**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**Analysis and Design of Algorithms**

***Submitted by***

**ANUSHREE HARRISH (1BM20CS020)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**May-2022 to July-2022**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “**Analysis and Design of Algorithms**” carried out by **ANUSHREE HARRISH(1BM20CS020),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge:               **Dr. Jyothi S Nayak**

Designation Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

`

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Experiment Title** | **Page No.** |
| **1** | Write a recursive program to Solve  **a)** Towers-of-Hanoi problem **b)** To find GCD | **5** |
| **2** | Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N. | **7** |
| **3** | Sort a given set of N integer elements using Selection Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort. | **11** |
| **4** | Write program to do the following:  **a)** Print all the nodes reachable from a given starting node in a digraph using BFS method.  **b)** Check whether a given graph is connected or not using DFS method. | **15** |
| **5** | Sort a given set of N integer elements using Insertion Sort technique and compute its time taken. | **19** |
| **6** | Write program to obtain the Topological ordering of vertices in a given digraph. | **22** |
| **7** | Implement Johnson Trotter algorithm to generate permutations. | **24** |
| **8** | Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort. | **28** |
| **9** | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | **33** |
| **10** | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. | **37** |
| **11** | Implement Warshall’s algorithm using dynamic programming | **39** |
| **12** | Implement 0/1 Knapsack problem using dynamic programming. | **41** |
| **13** | Implement All Pair Shortest paths problem using Floyd’s algorithm. | **44** |
| **14** | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. | **46** |
| **15** | Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm. | **49** |
| **16** | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | **51** |
| **17** | 1. Implement “Sum of Subsets” using Backtracking. “Sum of Subsets” problem: Find a subset of a given set S = {s1,s2,……,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S = {1,2,5,6,8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn’t have a solution. | **53** |
| **18** | Implement “N-Queens Problem” using Backtracking. | **56** |

**Course Outcome**

|  |  |
| --- | --- |
| **CO1** | Ability to **analyze** time complexity of Recursive and Non-Recursive algorithms using asymptotic notations. |
| **CO2** | Ability to **design** efficient algorithms using various design techniques. |
| **CO3** | Ability to **apply** the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete |
| **CO4** | Ability to **conduct** practical experiments to solve problems using an appropriate designing method and find time efficiency. |

**PROGRAM 1**

Write a recursive program to Solve

**a)** Towers-of-Hanoi problem

#include <stdio.h>

void towers(int, char, char, char);

int main()

{

int num;

printf("Enter the number of disks : ");

scanf("%d", &num);

printf("The sequence of moves involved in the Tower of Hanoi are :\n");

towers(num, 'A', 'C', 'B');

return 0;

}

void towers(int num, char frompeg, char topeg, char auxpeg)

{

if (num == 1)

{

printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);

return;

}

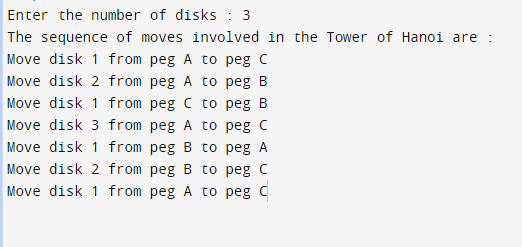
towers(num - 1, frompeg, auxpeg, topeg);

printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);

towers(num - 1, auxpeg, topeg, frompeg);

}

OUTPUT :



**b)** To find GCD

#include <stdio.h>

int hcf(int n1, int n2);

int main()

{

int n1, n2;

printf("Enter two positive integers: ");

scanf("%d %d", &n1, &n2);

printf("G.C.D of %d and %d is %d.", n1, n2, hcf(n1,n2));

return 0;

}

int hcf(int n1, int n2)

{

if (n2 != 0)

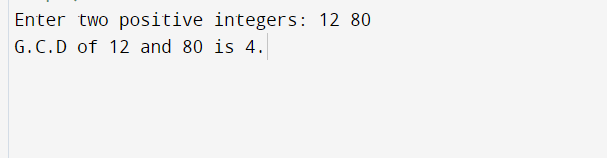
return hcf(n2, n1%n2);

else

return n1;

}

**Output :**

****

**PROGRAM 2**

Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N.

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

int bin\_srch(int [],int,int,int);

int lin\_srch(int [],int,int,int);

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

int n,a[10000];

int main()

{

int ch,key,search\_status,temp;

clock\_t end,start;

unsigned long int i, j;

while(1)

{

printf("\n1: Binary search\t 2: Linear search\t 3: Exit\n");

printf("\nEnter your choice:\t");

scanf("%d",&ch);

switch(ch)

{

case 1:

n=1000;

while(n<=5000) {

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

//a[i]=random(1000);

a[i]=i;

}

key=a[n-1];

start=clock();

search\_status=bin\_srch(a,0,n-1,key);

if(search\_status==-1)

printf("\nKey Not Found");

else

printf("\n Key found at position %d",search\_status);

for(j=0;j<500000;j++){ temp=38/600;}

end=clock();

printf("\nTime for n=%d is %f Secs",n,(((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

case 2:

n=1000;

while(n<=5000){

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

//a[i]=random(10000);

a[i]=i;

}

key=a[n-1];

start=clock();

search\_status=lin\_srch(a,0,n-1,key);

if(search\_status==-1)

printf("\nKey Not Found");

else

printf("\n Key found at position %d",search\_status);

for(j=0;j<500000;j++){ temp=38/600;}

end=clock();

printf("\nTime for n=%d is %f Secs",n,(((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

default:

exit(0);

}

getchar();

} }

int bin\_srch(int a[],int low,int high,int key)

{

int mid;

if(low>high){

return -1;

}

mid=(low+high)/2;

if(key==a[mid]){

return mid;

}

if(key<a[mid]){

return bin\_srch(a,low,mid-1,key);

}

Else {

return bin\_srch(a,mid+1,high,key);

} }

int lin\_srch(int a[],int i,int high,int key)

{

if(i>high){

return -1;

}

if(key==a[i]){

return i;

}

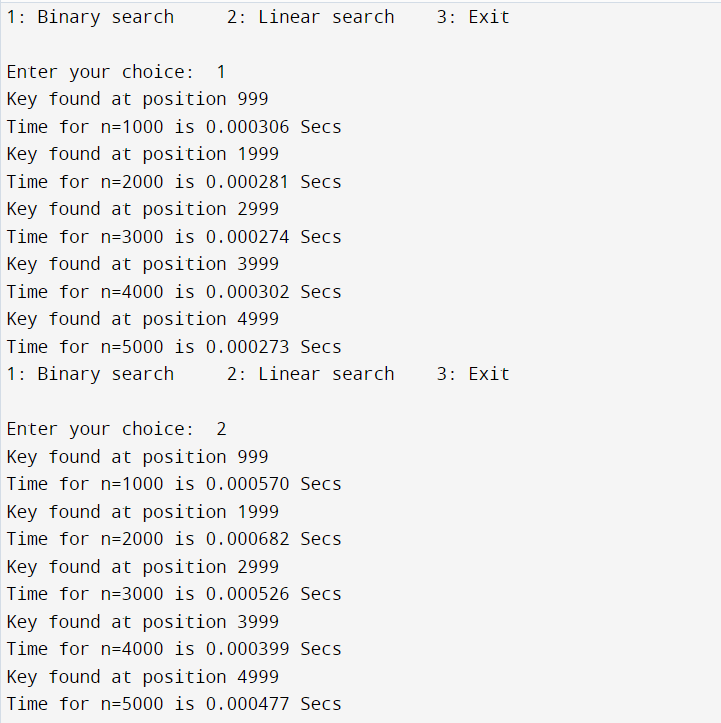
else{

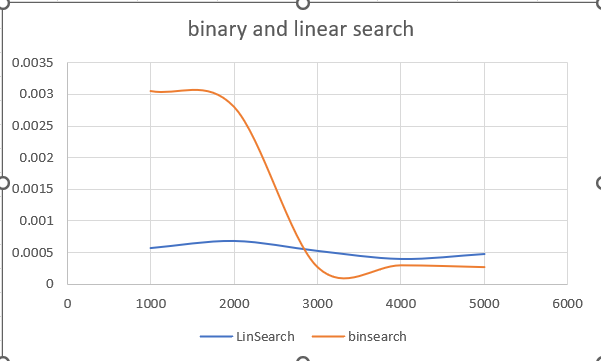
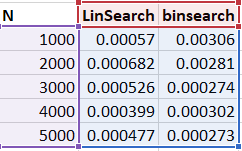
return lin\_srch(a,i+1,high,key);

}

}

**Output :**



**PROGRAM 3**

Sort a given set of N integer elements using Selection Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

void selsort(int n,int a[]);

void main()

{

int a[15000],n,i,j,ch,temp;

clock\_t start,end;

while(1){

printf("\n1:For manual entry of N value and array elements");

printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");

printf("\n3:To exit");

printf("\nEnter your choice:");

scanf("%d", &ch);

switch(ch) {

case 1: printf("\nEnter the number of elements: ");

scanf("%d",&n);

printf("\nEnter array elements: ");

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

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

}

start=clock();

selsort(n,a);

end=clock();

printf("\nSorted array is: ");

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

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

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

break;

case 2: n=10000;

while(n<=14500) {

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

//a[i]=random(1000);

a[i]=n-i;

}

start=clock();

selsort(n,a);

for(j=0;j<500000;j++){ temp=38/600;}

end=clock();

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

case 3: exit(0);

}

getchar();

}

}

void selsort(int n,int a[]){

int i,j,t,small,pos;

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

pos=i;

small=a[i];

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

if(a[j]<small) {

small=a[j];

pos=j;

} }

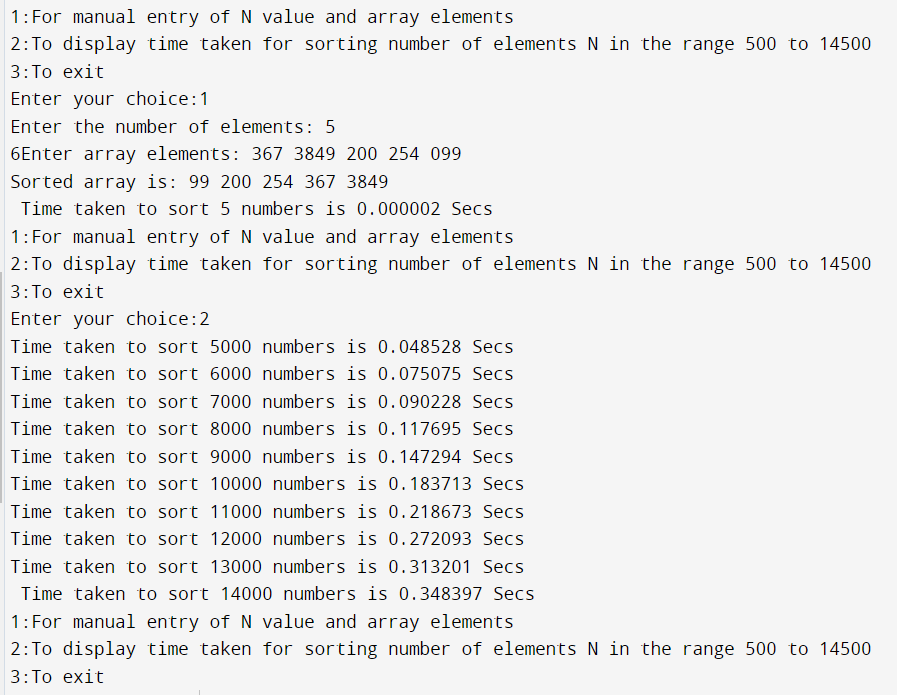
t=a[i];

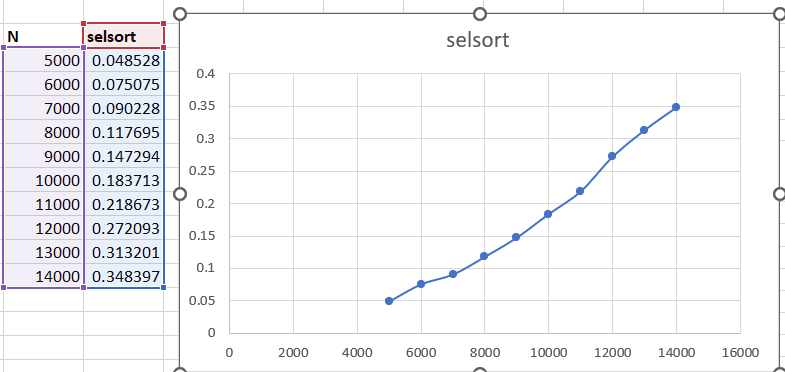
a[i]=a[pos];

a[pos]=t;

} }

**OUTPUT :**

****

****

**PROGRAM 4**

Write program to do the following:

**a)** Print all the nodes reachable from a given starting node in a digraph using BFS method.

#include<stdio.h>

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

void bfs(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("\n Enter the number of vertices:");

scanf("%d",&n);

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

q[i]=0;

visited[i]=0;

}

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

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

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

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

printf("\n Enter the starting vertex:");

scanf("%d",&v);

bfs(v);

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

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

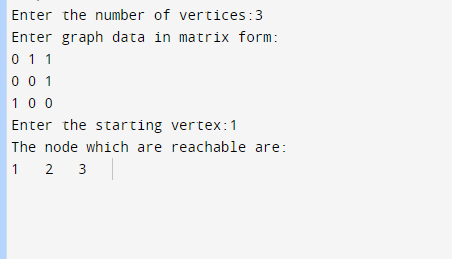
if(visited[i])

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

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

}

**OUTPUT :**

****

**b)** Check whether a given graph is connected or not using DFS method.

#include<stdio.h>

int a[10][10],n,vis[10];

int dfs(int);

void main()

{

int i,j,src,ans;

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

{

vis[j]=0;

}

printf("\nenter the no of nodes:\t");

scanf("%d",&n);

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

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

{

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

{

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

}

}

printf("\nenter the source node:\t");

scanf("%d",&src);

ans=dfs(src);

if(ans==1)

{

printf("\ngraph is connected\n");

}

else

{

printf("\ngragh is not connected\n");

}

}

int dfs(int src)

{

int j;

vis[src]=1;

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

{

if(a[src][j]==1&&vis[j]!=1)

{

dfs(j);

}

}

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

{

if(vis[j]!=1)

{

return 0;

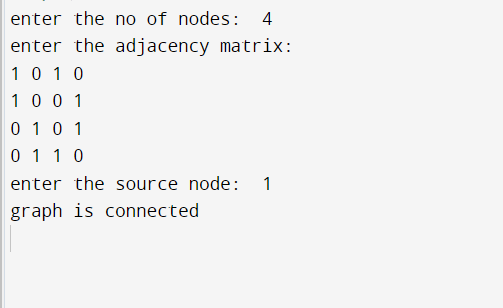
}

}

return 1;

}

**Output :**

****

**PROGRAM 5**

Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.

#include<stdio.h>

#include<time.h>

void insert\_sort(int n, int a[]){

int i, key, j;

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

key = a[i];

j = i - 1;

while (j >= 0 && a[j] > key) {

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

j = j - 1;

}

a[j + 1] = key;

}

}

void main(){

int a[15000],n,i,j,ch,temp;

clock\_t start,end;

while(1){

printf("\n1:For manual entry of N value and array elements");

printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");

printf("\n3:To exit");

printf("\nEnter your choice:");

scanf("%d", &ch);

switch(ch){

case 1: printf("\nEnter the number of elements: ");

scanf("%d",&n);

printf("\nEnter array elements: ");

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

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

}

start=clock();

insert\_sort(n,a);

end=clock();

printf("\nSorted array is: ");

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

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

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

break;

case 2: n=10000;

while(n<=14500) {

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

{

//a[i]=random(1000);

a[i]=n-i;

}

start=clock();

insert\_sort(n,a);

for(j=0;j<500000;j++){ temp=38/600;}

end=clock();

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

case 3: break;

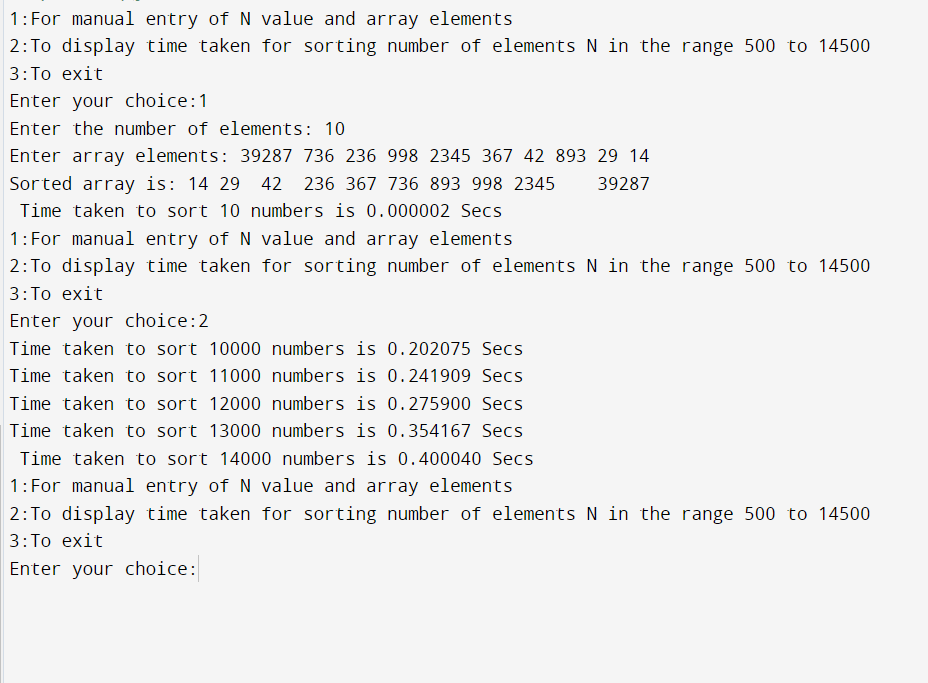
}

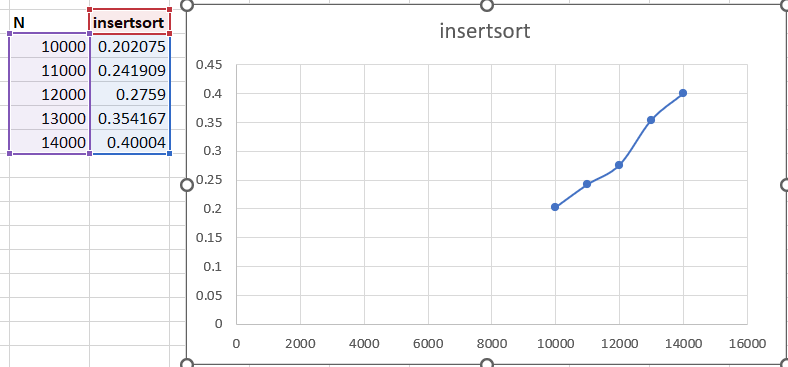
getchar();

}

}

**Output :**

****

****

**PROGRAM 6**

Write program to obtain the Topological ordering of vertices in a given digraph.

#include<stdio.h>

int main(){

int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;

printf("Enter the no of vertices:\n");

scanf("%d",&n);

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

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

printf("Enter row %d\n",i+1);

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

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

}

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

indeg[i]=0;

flag[i]=0; }

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

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

indeg[i]=indeg[i]+a[j][i];

printf("\nThe topological order is:");

while(count<n){

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

if((indeg[k]==0) && (flag[k]==0)){

printf("%d ",(k+1));

flag [k]=1;

}

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

if(a[i][k]==1)

indeg[k]--;

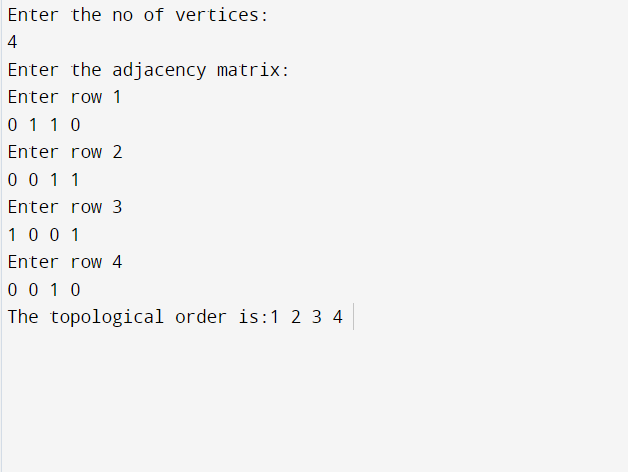
} }

count++;

} return 0;

}

**Output :**

****

**PROGRAM 7**

Implement Johnson Trotter algorithm to generate permutations.

#include<stdio.h>

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

int search(int a[],int n,int mobile)

{

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

{

if(a[i]==mobile)

return i+1;

}

return -1;

}

int findmobile(int a[], int dir[],int n)

{

int mobile=0;

int mobile\_prev=0;

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

{

if(dir[a[i]-1]==0 && i!=0)

{

if(a[i]>a[i-1] && a[i]>mobile\_prev)

{

mobile=a[i];

mobile\_prev=mobile;

}

}

if(dir[a[i]-1]==1 && i!=n-1)

{

if(a[i]>a[i+1] && a[i]>mobile\_prev)

{

mobile=a[i];

mobile\_prev=mobile;

}

}

}

if(mobile\_prev==0 && mobile==0)

return 0;

else

return mobile;

}

void printperm(int a[],int dir[], int n)

{

int mobile=findmobile(a,dir,n);

int pos=search(a,n,mobile);

if(dir[a[pos-1]-1]==0)

swap(&a[pos-1],&a[pos-2]);

else

swap(&a[pos-1],&a[pos]);

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

{

if(a[i]>mobile)

{

if(dir[a[i]-1]==0)

dir[a[i]-1]=1;

else

dir[a[i]-1]=0;

}

}

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

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

printf("\n");

}

int fact(int n)

{

int f=1;

for(int i=2;i<=n;i++)

f=f\*i;

return f;

}

int main()

{

int n;

printf("Enter n value\n");

scanf("%d",&n);

int dir[n],a[n];

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

{

dir[i]=0;

a[i]=i+1;

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

}

printf("\n");

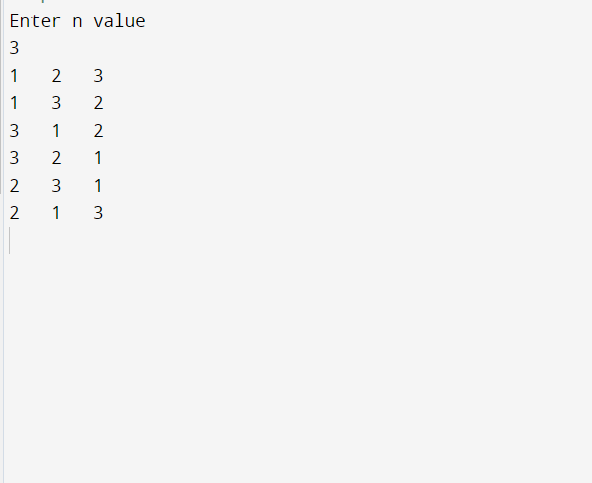
for(int i=1;i<fact(n);i++)

printperm(a,dir,n);

return 0;

}

**Output :**

****

**PROGRAM 8**

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

void split(int[],int,int);

void combine(int[],int,int,int);

void main()

{

int a[15000],n, i,j,ch, temp;

clock\_t start,end;

while(1)

{

printf("\n1:For manual entry of N value and array elements");

printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");

printf("\n3:To exit");

printf("\nEnter your choice:");

scanf("%d", &ch);

switch(ch)

{

case 1: printf("\nEnter the number of elements: ");

scanf("%d",&n);

printf("\nEnter array elements: ");

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

{

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

}

start=clock();

split(a,0,n-1);

end=clock();

printf("\nSorted array is: ");

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

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

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

break;

case 2:

n=500;

while(n<=14500) {

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

{

//a[i]=random(1000);

a[i]=n-i;

}

start=clock();

split(a,0,n-1);

for(j=0;j<500000;j++){ temp=38/600;}

end=clock();

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

case 3: exit(0);

}

getchar();

}

}

void split(int a[],int low,int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

split(a,low,mid);

split(a,mid+1,high);

combine(a,low,mid,high);

}

}

void combine(int a[],int low,int mid,int high)

{

int c[15000],i,j,k;

i=k=low;

j=mid+1;

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

{

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

{

c[k]=a[i];

++k;

++i;

}

else

{

c[k]=a[j];

++k;

++j;

}

}

if(i>mid){

while(j<=high){

c[k]=a[j];

++k;

++j;

}

}

if(j>high){

while(i<=mid){

c[k]=a[i];

++k;

++i;

}

}

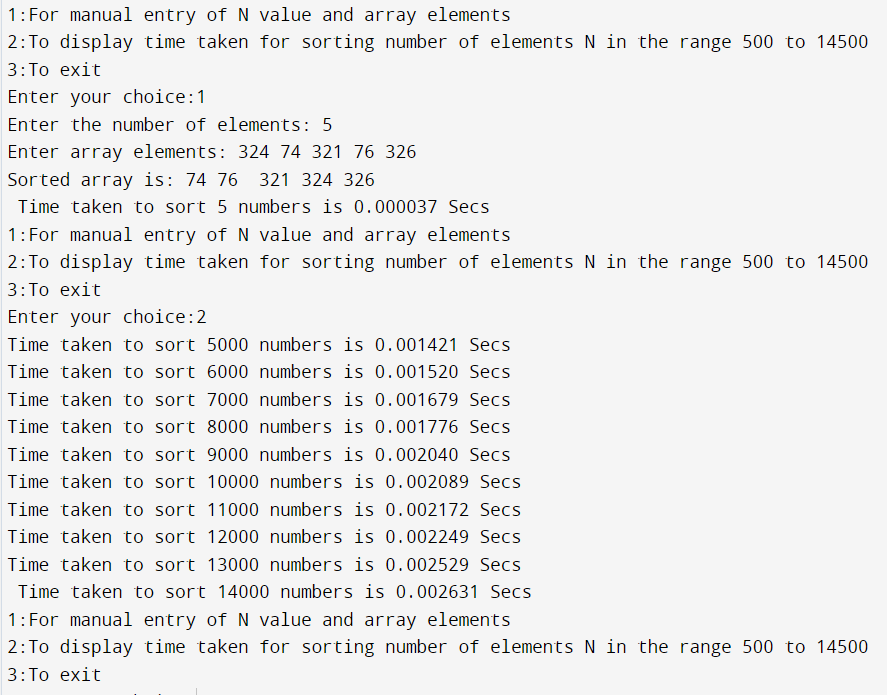
for(i=low;i<=high;i++) {

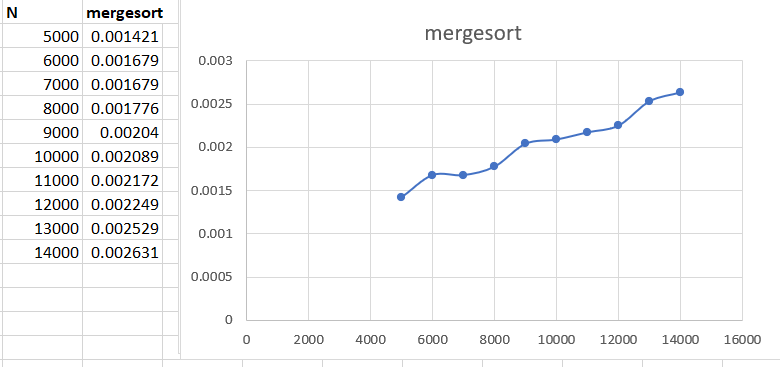
a[i]=c[i];

}

}

**OUTPUT :**

****

****

**PROGRAM 9**

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

void quicksort(int number[2000],int first,int last){

int i, j, pivot, temp;

if(first<last){

pivot=first;

i=first;

j=last;

while(i<j){

while(number[i]<=number[pivot]&&i<last)

i++;

while(number[j]>number[pivot])

j--;

if(i<j){

temp=number[i];

number[i]=number[j];

number[j]=temp;

}

}

temp=number[pivot];

number[pivot]=number[j];

number[j]=temp;

quicksort(number,first,j-1);

quicksort(number,j+1,last);

}

}

int main(){

int i, count, number[2000],op,j,temp,n;

clock\_t start ,end;

while(op!=3)

{

printf("1.for manual entry of N\n");

printf("2.to display the time taken for sorted of elements\n");

printf("3.quit\n");

printf("enter your choice\n");

scanf("%d",&op);

switch(op)

{

case 1 :

printf("enter how many elements:\n ");

scanf("%d",&count);

printf("Enter %d elements:\n ", count);

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

number[i]=rand();

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

}

start=clock();

quicksort(number,0,count-1);

end=clock();

printf("\nSorted elements: ");

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

printf(" %d",number[i]);

printf("\n Time taken to sort %d numbers is %f Secs\n",count, (((double)(end-start))/CLOCKS\_PER\_SEC));

break;

case 2 :

n=500;

while(n<=14500) {

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

{

//a[i]=random(1000);

number[i]=n-i;

}

start=clock();

quicksort(number,0,n-1);

for(j=0;j<500000;j++){ temp=38/600;}

end=clock();

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

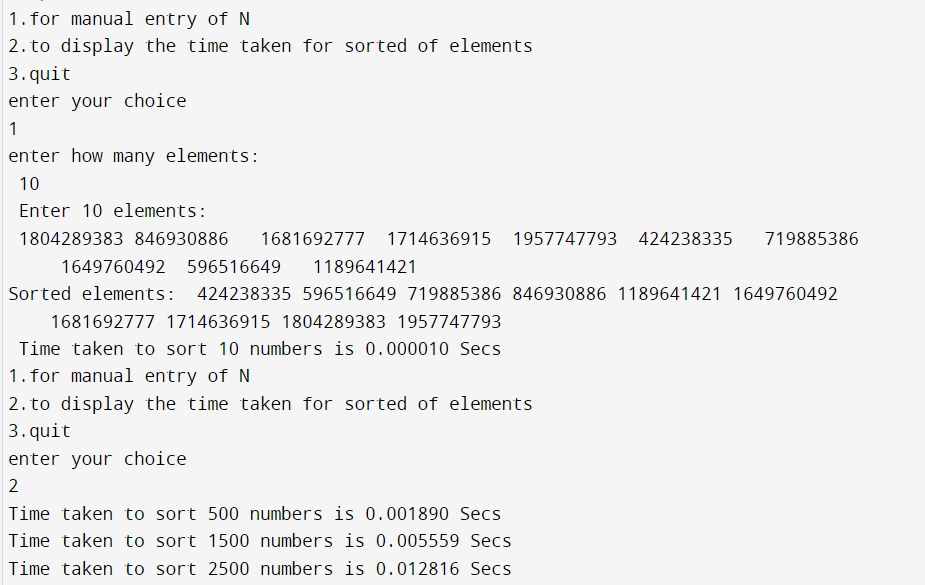
case 3: exit(0);

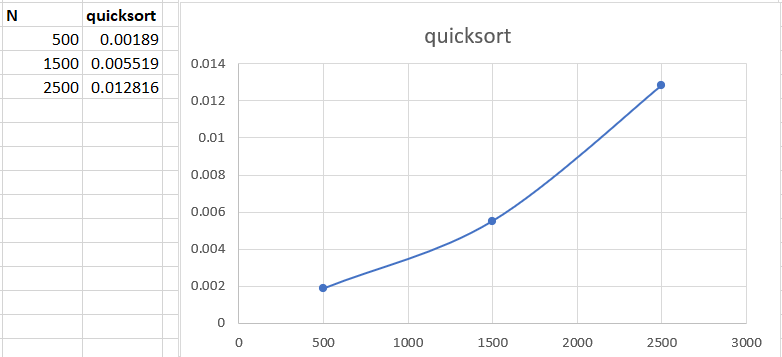
}

}return 0;

}

**Output :**

****

****

**PROGRAM 10**

Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

#include<stdio.h>

#include<time.h>

void swap(int \*a,int \*b){

int temp=\*a;

\*a=\*b;

\*b=temp;

}

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

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;

if(largest!=i){

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

heapify(arr,n,largest);

}

}

void heapsort(int arr[],int n){

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

heapify(arr,n,i);

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

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

heapify(arr,i,0);

}

}

void printarray(int arr[],int n){

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

printf("%d\n",arr[i]);

}

void main()

{

int n,i,j,temp,c;

printf("Enter the value of n:");

scanf("%d",&n);

int arr[20];

printf("\ngenerating random nos..\n");

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

arr[c]=rand()%100+1;

printf("%d\n",arr[c]);

}

clock\_t begin=clock();

heapsort(arr,n);

clock\_t end=clock();

printf("sorted array is\n");

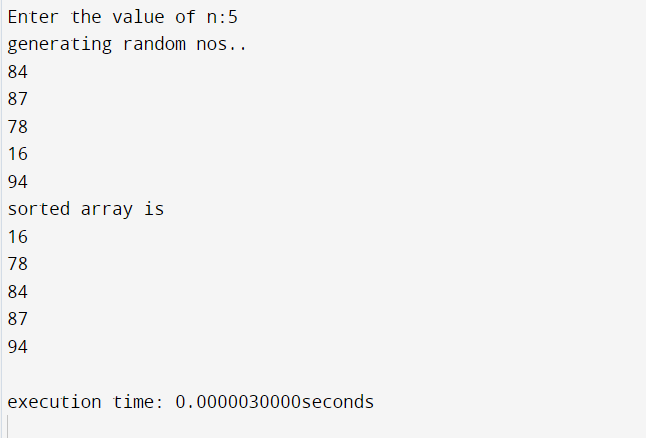
printarray(arr,n);

double time\_taken=(double)(end-begin)/CLOCKS\_PER\_SEC;

printf("\nexecution time: %.10fseconds\n",time\_taken );

}

**Output :**

****

**PROGRAM 11**

Implement Warshall’s algorithm using dynamic programming

#include<stdio.h>

#include<math.h>

int max(int,int);

void warshal(int p[10][10],int n) {

int i,j,k;

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

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

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

p[i][j]=max(p[i][j],p[i][k]&&p[k][j]);

}

int max(int a,int b) {

if(a>b)

return(a); else

return(b);

}

void main() {

int p[10][10]= {0},n,e,u,v,i,j;

printf("\n Enter the number of vertices:");

scanf("%d",&n);

printf("\n Enter the number of edges:");

scanf("%d",&e);

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

printf("\n Enter the end vertices of edge %d:",i);

scanf("%d%d",&u,&v);

p[u][v]=1;

}

printf("\n Matrix of input data: \n");

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

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

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

printf("\n");

}

warshal(p,n);

printf("\n Transitive closure: \n");

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

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

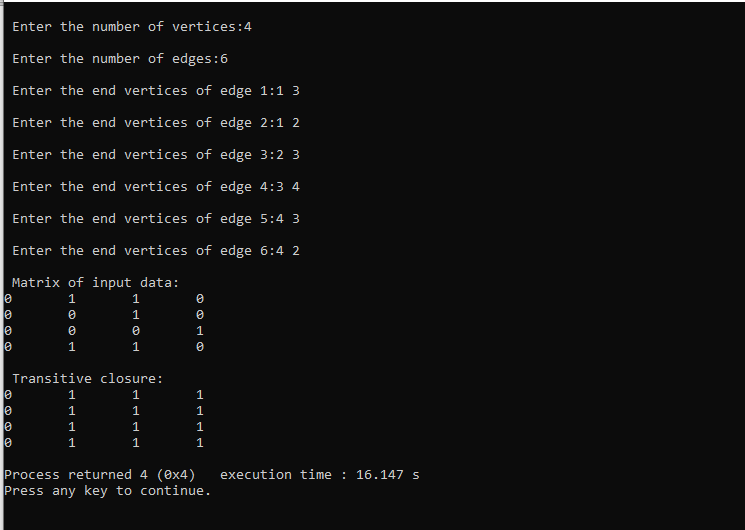
printf("%d\t",p[i][j]);

printf("\n");

}

}

**OUTPUT :**



**PROGRAM 12**

Implement 0/1 Knapsack problem using dynamic programming

#include<stdio.h>

void knapsack();

int max(int,int);

int i,j,n,m,p[10],w[10],v[10][10];

void main()

{

printf("\nenter the no. of items:\t");

scanf("%d",&n);

printf("\nenter the weight of the each item:\n");

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

{

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

}

printf("\nenter the profit of each item:\n");

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

{

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

}

printf("\nenter the knapsack's capacity:\t");

scanf("%d",&m);Implement 0/1 Knapsack problem using dynamic programming

knapsack();

}

void knapsack(){

int x[10];

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

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

{

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

v[i][j]=0;

}

else if(j-w[i]<0){

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

}

else {

v[i][j]=max(v[i-1][j],v[i-1][j-w[i]]+p[i]);

}

}

}

printf("\nthe output is:\n");

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

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

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

}

printf("\n\n");

}

printf("\nthe optimal solution is %d",v[n][m]);

printf("\nthe solution vector is:\n");

for(i=n;i>=1;i--){

if(v[i][m]!=v[i-1][m]) {

x[i]=1;

m=m-w[i];

}

else{

x[i]=0;

}

}

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

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

}

}

int max(int x,int y){

if(x>y) {

return x;

}

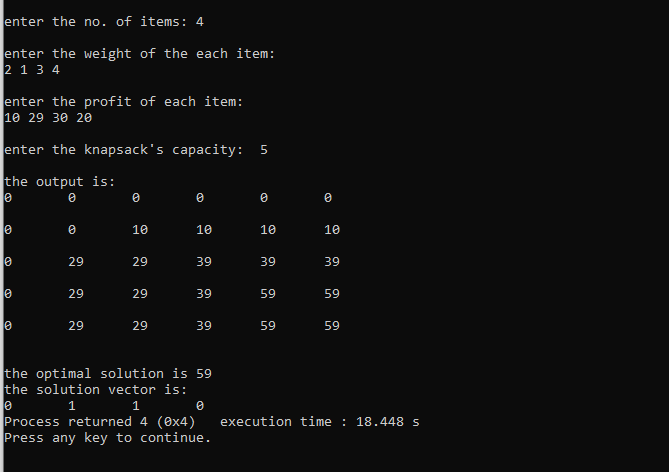
else {

return y;

}

}

**OUTPUT :**



**PROGRAM 13**

Implement All Pair Shortest paths problem using Floyd’s algorithm

#include<stdio.h>

int a[10][10],n;

void floyds();

int min(int,int);

void main()

{

int i,j;

printf("\nenter the no. of vertices:\t");

scanf("%d",&n);

printf("\nenter the cost matrix:\n");

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

{

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

{

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

}

}

floyds();

}

void floyds()

{

int i,j,k;

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

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

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

a[i][j]=min(a[i][j],a[i][k]+a[k][j]);

}

}

}

printf("\nall pair shortest path matrix is:\n");

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

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

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

}

printf("\n\n");

}

}

int min(int x,int y)

{

if(x<y){

return x;

}

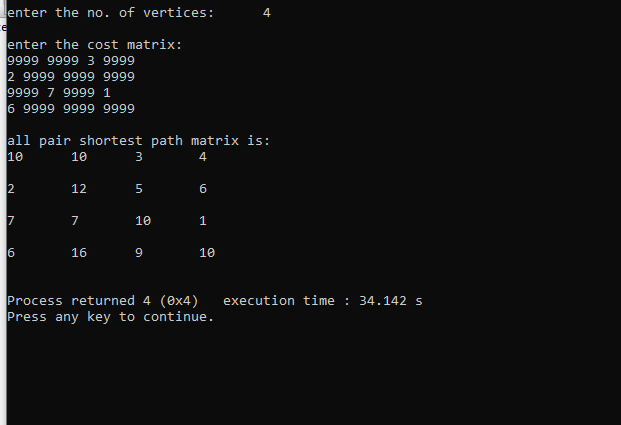
else{

return y;

}

}

**OUTPUT :**



**PROGRAM 14**

Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.

#include<stdio.h>

void prims();

int c[10][10],n;

void main()

{

int i,j;

printf("\nenter the no. of vertices:\t");

scanf("%d",&n);

printf("\nenter the cost matrix:\n");

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

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

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

}

}

prims();

}

void prims(){

int i,j,u,v,min;

int ne=0,mincost=0;

int elec[10];

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

elec[i]=0; }

elec[1]=1;

while(ne!=n-1){

min=9999;

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

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

if(elec[i]==1){

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

min=c[i][j];

u=i;

v=j;

} } } }

if(elec[v]!=1){

printf("\n%d----->%d=%d\n",u,v,min);

elec[v]=1;

ne=ne+1;

mincost=mincost+min;

}

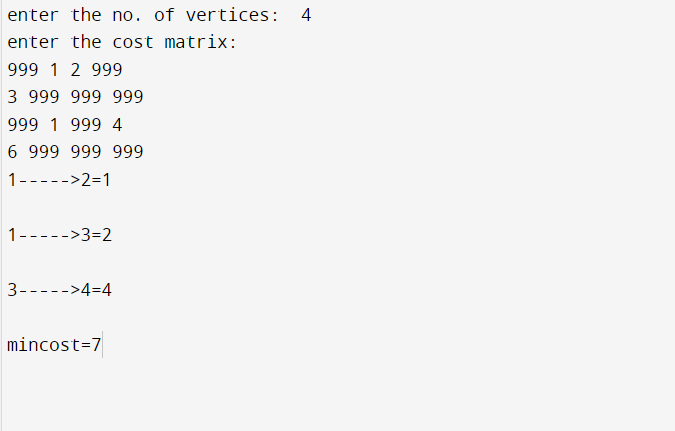
c[u][v]=c[v][u]=9999;

}

printf("\nmincost=%d",mincost);

}

**OUTPUT :**

****

**PROGRAM 15**

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.

#include<stdio.h>

void kruskals();

int c[10][10],n;

void main()

{

int i,j;

printf("\nenter the no. of vertices:\t");

scanf("%d",&n);

printf("\nenter the cost matrix:\n");

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

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

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

}

}

kruskals();

}

void kruskals(){

int i,j,u,v,a,b,min;

int ne=0,mincost=0;

int parent[10];

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

parent[i]=0;

}

while(ne!=n-1){

min=9999;

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

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

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

min=c[i][j];

u=a=i;

v=b=j;

}

}

}

while(parent[u]!=0){

u=parent[u];

}

while(parent[v]!=0){

v=parent[v];

}

if(u!=v){

printf("\n%d----->%d=%d\n",a,b,min);

parent[v]=u;

ne=ne+1;

mincost=mincost+min;

}

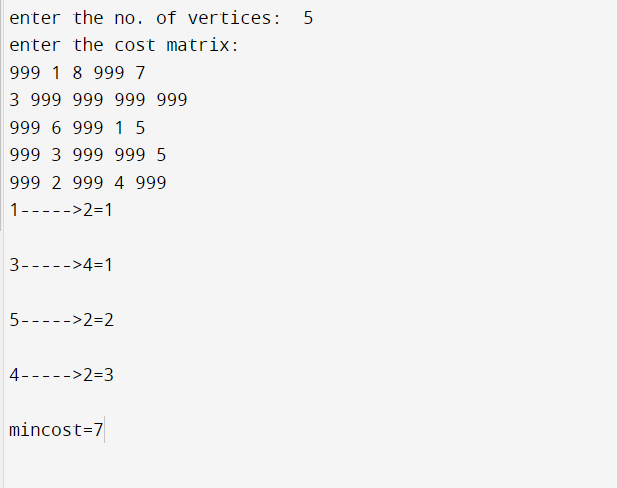
c[a][b]=c[b][a]=9999;

}

printf("\nmincost=%d",mincost);

}

**OUTPUT :**

****

**PROGRAM 16**

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.

#include<stdio.h>

void dijkstras();

int c[10][10],n,src;

void main()

{

int i,j;

printf("\nenter the no of vertices:\t");

scanf("%d",&n);

printf("\nenter the cost matrix:\n");

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

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

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

}

}

printf("\nenter the source node:\t");

scanf("%d",&src);

dijkstras();

}

void dijkstras(){

int vis[10],dist[10],u,j,count,min;

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

dist[j]=c[src][j];

}

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

vis[j]=0;

}

dist[src]=0;

vis[src]=1;

count=1;

while(count!=n){

min=9999;

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

if(dist[j]<min&&vis[j]!=1){

min=dist[j];

u=j;

}

}

vis[u]=1;

count++;

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

if(min+c[u][j]<dist[j]&&vis[j]!=1){

dist[j]=min+c[u][j];

}

}

}

printf("\nthe shortest distance is:\n");

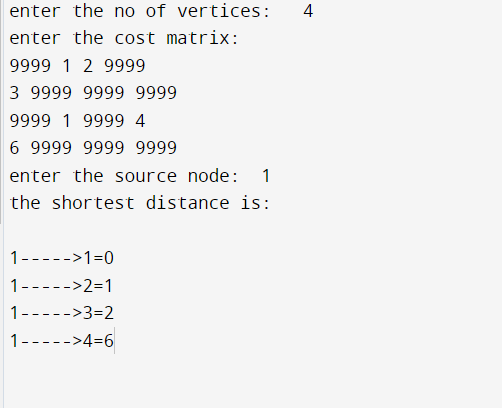
for(j=1;j<=n;j++){

printf("\n%d----->%d=%d",src,j,dist[j]);

}

}

**OUTPUT :**

****

**PROGRAM 17**

Implement “Sum of Subsets” using Backtracking. “Sum of Subsets” problem: Find a subset of a given set S = {s1,s2,……,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S = {1,2,5,6,8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn’t have a solution.

#include<stdio.h>

int count,w[10],d,x[10];

void subset(int cs, int k, int r)

{

int i;

x[k]=1;

if(cs+w[k]==d){

printf("\nSubset solution = %d\n", ++count);

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

if(x[i]==1)

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

}

}

else

if(cs+w[k]+w[k+1]<=d)

subset(cs+w[k], k+1, r-w[k]);

if((cs+r-w[k]>=d) && (cs+w[k+1])<=d){

x[k]=0;

subset(cs,k+1,r-w[k]);

}

}

void main()

{

int sum=0,i,n;

printf("Enter the number of elements\n");

scanf("%d", &n);

printf("Enter the elements in ascending order\n");

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

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

printf("Enter the required sum\n");

scanf("%d", &d);

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

sum+=w[i];

if(sum<d){

printf("No solution exists\n");

return;

}

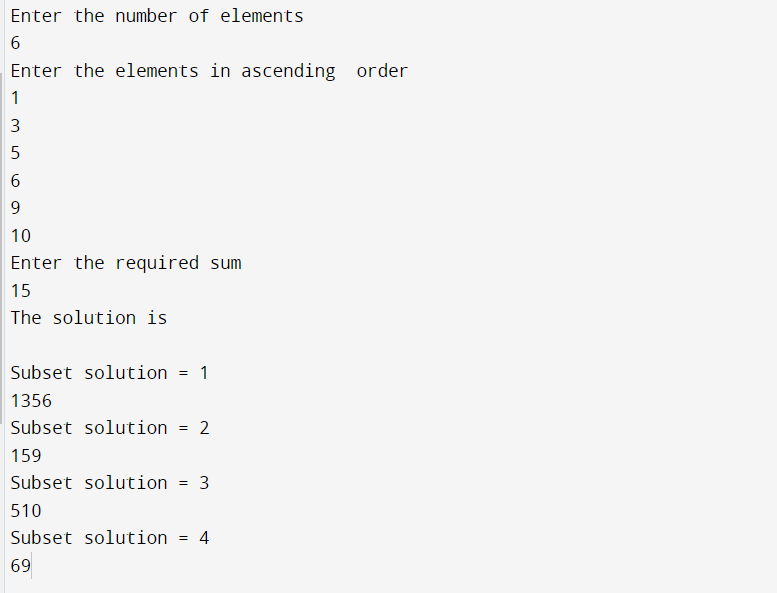
printf("The solution is\n");

count=0;

subset(0,0,sum);

}

**OUTPUT :**

****

**PROGRAM 18**

Implement “N-Queens Problem” using Backtracking.

#include<stdio.h>

void nqueens(int n)

{

int k,x[20],count=0;

k=1;

x[k]=0;

while(k!=0){

x[k]++;

while(place(x,k)!=1 && x[k]<=n)

x[k]++;

if(x[k]<=n){

if(k==n){

printf("\nSolution is %d\n", ++count);

printf("Queen\t\tPosition\n");

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

printf("%d\t\t%d\n", k,x[k]);

}else

{

k++;

x[k]=0;

}

}else

k--;

}

}

int place(int x[], int k){

int i;

for(i=1;i<=k-1;i++){

if(i+x[i]==k+x[k]||i-x[i]==k-x[k]||x[i]==x[k])

return 0;

}

return 1;

}

void main()

{

int n;

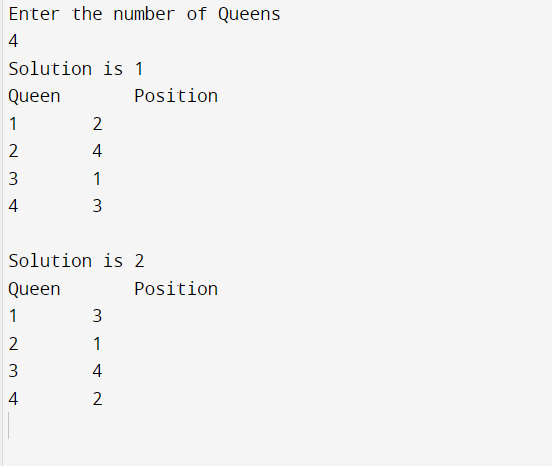
printf("Enter the number of Queens\n");

scanf("%d", &n);

nqueens(n);

}

**OUTPUT :**

****