**Practical – 1**

**AIM: Implementation and Time analysis of bubble sort.**

**Solution:**

#include<iostream> using namespace std; int main()

{

int a[100],n,i,j,swap; cout<<"Enter the number of elements: ";

cin>>n;

cout<<"Enter the elements: "; for(i=0;i<n;i++)cin>>a[i]; for(i=0;i<n;i++){

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

if(a[j]>a[j+1]){

swap=a[j]; a[j]=a[j+1]; a[j+1]=swap;

}

}

}

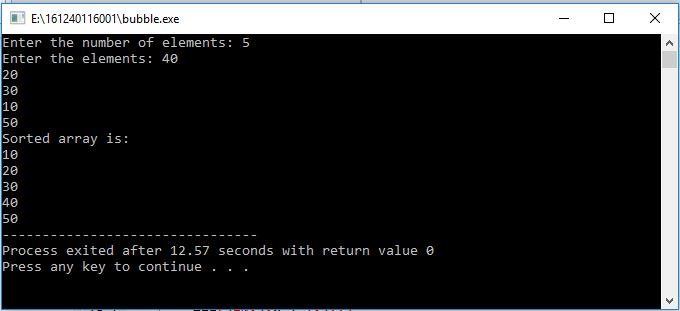
cout<<"Sorted array is: "; for(i=0;i<n;i++){ cout<<endl<<a[i];

}

return 0;

}

**Output:**



**Practical – 2**

**AIM: Implementation and Time analysis of selection sort.**

**Solution:**

#include<iostream> using namespace std; int main()

{

int a[100],i,j,swap,n,temp; cout<<"Enter the number of your element: ";

cin>>n;

cout<<"Insert the element :\n"; for(i=0;i<n;i++)

{

cin>>a[i];

}

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

{

swap=i;

for(j=i+1;j<n;j++)

{

if(a[swap]>a[j])

swap=j;

temp=a[i]; a[i]=a[swap]; a[swap]=temp;

}

}

cout<<endl<<"Sorted Array is:"; for(i=0;i<n;i++)

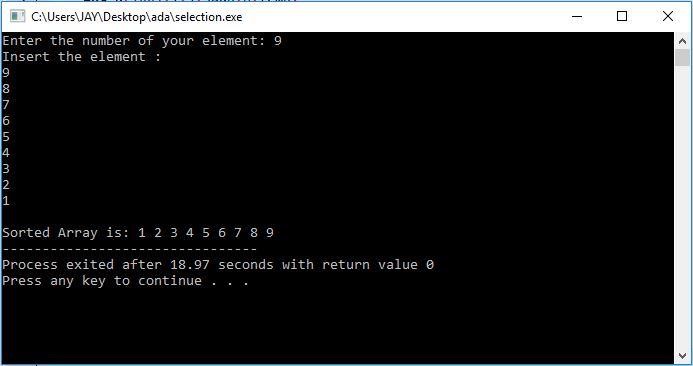
{ cout<<" "<<a[i];

}

return 0;

}

**Output:**



**Practical – 3**

**AIM: Implementation and Time analysis of insertion sort.**

**Solution:**

#include<iostream>

using namespace std;

int main()

{

int size, i, j, temp, a[100];

cout<<"Enter the size of the list: ";

cin>>size;

cout<<"Enter the elements of list : ";

for (i = 0; i< size; i++)

{cin>>a[i];}

for (i = 1; i< size; i++)

{

temp = a[i];

j = i - 1;

while ((temp < a[j]) && (j >= 0))

{

a[j + 1] = a[j];

j = j - 1;

}

a[j + 1] = temp;

for(j=0;j<size;j++)

{

cout<<" "<<a[j];

}cout<<endl;

}

cout<<"List after Sorting : ";

for (i = 0; i< size; i++)

{

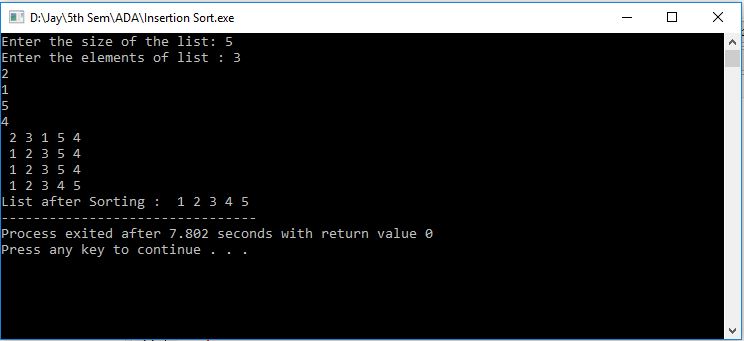
cout<<" "<<a[i];

}

return 0;

}

**Output:**



**Practical – 4**

**AIM: Implementation and Time analysis of merge sort.**

**Solution:**

#include <iostream>

using namespace std;

int Merge(int \*a, int low, int high, int mid){

int i, j, k, temp[high-low+1];

i = low;

k = 0;

j = mid + 1;

while (i<= mid && j <= high){

if (a[i] < a[j]){

temp[k] = a[i];

k++;

i++;

}

else{

temp[k] = a[j];

k++;

j++;

}

}

while (i<= mid){

temp[k] = a[i];

k++;

i++;

}

while (j <= high){

temp[k] = a[j];

k++;

j++;

}

for (i = low; i<= high; i++){

a[i] = temp[i-low];

}

}

int MergeSort(int \*a, int low, int high){

int mid;

if (low <high){

mid=(low+high)/2;

MergeSort(a, low, mid);

MergeSort(a, mid+1, high);

Merge(a, low, high, mid);

}

}

int main(){

int n, i;

cout<<"Enter the size of list: ";

cin>>n;

int arr[n];

cout<<"\nEnter element of list: "<<endl;

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

cin>>arr[i];

}

MergeSort(arr, 0, n-1);

cout<<"\nSorted Data ";

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

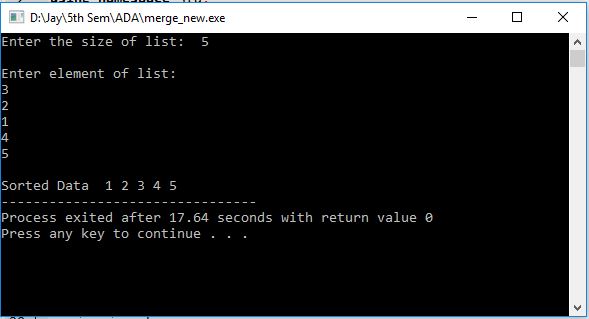
cout<<" "<<arr[i];

}

return 0;

}

**Output:**



**Practical – 5**

**AIM: Implementation and Time analysis of quick sort.**

**Solution:**

#include <iostream>

using namespace std;

void quick\_sort(int[],int,int);

int partition(int[],int,int);

int main(){

int a[50],n,i;

cout<<"Enter the size of list: ";

cin>>n;

cout<<"\nEnter the elements of list: ";

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

cin>>a[i];

}

quick\_sort(a,0,n-1);

cout<<"\nList after sorting: ";

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

cout<<a[i]<<" ";

}

return 0;

}

void quick\_sort(int a[],int l,int u){

int j;

if(l<u){

j=partition(a,l,u);

quick\_sort(a,l,j-1);

quick\_sort(a,j+1,u);

}

}

int partition(int a[],int l,int u){

int v,i,j,temp;

v=a[l];

i=l;

j=u+1;

do{

do{

i++;

}while(a[i]<v&&i<=u);

do{

j--;

}while(v<a[j]);

if(i<j){

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}while(i<j);

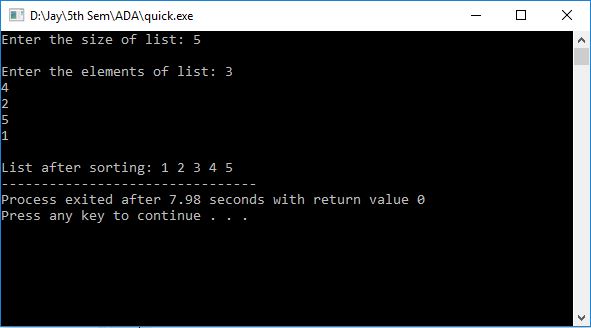
a[l]=a[j];

a[j]=v;

return(j);

}

**Output:**



**Practical – 6**

**AIM: Implementation of Binary Search**

**Solution:**

#include<iostream>

using namespace std;

int arr[50];

int search(int low,int high, int a){

int mid=(low+high)/2, count;

if(a == arr[mid]){

count=0;

cout<<"Given element "<<a<<" is identified at position : "<<mid+1;

return 0;

}

else if(a <arr[mid]){

int high1 = mid;

search(low, high1, a);

}

else if(a >arr[mid]){

int low1 = mid+1;

search(low1,high,a);

}

return 1;

}

int main(){

int n,zero=0,count=0;

cout<<"Enter the size of an array : ";

cin>>n;

int val;

cout<<"Enter all "<<n<<" elements to an array in sorted form \n";

for(int k=0; k<n; k++)cin>>arr[k];

cout<<"Enter the number to search with binary search : ";

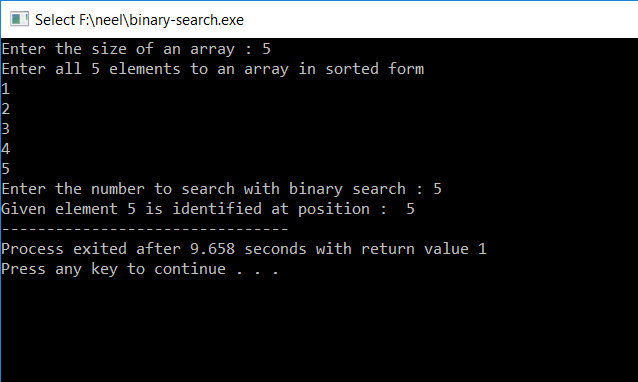
cin>>val;

search(zero,n,val);

return 1;

}

**Output:**



**Practical – 7**

**AIM: Implementation and Time analysis of heap sort.**

**Solution:**

#include <iostream>

using namespace std;

void buildHeap(int array[],int size,int i)

{

int max = i;

int left = 2\*i+1;

int right = 2\*i+2;

if(left<size && array[left]>array[max])

max = left;

if(right<size && array[right]>array[max])

max = right;

if(max!=i)

{

swap(array[i],array[max]);

buildHeap(array,size,max);

}

}

void heapSort(int array[],int size)

{

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

buildHeap(array,size,i);

for(int i=size-1;i>=0;i--)

{

swap(array[0],array[i]);

buildHeap(array,i,0);

}

}

int main()

{

int size;

cout<<"Enter the size of list : ";

cin>>size;

int array[size],n=0;

cout<<"Enter the elements to list \n";

while(n<size)cin>>array[n++];

heapSort(array,size);

cout<<"Sorted list: ";

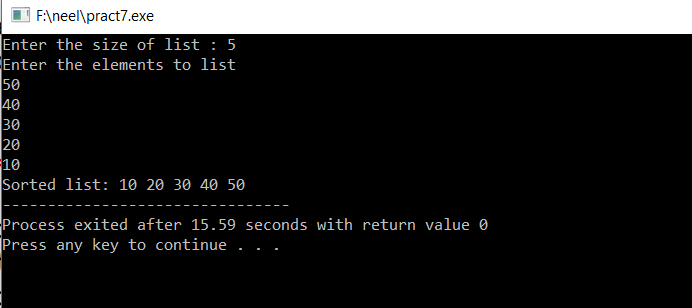
for(int i=0;i<size;i++)

cout<<array[i]<<" ";

return 0;

}

**Output:**

****

**Practical – 8.1**

**AIM: Find the factorial of the given number using recursive function.**

**Solution:**

#include<iostream>

using namespace std;

int fact(int n)

{

if(n!=0)

{

return n \* fact(n-1);

}

else

{

return 1;

}

}

int main()

{

int data;

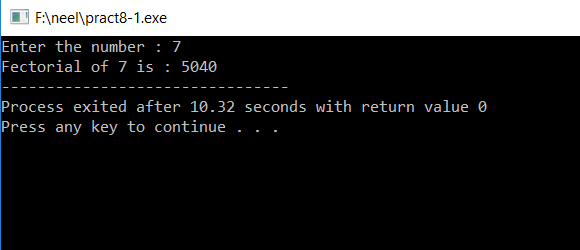
cout<<"Enter the number : ";

cin>>data;

cout<<"Fectorial of "<<data<<" is : "<<fact(data);

}

**Output:**



**Practical – 8.2**

**AIM: Find the Fibonacci series using recursive function.**

**Solution:**

#include<iostream>

using namespace std;

int feb(int n)

{

if((n==1)||(n==0))

{

return (n);

}

else

{

return (feb(n-1)+feb(n-2));

}

}

int main()

{

int number,i=0;

cout<<"Enter the number: ";

cin>>number;

while(i<number)

{

cout<<" "<<feb(i);

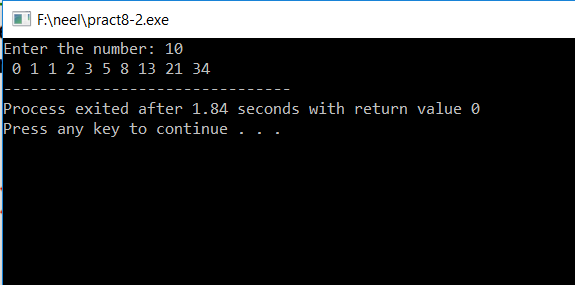
i++;

}

return 0;

}

**Output:**

****

**Practical – 9**

**AIM: Implementation of making change problem using dynamic programming.**

**Solution:**

#include<stdio.h>

#include<conio.h>

void main()

{

int d[100],mk[100][100],n,N,i=0,j=0,a,b;

clrscr();

printf("Enter number of coins you have: ");

scanf("%d",&n);

printf("Enter units: ");

scanf("%d",&N);

for(i=1;i<=n;i++)

{

printf("Enter d[%d] = ",i);

scanf("%d",&d[i]);

}

for(i=1;i<=n;i++)

{

for(j=0;j<=N;j++)

{

if(j==0)

mk[i][0]=0;

else if(i==1)

mk[1][j]=1+mk[1][j-d[i]];

else if(j<d[i])

mk[i][j]=mk[i-1][j];

else

{

a=mk[i-1][j];

b=1+mk[i][j-d[i]];

if(a<b)

{

mk[i][j]=a;

}

else

{

mk[i][j]=b;

}

}

}

}

printf("\nTable for making change:\n");

for(i=1;i<=n;i++){

for(j=0;j<=N;j++)

printf("%d ",mk[i][j]);

printf("\n");

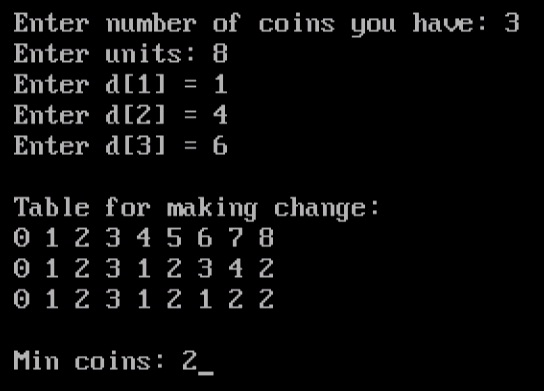
}

printf("\nMin coins: %d",mk[n][N]);

getch();

}

**Output:**

****

**Practical - 10**

**AIM: Implementation of a knapsack problem using dynamic programming.**

**Solution:-**

#include<stdio.h>

#include<conio.h>

void main()

{

intn,W,w[100],v[100],t[100][100],i=0,j=0,a,b;

clrscr();

printf("Enter total items n: ");

scanf("%d",&n);

printf("Enter capacity W: ");

scanf("%d",&W);

printf("Enter weights: \n");

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

{

printf("Enter w[%d]: ",i);

scanf("%d",&w[i]);

}

printf("Enter values: \n");

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

{

printf("Enter v[%d]: ",i);

scanf("%d",&v[i]);

}

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

{

for(j=0;j<=W;j++)

{

if(i==0 || j==0)

{

t[i][j]=0;

}

else if(j<w[i])

{

t[i][j]=t[i-1][j];

}

else

{

a=t[i-1][j];

b=v[i]+t[i-1][j-w[i]];

if(a>b)

{

t[i][j]=a;

}

else

{

t[i][j]=b;

}

}

}

}

printf("\nTable for Knapsack Problem:\n");

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

{

for(j=0;j<=W;j++)

{

printf("%d ",t[i][j]);

}

printf("\n");

}

j=W;

for(i=n;i>0;i--)

{

if(t[i][j]!=t[i-1][j])

{

printf("\nItem %d is selected.",i);

j=j-w[i];

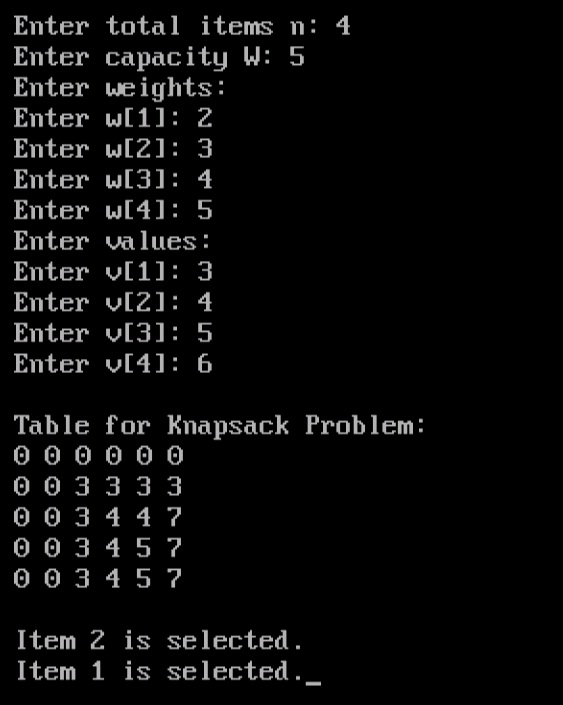
}

}

getch();

}

**Output:**

****

**Practical – 11**

**Aim: Implementation of chain matrix multiplication using dynamic programming.**

**Program Input:**

#include<stdio.h>

#include<conio.h>

void main(){

int d[100],m[100][100],n,i=0,j=0,k=0,s=0,t[10],l=0,temp;

clrscr();

printf("\nEnter the value of n: ");

scanf("%d",&n);

printf("\nEnter the value of d:");

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

printf("\nEnter d[%d]: ",i);

scanf("%d",&d[i]);

}

for(s=0;s<n;s++){

if(s==0){

for(i=1;i<=n;i++){

m[i][i]=0;

}

}

else if(s==1){

for(i=1;i<n;i++)

{

m[i][i+1]=(d[i-1]\*d[i]\*d[i+1]);

}

}

else

{

for(i=1;i<=(n-s);i++)

{

l=0;

for(k=i;k<(i+s);k++)

{

t[l++]=m[i][k]+m[k+1][i+s]+(d[i-1]\*d[k]\*d[i+s]);

}

for(k=1;k<l;k++){

temp=t[0];

if(t[k]<temp){

temp=t[k];

}

}

m[i][i+s]=temp;

}

}

}

printf("\nTable:\n");

for(i=1;i<=n;i++){

for(j=1;j<=n;j++)

printf("%d\t",m[i][j]);

printf("\n");

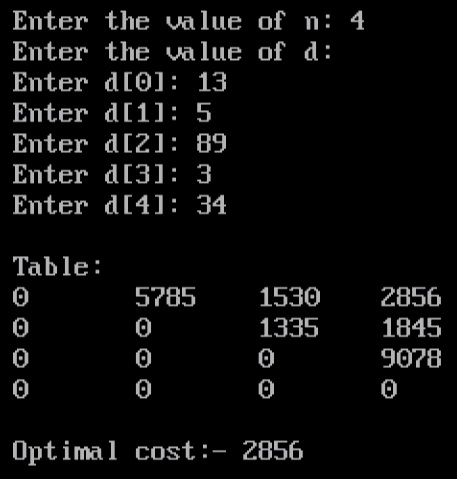
}

printf("\nOptimal cost:- %d",m[1][n]);

getch();

}

**Output:**

****

**Practical - 12**

**AIM: Implement prim’s algorithm.**

**Solution:-**

#include<stdio.h>

#include<conio.h>

int main()

{

int am[100][100],i=0,j=0,n,min=0,mc=0,a;

printf("\nEnter no. of nodes: ");

scanf("%d",&n);

printf("\nEnteradjancency matrix: \n");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

scanf("%d",&am[i][j]);

}

}

for(i=1;i<n;i++)

{

min=999;

for(j=1;j<=n;j++){

if(am[i][j]!=0 && am[i][j]<min){

min=am[i][j];

a=j;

}

}

mc=mc+min;

am[a][i]=0;

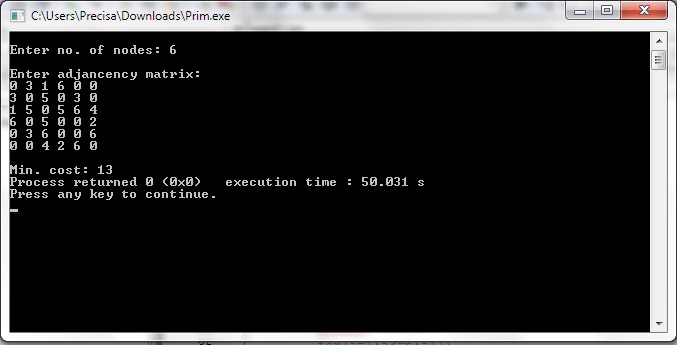
}

printf("\nMin. cost: %d",mc);

return 0;

}

**Output:**

****

**Practical -13**

**AIM: Implement Kruskal’s algorithm.**

**Solution:-**

#include<stdio.h>

#include<conio.h>

void main()

{

int a[100][100],i,j,s=1,n,min=99,cost=0,x,y;

printf("Enter number of node: ");

scanf("%d",&n);

printf("Enter the adjcent matrix: \n");

for(i=1;i<=n;i++){

for(j=1;j<=n;j++)

scanf("%d",&a[i][j]);

}

while(s<n){

for(i=1;i<=n;i++){

for(j=1;j<=n;j++) {

if(j>i && a[i][j]!=0){

if(a[i][j]<min){

min=a[i][j];

x=i;

y=j;

}

}

}

}

cost=cost+min;

s++;

min=99;

a[x][y]=0;

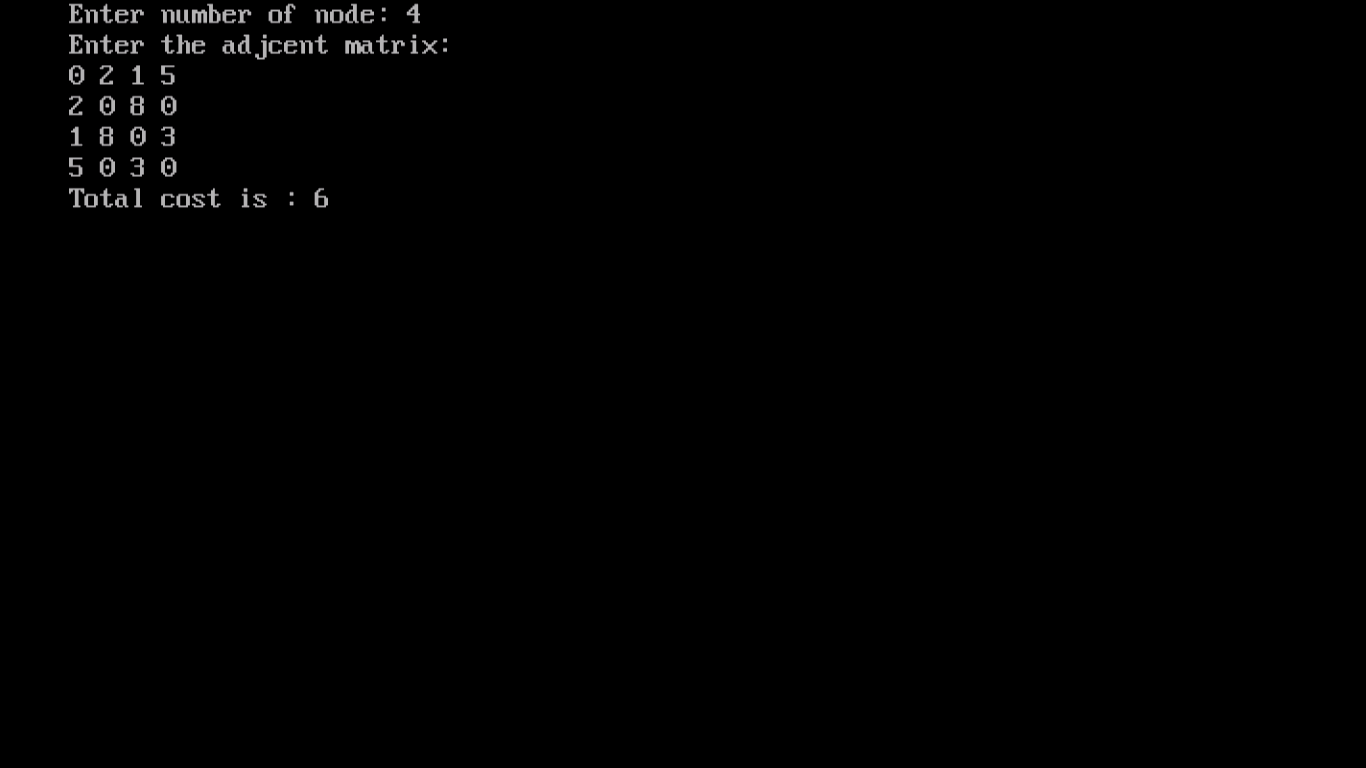
}

printf("Total cost is : %d",cost);

getch();

}

**Output:**

****

**Practical -14**

**AIM: Implementation of a knapsack problem using greedy algorithm.**

**Solution:-**

#include<stdio.h>

#include<conio.h>

void main()

{

intn,i=0,j=0,s;

float v[100],w[100],vw[100],t,f[50]={0},mw=0.0,mp=0.0;

printf("\nEnter number of items: ");

scanf("%d",&n);

printf("\nEnter knapsack size: ");

scanf("%d",&s);

printf("\nEnter weights: \n");

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

{

printf("Enter w[%d]= ",i);

scanf("%f",&w[i]);

}

printf("\nEnter profits: \n");

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

{

printf("Enter v[%d]= ",i);

scanf("%f",&v[i]);

}

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

{

vw[i]=(v[i]/w[i]);

}

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

{

for(j=0;j<=i;j++)

{

if(vw[i]>vw[j])

{

t=vw[i];

vw[i]=vw[j];

vw[j]=t;

t=v[i];

v[i]=v[j];

v[j]=t;

t=w[i];

w[i]=w[j];

w[j]=t;

}

}

}

printf("\nItem\tWeights\tProfits\tv/w");

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

{

printf("\n%d\t%.2f\t%.2f\t%.2f",i,w[i],v[i],vw[i]);

}

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

{

if(w[i]>s)

break;

else

{

f[i]=1.0;

s=s-w[i];

}

}

if(i<n)

f[i]=s/w[i];

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

w[i]=w[i]\*f[i];

v[i]=v[i]\*f[i];

}

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

mw=mw+w[i];

mp=mp+v[i];

}

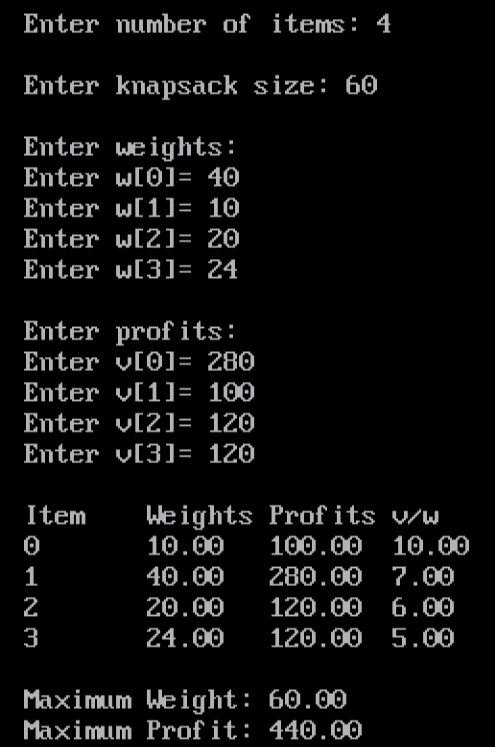
printf("\n\nMaximum Weight: %.2f",mw);

printf("\nMaximum Profit: %.2f",mp);

getch();

}

**Output:**



**Practical -15**

**AIM: Implementation of Graph and Searching (DFS).**

**Solution:-**

#include<stdio.h>

#include<conio.h>

int a[20][20],reach[20],n;

voiddfs(int v){

int i;

reach[v]=1;

for(i=1;i<=n;i++) {

if(a[v][i] && !reach[i]){

printf("\n%d->%d",v,i);

dfs(i);

}

}

}

void main(){

inti,j,count=0;

printf("\nEnter number of vertices: ");

scanf("%d",&n);

for(i=1;i<=n;i++){

reach[i]=0;

for(j=1;j<=n;j++)

{

a[i][j]=0;

}

}

printf("\nEnter the adjacency matrix:\n");

for(i=1;i<=n;i++){

for (j=1;j<=n;j++)

scanf("%d",&a[i][j]);

}

dfs(1);

printf("\n");

for(i=1;i<=n;i++){

if(reach[i]){

count++;

}

}

if(count==n)

{

printf("\nGraph is connected");

}

else

{

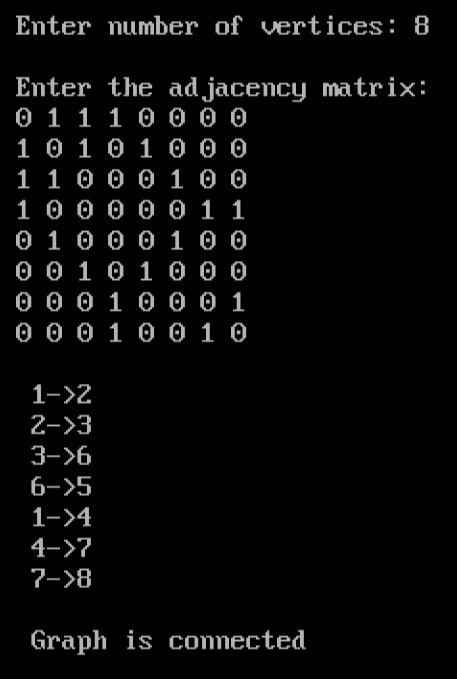
printf("\nGraph is not connected");

}

getch();

}

**Output:**



**Practical -16**

**AIM: Implementation of Graph and Searching (BFS).**

**Solution:-**

#include<stdio.h>

#include<conio.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;

voidbfs(int v)

{

for(i=1;i<=n;i++)

{

if(a[v][i] && !visited[i])

{

q[++r]=i;

}

}

if(f<=r)

{

visited[q[f]]=1;

bfs(q[f++]);

}

}

void main()

{

int v;

printf("\nEnter the number of vertices: ");

scanf("%d",&n);

for (i=1;i<=n;i++)

{

q[i]=0;

visited[i]=0;

}

printf("\nEnter graph data in matrix form:\n");

for (i=1;i<=n;i++)

{

for (j=1;j<=n;j++)

{

scanf("%d",&a[i][j]);

}

}

printf("\nEnter the starting vertex: ");

scanf("%d",&v);

bfs(v);

printf("\nThe node which are reachable are:\n");

for (i=1;i<=n;i++)

{

if(visited[i])

{

printf("%d\t",i);

}

else

{

printf("\n Bfs is not possible");

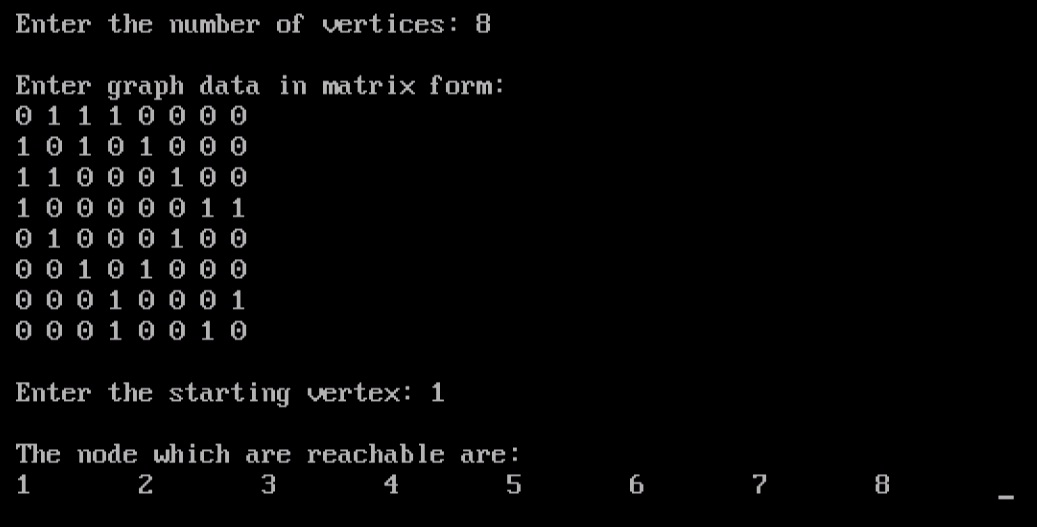
}

}

getch();

}

**Output:**



**Practical - 17**

**AIM: Implement LCS problem.**

**Solution:**

#include<bits/stdc++.h>

int max(int a, int b);

int lcs( char \*X, char \*Y, int m, int n ) {

int L[m+1][n+1];

int i, j;

for (i=0; i<=m; i++) {

for (j=0; j<=n; j++) {

if (i == 0 || j == 0)

L[i][j] = 0;

else if (X[i-1] == Y[j-1])

L[i][j] = L[i-1][j-1] + 1;

else

L[i][j] = max(L[i-1][j], L[i][j-1]);

}

}

return L[m][n];

}

int max(int a, int b) {

return (a > b)? a : b;

}

int main()

{

char X[] = "AGGTAB";

char Y[] = "GXTXAYB";

int m = strlen(X);

int n = strlen(Y);

printf("Length of LCS is %d", lcs( X, Y, m, n ) );

return 0;

}

**Output:**

