << "Front -> " << front;**

**cout << endl**

**<< "Items -> ";**

**for (i = front; i != rear; i = (i + 1) % SIZE)**

**cout << items[i];**

**cout << items[i];**

**cout << endl**

**<< "Rear -> " << rear;**

**}**

**}**

**};**

**int main() {**

**Queue q;**

**// Fails because front = -1**

**q.deQueue();**

**q.enQueue();**

**q.enQueue();**

**q.enQueue();**

**q.enQueue();**

**q.enQueue();**

**// Fails to enqueue because front == 0 && rear == SIZE - 1**

**q.enQueue();**

**q.display();**

**int elem = q.deQueue();**

**if (elem != -1)**

**cout << endl**

**<< "Deleted Element is " << elem;**

**q.display();**

**q.enQueue();**

**q.display();**

**// Fails to enqueue because front == rear + 1**

**q.enQueue();**

**return 0;**

**}**

**5. Expression Tree**

**#include <iostream>**

**using namespace std;**

**struct n {**

**char d;**

**n \*l;**

**n \*r;**

**};**

**char pf[50];**

**int top = -1;**

**n \*a[50];**

**int r(char inputch) {**

**if (inputch == '+' || inputch == '-' || inputch == '\*' || inputch== '/')**

**return (-1);**

**else if (inputch >= 'A' || inputch <= 'Z')**

**return (1);**

**else if (inputch >= 'a' || inputch <= 'z')**

**return (1);**

**else**

**return (-100);**

**}**

**void push(n \*tree) {**

**top++;**

**a[top] = tree;**

**}**

**n \*pop() {**

**top--;**

**return (a[top + 1]);**

**}**

**void construct\_expression\_tree(char \*suffix) {**

**char s;**

**n \*newl, \*p1, \*p2;**

**int flag;**

**s = suffix[0];**

**for (int i = 1; s != 0; i++) {**

**flag = r(s);**

**if (flag == 1) {**

**newl = new n;**

**newl->d = s;**

**newl->l = NULL;**

**newl->r = NULL;**

**push(newl);**

**} else {**

**p1 = pop();**

**p2 = pop();**

**newl = new n;**

**newl->d = s;**

**newl->l = p2;**

**newl->r = p1;**

**push(newl);**

**}**

**s = suffix[i];**

**}**

**}**

**void preOrder(n \*tree) {**

**if (tree != NULL) {**

**cout << tree->d;**

**preOrder(tree->l);**

**preOrder(tree->r);**

**}**

**}**

**void inOrder(n \*tree) {**

**if (tree != NULL) {**

**inOrder(tree->l);**

**cout << tree->d;**

**inOrder(tree->r);**

**}**

**}**

**void postOrder(n \*tree) {**

**if (tree != NULL) {**

**postOrder(tree->l);**

**postOrder(tree->r);**

**cout << tree->d;**

**}**

**}**

**int main(int argc, char \*\*argv) {**

**cout << "Enter Postfix Expression : ";**

**cin >> pf;**

**construct\_expression\_tree(pf);**

**cout << "In-Order Traversal : \n";**

**inOrder(a[0]);**

**cout << "\nPre-Order Traversal : \n";**

**preOrder(a[0]);**

**cout << "\nPost-Order Traversal : \n";**

**postOrder(a[0]);**

**return 0;**

**}**

**6. Binary Search Tree**

**# include <iostream> # include <cstdlib> using namespace std; /\* \* Node Declaration \*/ struct node { int info; struct node \*left; struct node \*right; }\*root; /\* \* Class Declaration \*/ class BST { public: void find(int, node \*\*, node \*\*); void insert(node \*, node \*); void del(int); void case\_a(node \*,node \*); void case\_b(node \*,node \*); void case\_c(node \*,node \*); void preorder(node \*); void inorder(node \*); void postorder(node \*); void display(node \*, int); BST() { root = NULL; } }; /\* \* Main Contains Menu \*/ int main() { int choice, num; BST bst; node \*temp; while (1) { cout<<"-----------------"<<endl; cout<<"Operations on BST"<<endl;**

**cout<<"-----------------"<<endl; cout<<"1.Insert Element "<<endl; cout<<"2.Delete Element "<<endl; cout<<"3.Inorder Traversal"<<endl; cout<<"4.Preorder Traversal"<<endl; cout<<"5.Postorder Traversal"<<endl; cout<<"6.Display"<<endl; cout<<"7.Quit"<<endl; cout<<"Enter your choice : "; cin>>choice; switch(choice) { case 1: temp = new node; cout<<"Enter the number to be inserted : "; cin>>temp->info; bst.insert(root, temp); break; case 2: if (root == NULL) { cout<<"Tree is empty, nothing to delete"<<endl; continue; } cout<<"Enter the number to be deleted : "; cin>>num; bst.del(num); break; case 3: cout<<"Inorder Traversal of BST:"<<endl; bst.inorder(root); cout<<endl; break; case 4: cout<<"Preorder Traversal of BST:"<<endl; bst.preorder(root); cout<<endl; break; case 5: cout<<"Postorder Traversal of BST:"<<endl; bst.postorder(root); cout<<endl; break; case 6: cout<<"Display BST:"<<endl; bst.display(root,1);**

**cout<<endl; break; case 7: exit(1); default: cout<<"Wrong choice"<<endl; } } } /\* \* Find Element in the Tree \*/ void BST::find(int item, node \*\*par, node \*\*loc) { node \*ptr, \*ptrsave; if (root == NULL) { \*loc = NULL; \*par = NULL; return; } if (item == root->info) { \*loc = root; \*par = NULL; return; } if (item < root->info) ptr = root->left; else ptr = root->right; ptrsave = root; while (ptr != NULL) { if (item == ptr->info) { \*loc = ptr; \*par = ptrsave; return; } ptrsave = ptr; if (item < ptr->info) ptr = ptr->left; else ptr = ptr->right;**

**} \*loc = NULL; \*par = ptrsave; } /\* \* Inserting Element into the Tree \*/ void BST::insert(node \*tree, node \*newnode) { if (root == NULL) { root = new node; root->info = newnode->info; root->left = NULL; root->right = NULL; cout<<"Root Node is Added"<<endl; return; } if (tree->info == newnode->info) { cout<<"Element already in the tree"<<endl; return; } if (tree->info > newnode->info) { if (tree->left != NULL) { insert(tree->left, newnode); } else { tree->left = newnode; (tree->left)->left = NULL; (tree->left)->right = NULL; cout<<"Node Added To Left"<<endl; return; } } else { if (tree->right != NULL) { insert(tree->right, newnode); } else**

**{ tree->right = newnode; (tree->right)->left = NULL; (tree->right)->right = NULL; cout<<"Node Added To Right"<<endl; return; } } } /\* \* Delete Element from the tree \*/ void BST::del(int item) { node \*parent, \*location; if (root == NULL) { cout<<"Tree empty"<<endl; return; } find(item, &parent, &location); if (location == NULL) { cout<<"Item not present in tree"<<endl; return; } if (location->left == NULL && location->right == NULL) case\_a(parent, location); if (location->left != NULL && location->right == NULL) case\_b(parent, location); if (location->left == NULL && location->right != NULL) case\_b(parent, location); if (location->left != NULL && location->right != NULL) case\_c(parent, location); free(location); } /\* \* Case A \*/ void BST::case\_a(node \*par, node \*loc ) { if (par == NULL) { root = NULL;**

**} else { if (loc == par->left) par->left = NULL; else par->right = NULL; } } /\* \* Case B \*/ void BST::case\_b(node \*par, node \*loc) { node \*child; if (loc->left != NULL) child = loc->left; else child = loc->right; if (par == NULL) { root = child; } else { if (loc == par->left) par->left = child; else par->right = child; } } /\* \* Case C \*/ void BST::case\_c(node \*par, node \*loc) { node \*ptr, \*ptrsave, \*suc, \*parsuc; ptrsave = loc; ptr = loc->right; while (ptr->left != NULL) { ptrsave = ptr; ptr = ptr->left; }**

**suc = ptr; parsuc = ptrsave; if (suc->left == NULL && suc->right == NULL) case\_a(parsuc, suc); else case\_b(parsuc, suc); if (par == NULL) { root = suc; } else { if (loc == par->left) par->left = suc; else par->right = suc; } suc->left = loc->left; suc->right = loc->right; } /\* \* Pre Order Traversal \*/ void BST::preorder(node \*ptr) { if (root == NULL) { cout<<"Tree is empty"<<endl; return; } if (ptr != NULL) { cout<<ptr->info<<" "; preorder(ptr->left); preorder(ptr->right); } } /\* \* In Order Traversal \*/ void BST::inorder(node \*ptr) { if(ptr!=NULL) { inorder(ptr->left);**

**cout<<"\t"<<ptr->info; inorder(ptr->right); } } /\* \* Postorder Traversal \*/ void BST::postorder(node \*ptr) { if (root == NULL) { cout<<"Tree is empty"<<endl; return; } if (ptr != NULL) { postorder(ptr->left); postorder(ptr->right); cout<<ptr->info<<" "; } } /\* \* Display Tree Structure \*/ void BST::display(node \*ptr, int level) { int i; if (ptr != NULL) { display(ptr->right, level+1); cout<<endl; if (ptr == root) cout<<"Root->: "; else { for (i = 0;i < level;i++) cout<<" "; } cout<<ptr->info; display(ptr->left, level+1); } }**

**7. Kruskal’s Algorithm**

**#include <iostream>**

**#include <vector>**

**#include <utility>**

**#include <algorithm>**

**using namespace std;**

**const int MAX = 1000;**

**int id[MAX], nodes, edges;**

**pair <long long, pair<int, int> > p[MAX];**

**void init()**

**{**

**for(int i = 0;i < MAX;++i)**

**id[i] = i;**

**}**

**int root(int x)**

**{**

**while(id[x] != x)**

**{**

**id[x] = id[id[x]];**

**x = id[x];**

**}**

**return x;**

**}**

**void union1(int x, int y)**

**{**

**int p = root(x);**

**int q = root(y);**

**id[p] = id[q];**

**}**

**long long kruskal(pair<long long, pair<int, int> > p[])**

**{**

**int x, y;**

**long long cost, minimumCost = 0;**

**for(int i = 0;i < edges;++i)**

**{**

**x = p[i].second.first;**

**y = p[i].second.second;**

**cost = p[i].first;**

**if(root(x) != root(y))**

**{**

**minimumCost += cost;**

**cout<<x<<" ----> "<<y<<" :"<<p[i].first<<endl;**

**union1(x, y);**

**}**

**}**

**return minimumCost;**

**}**

**int main()**

**{**

**int x, y;**

**long long weight, cost, minimumCost;**

**init();**

**cout <<"Enter Nodes and edges"<<endl;**

**cin >> nodes >> edges;**

**for(int i = 0;i < edges;++i)**

**{**

**cout<<"Enter the value of X, Y and edges"<<endl;**

**cin >> x >> y >> weight;**

**p[i] = make\_pair(weight, make\_pair(x, y));**

**}**

**sort(p, p + edges);**

**minimumCost = kruskal(p);**

**cout <<"Minimum cost is "<< minimumCost << endl;**

**return 0;**

**}**

**8. Prim’s Algorithm**

**#include<iostream>**

**using namespace std;**

**const int V=6;**

**int min\_Key(int key[], bool visited[])**

**{**

**int min = 999, min\_index;**

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

**if (visited[v] == false && key[v] < min)**

**{**

**min = key[v];**

**min\_index = v;**

**}**

**}**

**return min\_index;**

**}**

**int print\_MST(int parent[], int cost[V][V])**

**{**

**int minCost=0;**

**cout<<"Edge \tWeight\n";**

**for (int i = 1; i< V; i++) {**

**cout<<parent[i]<<" - "<<i<<" \t"<<cost[i][parent[i]]<<" \n";**

**minCost+=cost[i][parent[i]];**

**}**

**cout<<"Total cost is"<<minCost;**

**}**

**void find\_MST(int cost[V][V])**

**{**

**int parent[V], key[V];**

**bool visited[V];**

**for (int i = 0; i< V; i++) {**

**key[i] = 999;**

**visited[i] = false;**

**parent[i]=-1;**

**}**

**key[0] = 0;**

**parent[0] = -1;**

**for (int x = 0; x < V - 1; x++)**

**{**

**int u = min\_Key(key, visited);**

**visited[u] = true;**

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

**{**

**if (cost[u][v]!=0 && visited[v] == false && cost[u][v] < key[v])**

**{**

**parent[v] = u;**

**key[v] = cost[u][v];**

**}**

**}**

**}**

**print\_MST(parent, cost);**

**}**

**int main()**

**{**

**int cost[V][V];**

**cout<<"Enter the vertices for a graph with 6 vetices";**

**for (int i=0;i<V;i++)**

**{**

**for(int j=0;j<V;j++)**

**{**

**cin>>cost[i][j];**

**}**

**}**

**find\_MST(cost);**

**return 0;**

**}**

**9. Shortest Path : Dijkstra’s Algorithm**

**#include<iostream>**

**#include<climits>**

**using namespace std;**

**int minimumDist(int dist[], bool Tset[])**

**{**

**int min=INT\_MAX,index;**

**for(int i=0;i<6;i++)**

**{**

**if(Tset[i]==false && dist[i]<=min)**

**{**

**min=dist[i];**

**index=i;**

**}**

**}**

**return index;**

**}**

**void Dijkstra(int graph[6][6],int src)**

**{**

**int dist[6];**

**bool Tset[6];**

**for(int i = 0; i<6; i++)**

**{**

**dist[i] = INT\_MAX;**

**Tset[i] = false;**

**}**

**dist[src] = 0;**

**for(int i = 0; i<6; i++)**

**{**

**int m=minimumDist(dist,Tset);**

**Tset[m]=true;**

**for(int i = 0; i<6; i++)**

**{**

**if(!Tset[i] && graph[m][i] && dist[m]!=INT\_MAX && dist[m]+graph[m][i]<dist[i])**

**dist[i]=dist[m]+graph[m][i];**

**}**

**}**

**cout<<"Vertex\t\tDistance from source"<<endl;**

**for(int i = 0; i<6; i++)**

**{**

**char str=65+i;**

**cout<<str<<"\t\t\t"<<dist[i]<<endl;**

**}**

**}**

**int main()**

**{**

**int graph[6][6]={**

**{0, 10, 20, 0, 0, 0},**

**{10, 0, 0, 50, 10, 0},**

**{20, 0, 0, 20, 33, 0},**

**{0, 50, 20, 0, 20, 2},**

**{0, 10, 33, 20, 0, 1},**

**{0, 0, 0, 2, 1, 0}};**

**Dijkstra(graph,0);**

**return 0;**

**}**

**10. Heap Sort**

**#include <iostream>**

**using namespace std;**

**void heapify(int arr[], int n, int i) {**

**// Find largest among root, left child and right child**

**int largest = i;**

**int left = 2 \* i + 1;**

**int right = 2 \* i + 2;**

**if (left < n && arr[left] > arr[largest])**

**largest = left;**

**if (right < n && arr[right] > arr[largest])**

**largest = right;**

**// Swap and continue heapifying if root is not largest**

**if (largest != i) {**

**swap(arr[i], arr[largest]);**

**heapify(arr, n, largest);**

**}**

**}**

**// main function to do heap sort**

**void heapSort(int arr[], int n) {**

**// Build max heap**

**for (int i = n / 2 - 1; i >= 0; i--)**

**heapify(arr, n, i);**

**// Heap sort**

**for (int i = n - 1; i >= 0; i--) {**

**swap(arr[0], arr[i]);**

**// Heapify root element to get highest element at root again**

**heapify(arr, i, 0);**

**}**

**}**

**// Print an array**

**void printArray(int arr[], int n) {**

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

**cout << arr[i] << " ";**

**cout << "\n";**

**}**

**// Driver code**

**int main() {**

**int arr[] = {1, 12, 9, 5, 6, 10};**

**int n = sizeof(arr) / sizeof(arr[0]);**

**heapSort(arr, n);**

**cout << "Sorted array is \n";**

**printArray(arr, n);**

**}**