**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

**Analysis and Design of Algorithms**

***Submitted by***

**IMADH AJAZ BANDAY (1BM20CS059)**

***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 **IMADH AJAZ BANDAY (1BM20CS059),** 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. Nagarathna N               **Dr. Jyothi S Nayak**

Professor 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 |  |
| **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. |  |
| **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. |  |
| **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. |  |
| **5** | Sort a given set of N integer elements using Insertion Sort technique and compute its time taken. |  |
| **6** | Write program to obtain the Topological ordering of vertices in a given digraph. |  |
| **7** | Implement Johnson Trotter algorithm to generate permutations. |  |
| **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. |  |
| **9** | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. |  |
| **10** | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. |  |
| **11** | Implement Warshall’s algorithm using dynamic programming |  |
| **12** | Implement 0/1 Knapsack problem using dynamic programming. |  |
| **13** | Implement All Pair Shortest paths problem using Floyd’s algorithm. |  |
| **14** | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. |  |
| **15** | Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm. |  |
| **16** | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. |  |
| **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. |  |
| **18** | Implement “N-Queens Problem” using Backtracking. |  |

**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 Title**

**Complete executed program**

**Result Screen shot**

**Graph (only for Searching and Sorting programs)**

**Program-1:**

***Write a recursive program to Solve***

1. ***Towers-of-Hanoi problem b) To find GCD***
2. **Tower Of Hanoi**

#include <stdio.h>

#include <conio.h>

#include <math.h>

void hanoi(int x, char from, char to, char aux)

{

if(x==1)

printf("Move disk from %c to %c\n",from,to);

else

{

hanoi(x-1,from,aux,to);

printf("Move disk from %c to %c\n",from,to);

hanoi(x-1,aux,to,from);

}

}

void main()

{

int disk;

int moves;

printf("Enter the number of disks you want to play with:");

scanf("%d",&disk);

moves=pow(2,disk)-1;

printf("The number of moves required is:%d\n",moves);

hanoi(disk,'A','C','B');

}

***OUTPUT:***



**b. GCD**

#include <stdio.h>

int gcd(int n1,int n2);

int main()

{

int n1,n2;

printf("Enter two positive integers:");

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

printf("GCD of %d and %d is %d",n1,n2,gcd(n1,n2));

return 0;

}

int gcd(int n1, int n2)

{

if(n2!=0)

return gcd(n2,n1%n2);

else

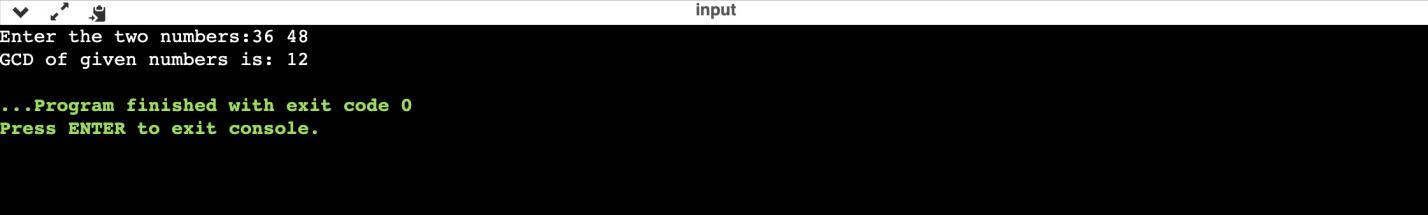
{

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<stdlib.h>

#include<time.h>

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

int lin\_srch(int [], int ,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("1.Binary Search 2.Linear Search 3. Exit\n ");

printf("Enter 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("key not found \n");

else

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

for(j=0;j<5000000;j++){temp = 38/600;}

end= clock();

printf("\n time 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(1000);

a[i] = i;

}

key = a[n-1];

start = clock();

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

if(search\_status == -1)

printf("key not found \n");

else

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

for(j=0;j<5000000;j++){temp = 38/600;}

end= clock();

printf("\n time 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 low,int high,int key)

{

if(low>high)

{

return -1;

}

if(key == a[low])

{

return low;

}

else

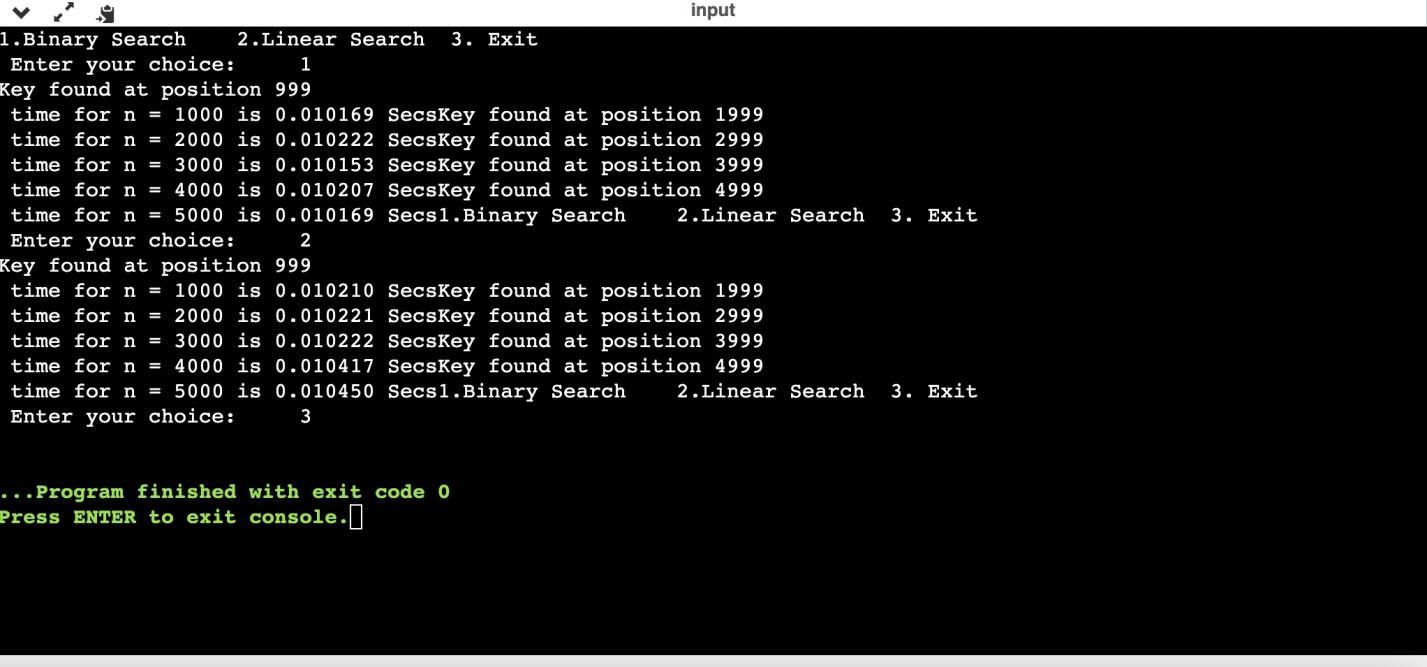
{

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

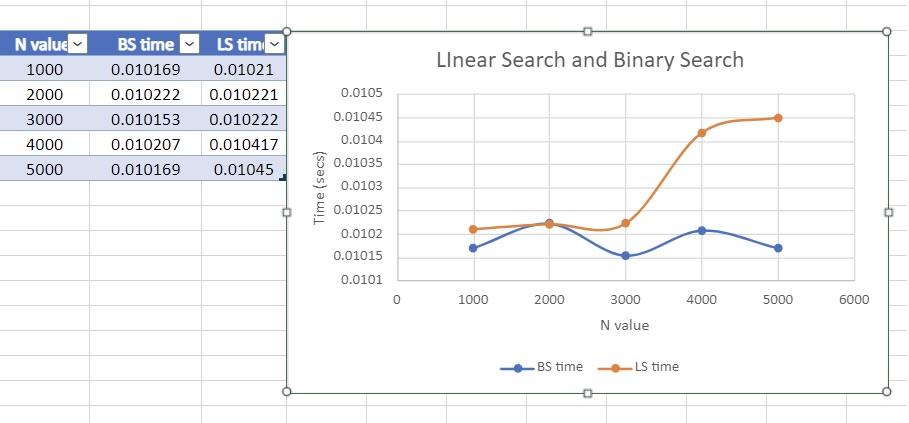
}

}

***OUTPUT:***

******

***TABLE VALUES AND GRAPH:***

******

**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("\n 1: For manual entry of N values and array elements:");

printf("\n 2: To display time taken for sorting number of elements N in the range 500 to 14500:");

printf("\n 3: To exit");

printf("\n Enter your choice:");

scanf("%d",&ch);

switch(ch)

{

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

scanf("%d",&n);

printf("\n Enter array elements:");

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

{

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

}

start=clock();

selsort(n,a);

end=clock();

printf("\n Sorted 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]=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);

}

}

}

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];

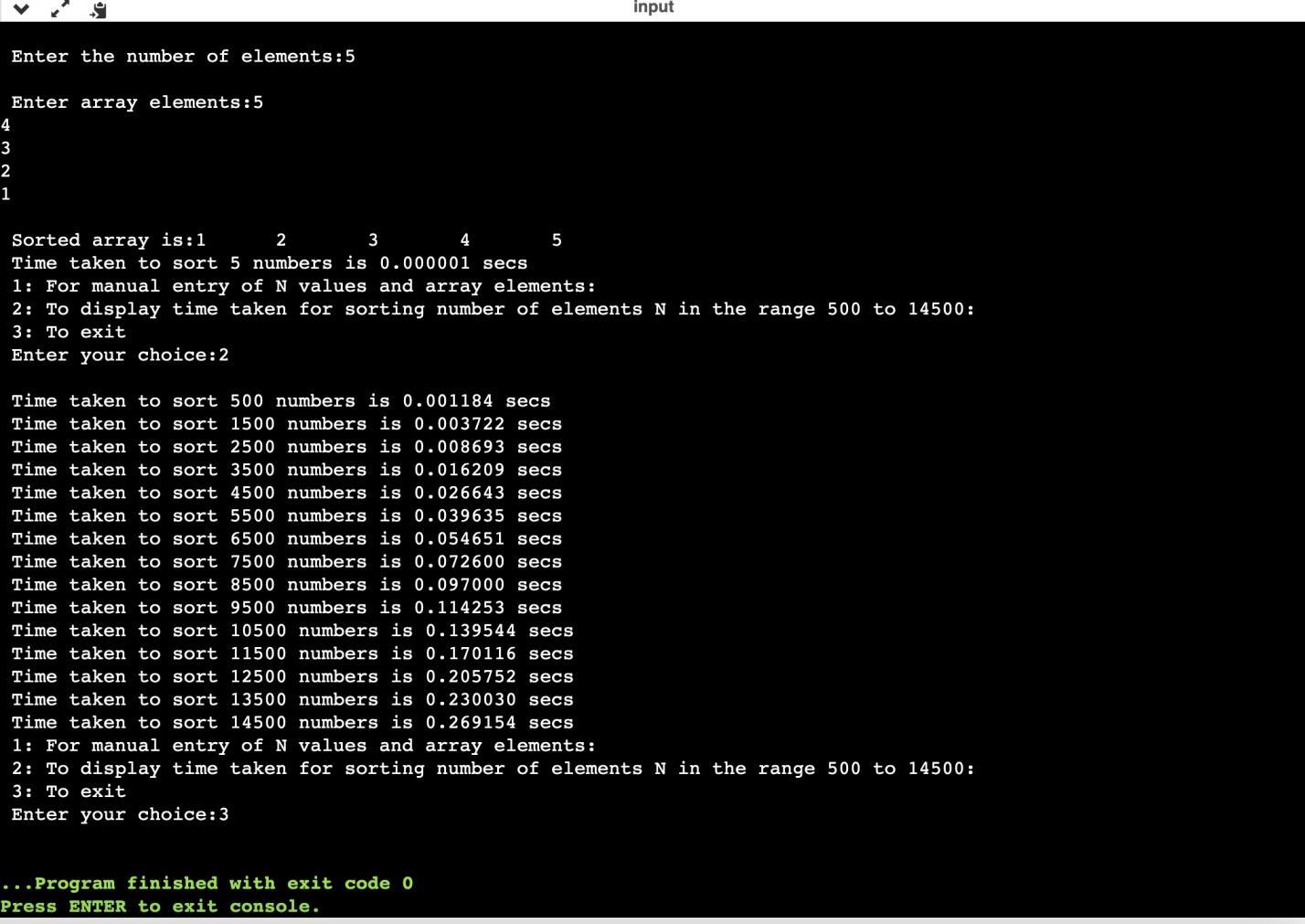
a[i]=a[pos];

a[pos]=t;

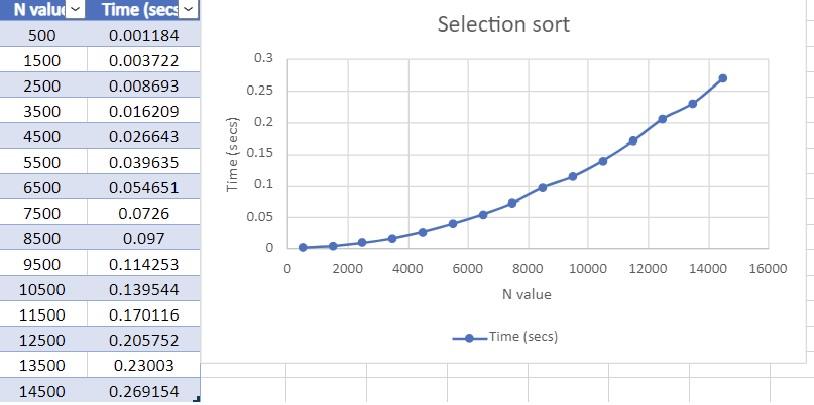
}

}

***OUTPUT:***

****

***TABLE VALUES AND GRAPH:***

****

**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.***

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

***method.***

**a. BFS**

#include<stdio.h>

#include<conio.h>

int a[10][10],n;

void bfs(int);

void main()

{

int i,j,src;

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);

bfs(src);

getch();

}

void bfs(int src)

{

int q[10],f=0,r=-1,vis[10],i,j;

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

{

vis[j]=0;

}

vis[src]=1;

r=r+1;

q[r]=src;

while(f<=r)

{

i=q[f];

f=f+1;

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

{

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

{

vis[j]=1;

r=r+1;

q[r]=j;

}

}

}

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

{

if(vis[j]!=1)

{

printf("\nnode %d is not reachable\n",j);

}

else

{

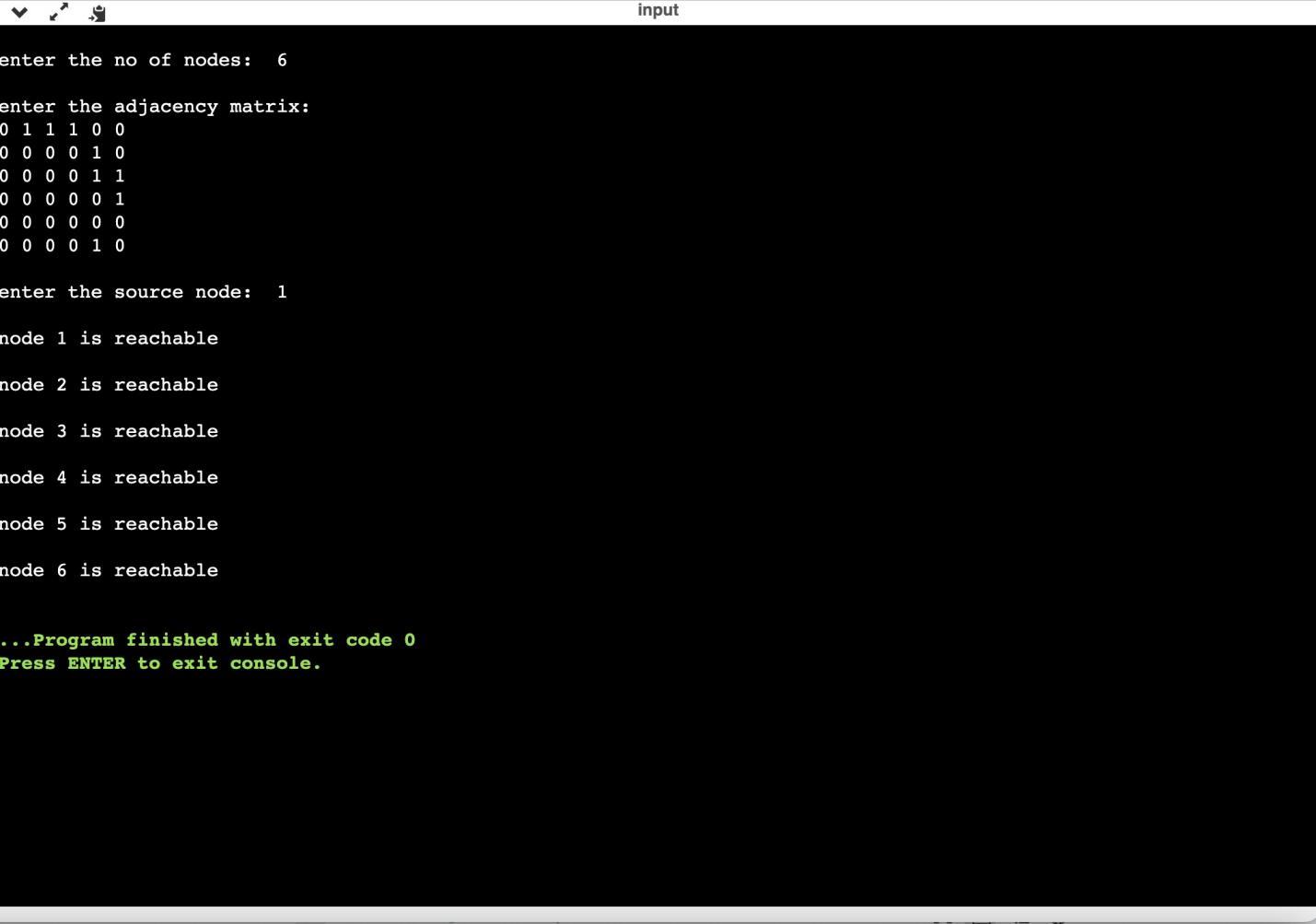
printf("\nnode %d is reachable\n",j);

}

}

}

***OUTPUT:***

****

**b. DFS**

#include<stdio.h>

#include<conio.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");

}

getch();

}

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;

}

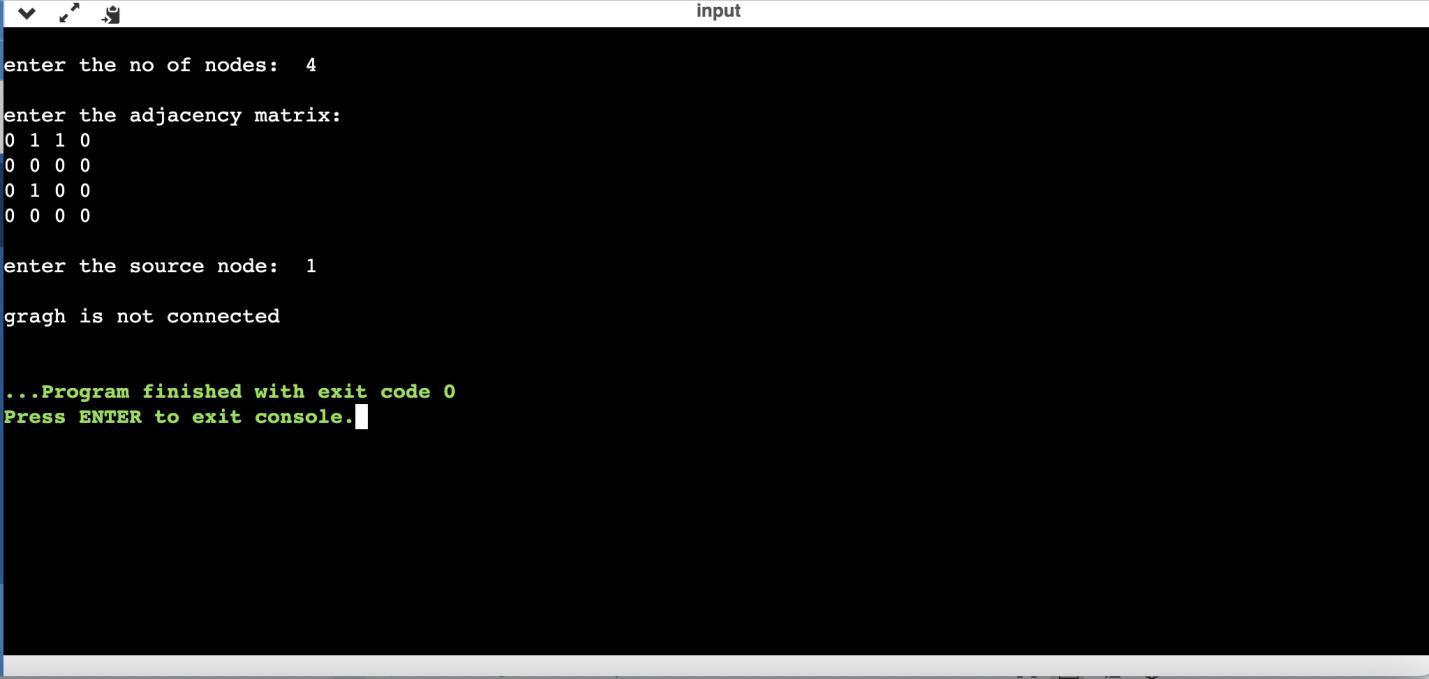
}

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<conio.h>

#include<stdlib.h>

#include<time.h>

/\*void printarray(int a[], int size) {

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

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

}

printf("\n");

}\*/

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

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

int key = a[i];

int j = i - 1;

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

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

--j;

}

a[j + 1] = key;

}

}

int main()

{

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

int ch,n;

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");

printf("\n3:Exit");

printf("\nEnter your choice:");

scanf("%d",&ch);

switch(ch)

{

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

scanf("%d",&n);

printf("\nEnter the array elements: ");

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

{

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

}

start=clock();

insertionsort(a,n);

end=clock();

printf("\nsorted array");

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

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

}

//printarray(a,n);

printf("\ntime taken to sort %d number of elements is %f secs",n,(((double)(end-start)/CLOCKS\_PER\_SEC)));

break;

case 2:

printf("\nrRunning values from 500 to 14500");

n=500;

while(n<=14500)

{

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

{

a[i]=rand();

}

start=clock();

insertionsort(a,n);

for(j=0;j<5000;j++)

{

int temp=38/600;

}

end=clock();

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

n=n+1000;

}

break;

case 3:exit(0);

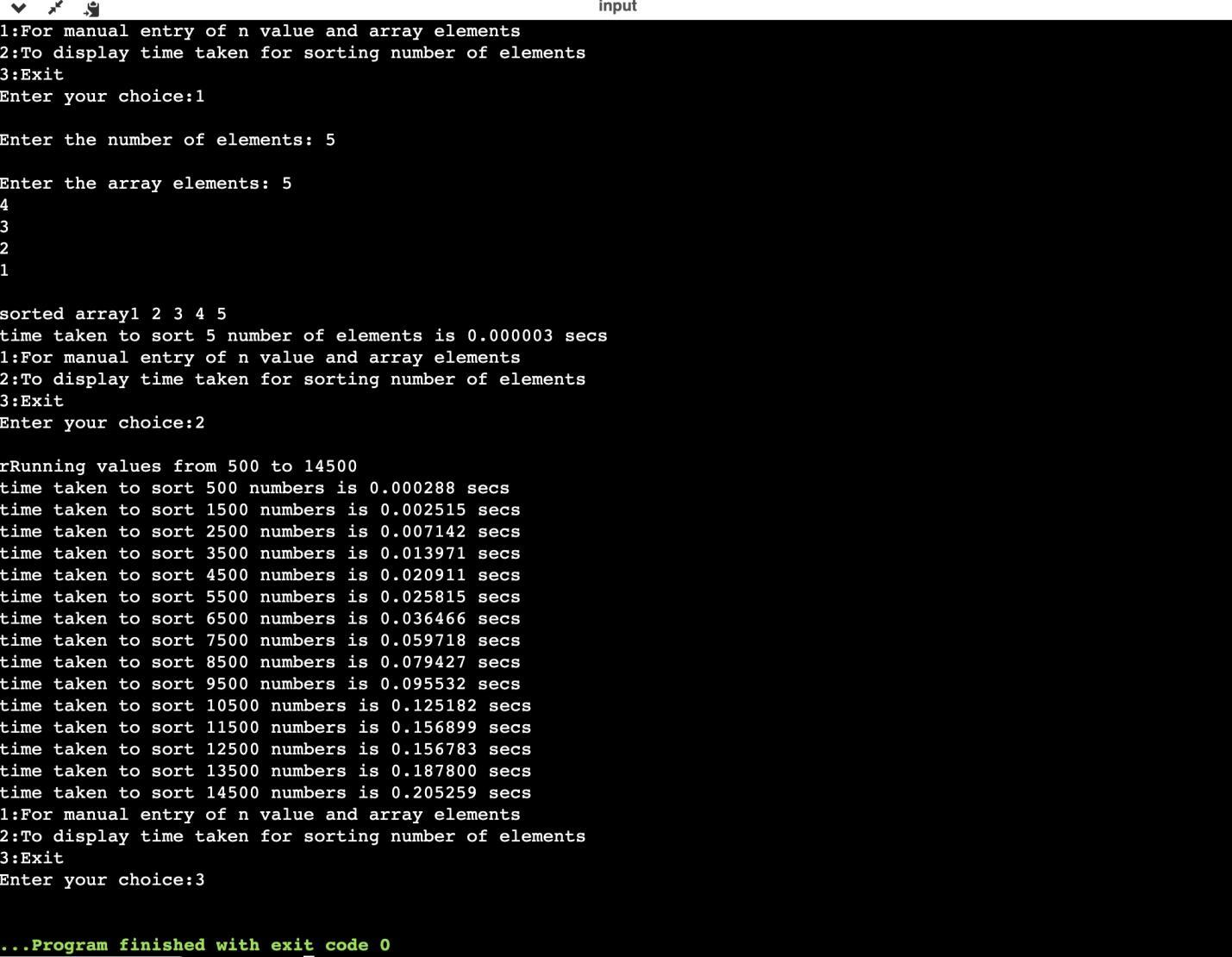
}

}

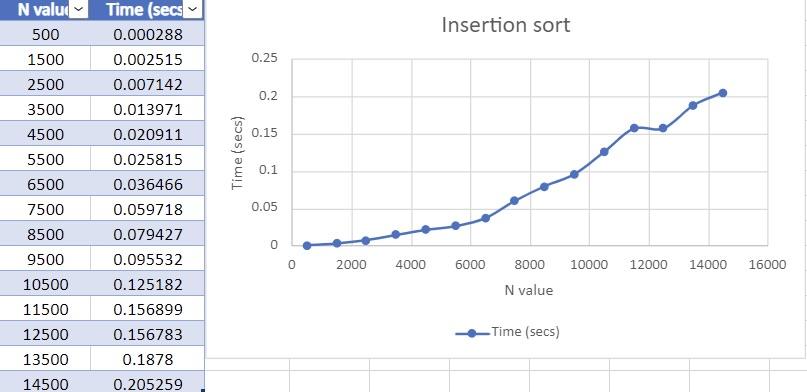
return 0;

}

***OUTPUT:***

****

***TABLE VALUES AND GRAPH:***

******

**Program-6:**

***Write program to obtain the Topological ordering of vertices in a***

***given digraph.***

#include<stdio.h>

#include<conio.h>

void source\_removal(int n, int a[10][10]) {

int i, j, k, u, v, top, s[10], t[10], indeg[10], sum;

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

sum = 0;

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

sum += a[j][i];

}

indeg[i]=sum;

}

top = -1;

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

if(indeg[i] == 0) {

s[++top] = i;

}

}

k = 0;

while(top != -1) {

u = s[top--];

t[k++] = u;

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

if(a[u][v] == 1) {

indeg[v] = indeg[v] - 1;

if(indeg[v] == 0)

s[++top] = v;

}

}

}

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

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

}

}

void main() {

int i, j, a[10][10], n;

printf("Enter number of nodes\n");

scanf("%d", &n);

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

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

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

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

}

}

source\_removal(n,a);

getch();

}

***OUTPUT:***

******

**Program-7:**

***Implement Johnson Trotter algorithm to generate permutations***

#include <stdio.h>

#include <stdlib.h>

int flag = 0;

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

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int search(int arr[],int num,int mobile)

{

int g;

for(g=0;g<num;g++)

{

if(arr[g] == mobile)

{

return g+1;

}

else

{

flag++;

}

}

return -1;

}

int find\_Moblie(int arr[],int d[],int num)

{

int mobile = 0;

int mobile\_p = 0;

int i;

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

{

if((d[arr[i]-1] == 0) && i != 0)

{

if(arr[i]>arr[i-1] && arr[i]>mobile\_p)

{

mobile = arr[i];

mobile\_p = mobile;

}

else

{

flag++;

}

}

else if((d[arr[i]-1] == 1) & i != num-1)

{

if(arr[i]>arr[i+1] && arr[i]>mobile\_p)

{

mobile = arr[i];

mobile\_p = mobile;

}

else

{

flag++;

}

}

else

{

flag++;

}

}

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

return 0;

else

return mobile;

}

void permutations(int arr[],int d[],int num)

{

int i;

int mobile = find\_Moblie(arr,d,num);

int pos = search(arr,num,mobile);

if(d[arr[pos-1]-1]==0)

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

else

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

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

{

if(arr[i] > mobile)

{

if(d[arr[i]-1]==0)

d[arr[i]-1] = 1;

else

d[arr[i]-1] = 0;

}

}

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

{

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

}

}

int factorial(int k)

{

int f = 1;

int i = 0;

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

{

f = f\*i;

}

return f;

}

int main()

{

int num = 0;

int i;

int j;

int z = 0;

printf("Johnson trotter algorithm to find all permutations of given numbers \n");

printf("Enter the number\n");

scanf("%d",&num);

int arr[num],d[num];

z = factorial(num);

printf("%d",z);

printf("\nAll possible permutations are: \n");

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

{

d[i] = 0;

arr[i] = i+1;

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

}

printf("\n");

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

{

permutations(arr,d,num);

printf("\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 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<=15000)

{

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

{

a[i]=n-i;

}

start=clock();

split(a,0,n-1);

for(j=0;j<50000000;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++;

}

}

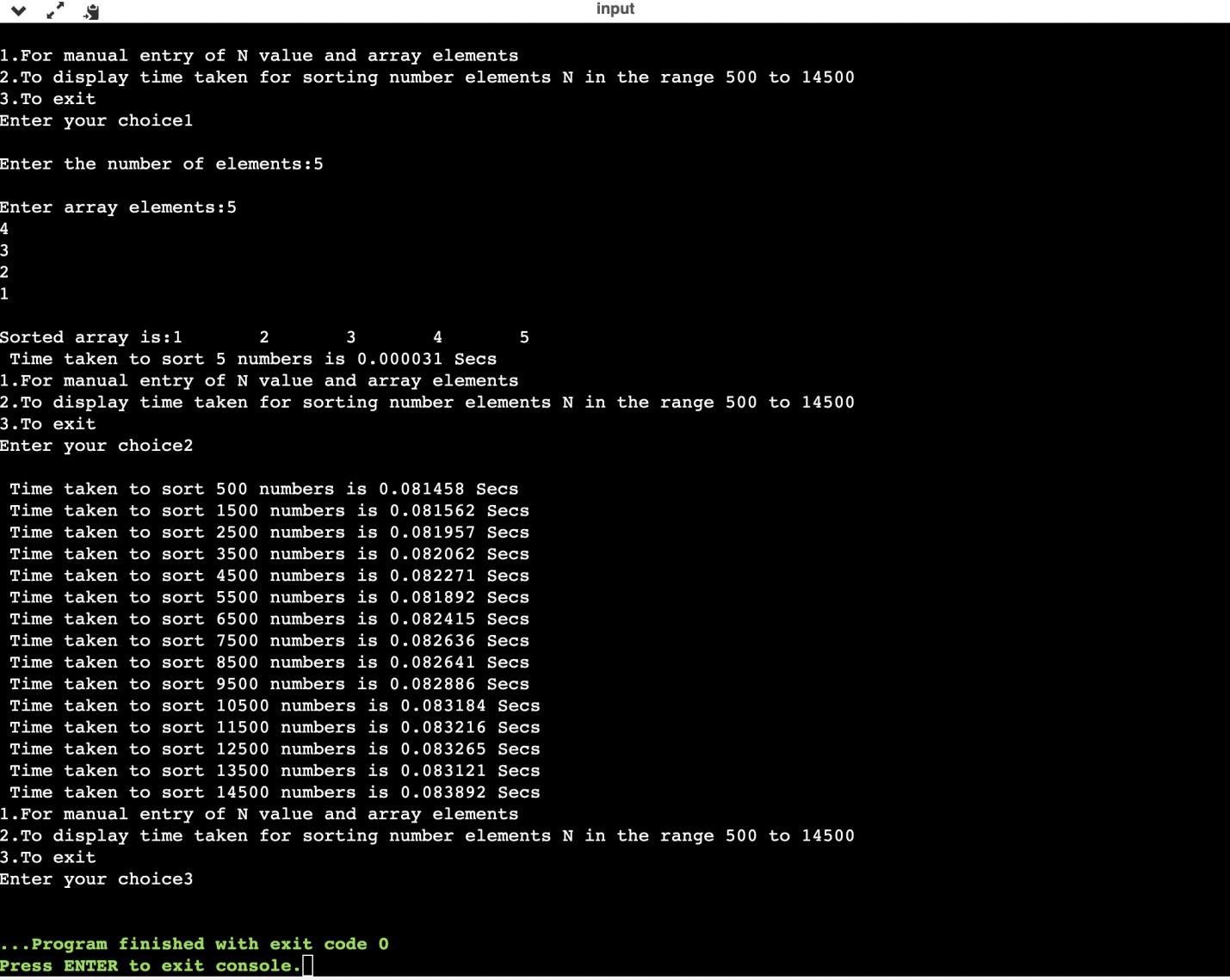
for(i=low;i<=high;i++){

a[i]=c[i];

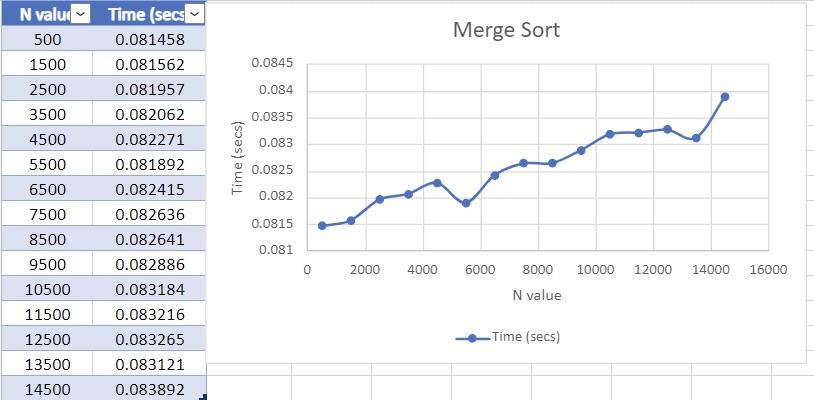
}

}

***OUTPUT:***



***TABLE VALUES AND GRAPH:***



**Program-9:**

***Sort a given set of N integer elements using Quick Sort technique***

***and compute its time taken.***

#include<stdio.h>

#include<conio.h>

#include<time.h>

#include<stdlib.h>

int partition(int a[],int low,int high)

{

int start=low;

int end=high;

int pivot=a[low];

int i,j,temp;

while(start<end)

{

while(a[start]<=pivot)

{

start++;

}

while(a[end]>pivot)

{

end--;

}

if(start<end)

{

temp=a[start];

a[start]=a[end];

a[end]=temp;

}

}

temp=a[low];

a[low]=a[end];

a[end]=temp;

return end;

}

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

{

int loc;

if(low<high)

{

loc=partition(a,low,high);

quicksort(a,low,loc-1);

quicksort(a,loc+1,high);

}

}

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 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();

quicksort(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<=15000)

{

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

{

a[i]=n-i;

}

start=clock();

quicksort(a,0,n-1);

for(j=0;j<50000000;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);

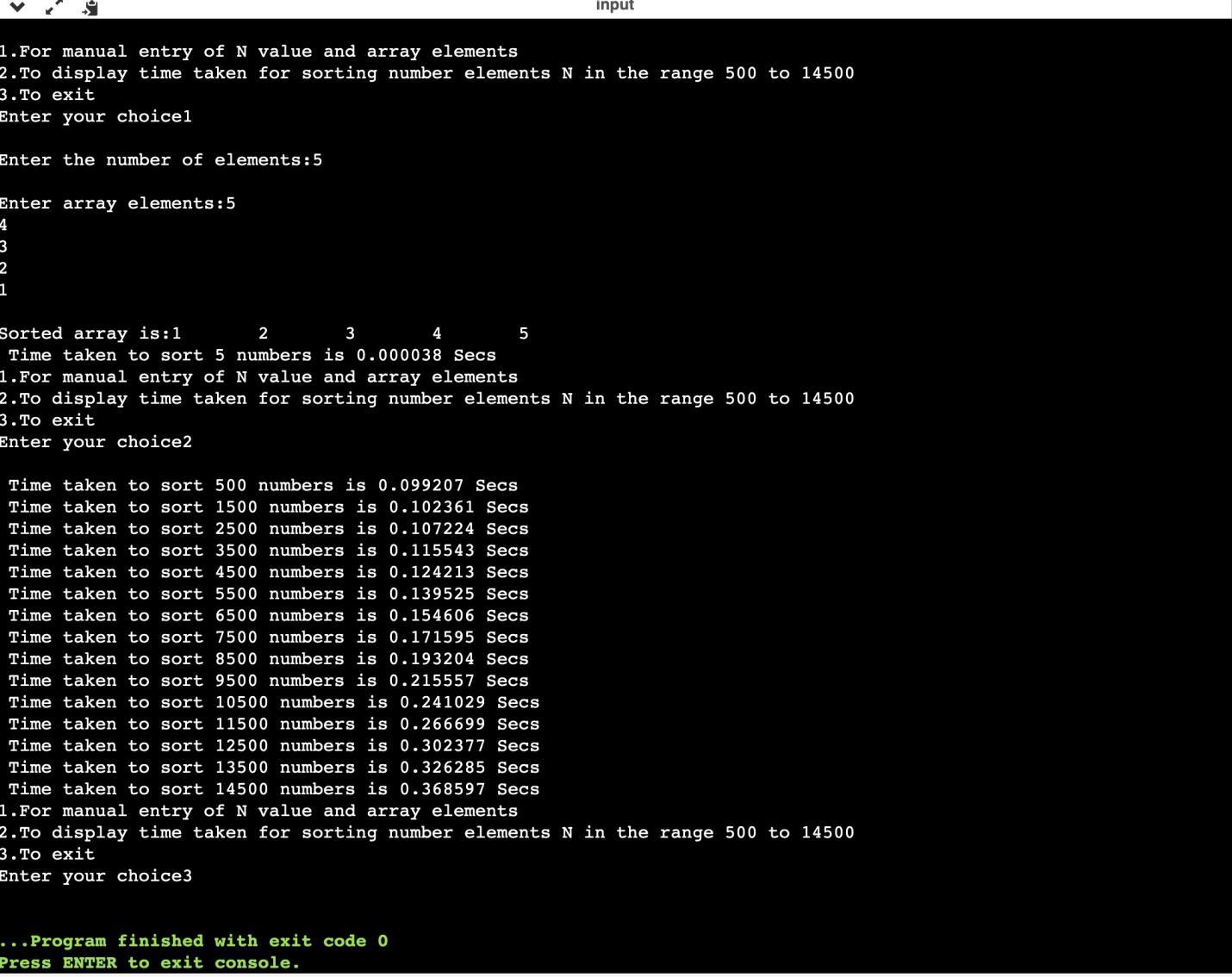
}

getchar();

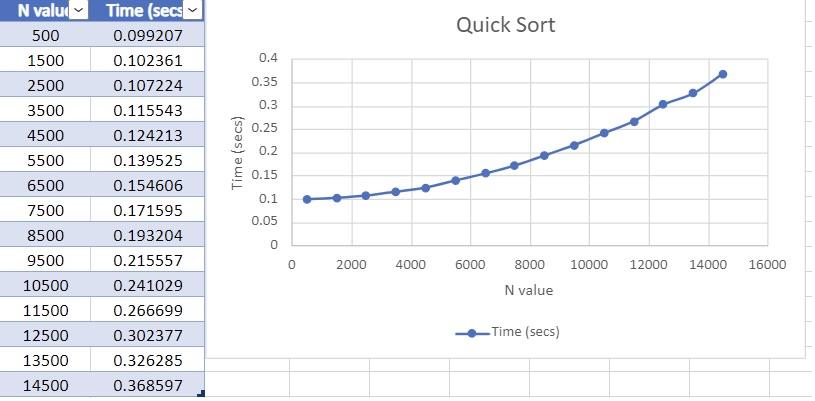
}

}

***OUTPUT:***



***TABLE VALUES AND GRAPH:***

****

**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>

#include <stdlib.h>

#include <math.h>

void swap(int \*,int \*);

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

void heapSort(int[], int);

void main()

{

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

clock\_t start,end;

while(1)

{

printf("\n 1: For manual entry of N values and array elements:");

printf("\n 2: To display time taken for sorting number of elements N in the range 500 to 14500:");

printf("\n 3: To exit");

printf("\n Enter your choice:");

scanf("%d",&ch);

switch(ch)

{

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

scanf("%d",&n);

printf("\n Enter array elements:");

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

{

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

}

start=clock();

heapSort(a, n);

end=clock();

printf("\n Sorted array is:");

for(i=n-1;i>=0;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]=n-i;

}

start=clock();

heapSort(a, n);

for(j=0;j<50000000;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);

}

}

}

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--)

{

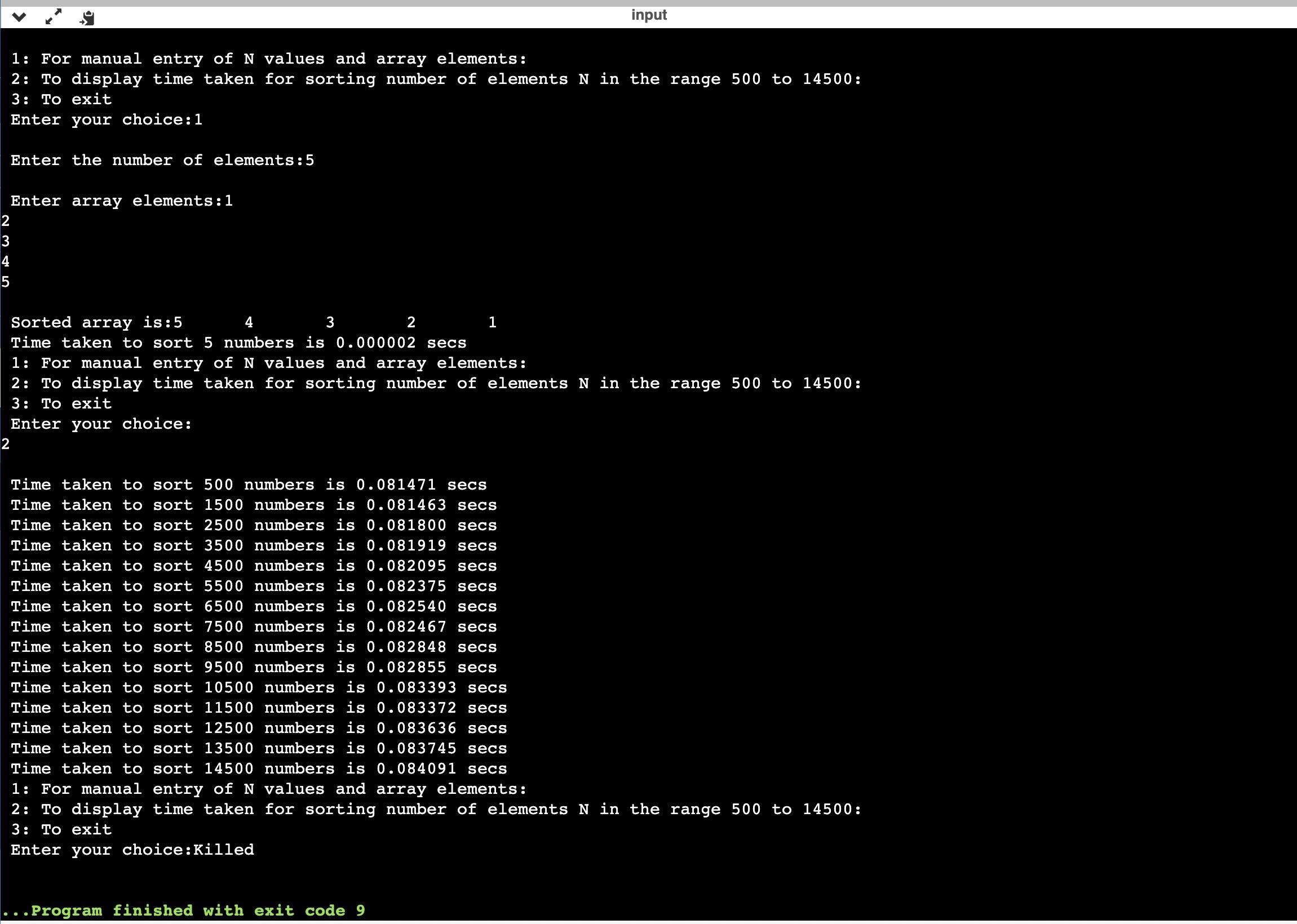
swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

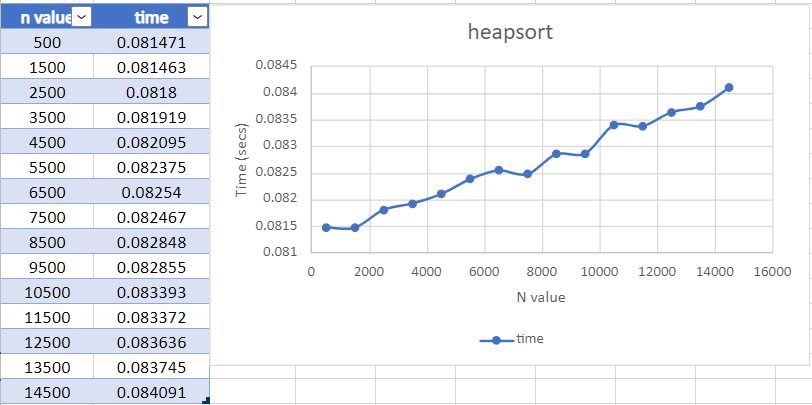
}

}

***OUTPUT:***

****

***TABLE VALUES AND GRAPH:***

****

**Program-11:**

***Implement Warshall’s algorithm using dynamic programming.***

#include<stdio.h>

int a[10][10],r[10][10][10];

void warshall(int n){

int k=0;

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

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

r[k][i][j]=a[i][j];

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

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

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

r[k][i][j]=r[k-1][i][j] || (r[k-1][i][k] && r[k-1][k][j]);

}

int main(){

int n;

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

scanf("%d",&n);

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

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

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

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

warshall(n);

printf("Transitive Closure:\n");

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

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

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

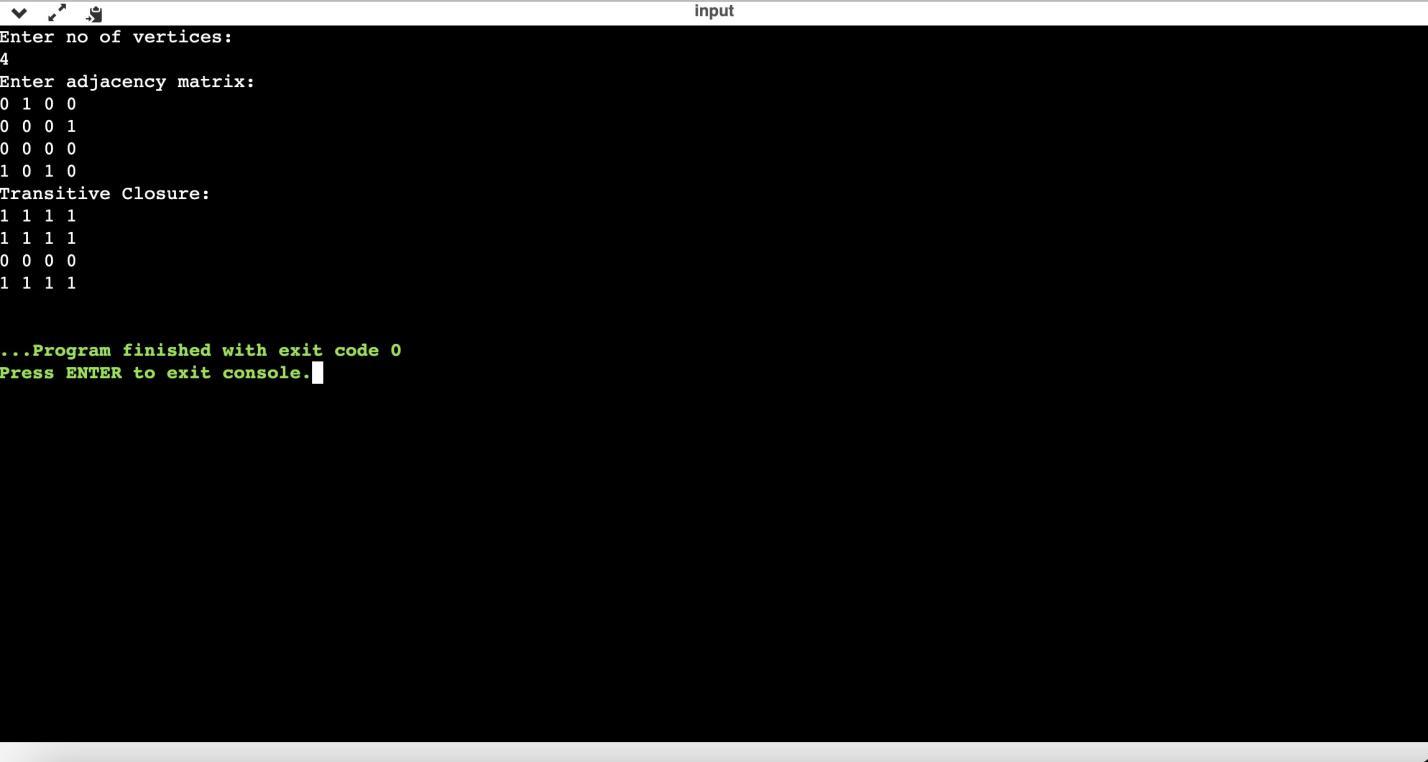
}

printf("\n");

}

}

***OUTPUT:***



**Program-12:**

***Implement 0/1 Knapsack problem using dynamic programming.***

#include <stdio.h>

#include<conio.h>

int max(int,int);

void knapsack();

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

int main()

{

printf("\nenter the number of items\n");

scanf("%d",&n);

printf("\nenter the weight of 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 capacity \n");

scanf("%d",&m);

knapsack();

return 0;

}

void knapsack()

{

int x[10];

k=0;

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("\Output matrix is\n");

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

{

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

{

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

}

printf("\n\n");

}

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

printf("\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 ",x[i]);

}

}

int max(int x,int y)

{

if(x>y)

{

return x;

}

else

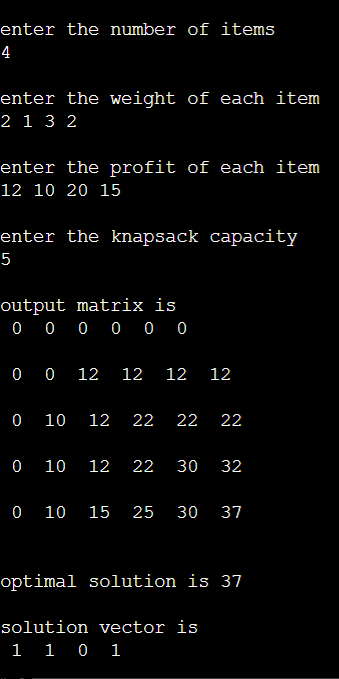
{

return y;

}

}

***OUTPUT:***

****

**Program-13 :**

***Implement All Pair Shortest Paths problem using Floyd’s algorithm.***

#include<stdio.h>

#include<conio.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();

getch();

}

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

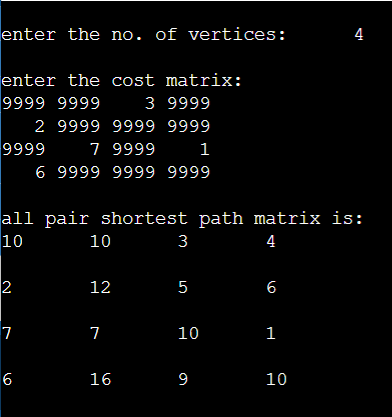
{

return y;

}

}

***OUTPUT:***

****

**Program-14 :**

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

#include<stdio.h>

#include<conio.h>

void prims();

int c[10][10],n;

int 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();

return 0;

}

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;

}

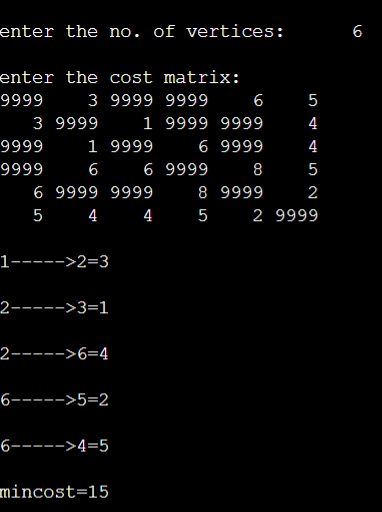
c[u][v]=c[v][u]=9999;

}

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

}

**output:**

******

**Program-15:**

***Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s 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;

}

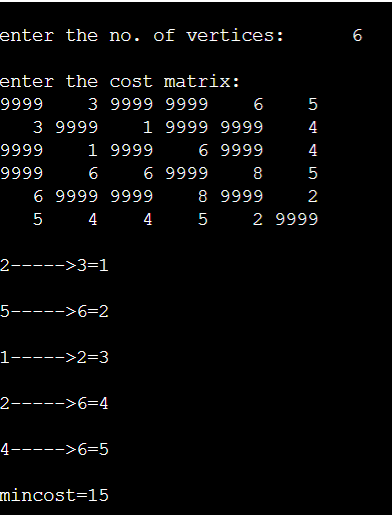
c[a][b]=c[b][a]=9999;

}

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>;

#include<conio.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();

getch();

}

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("\the shortest distance is:\n");

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

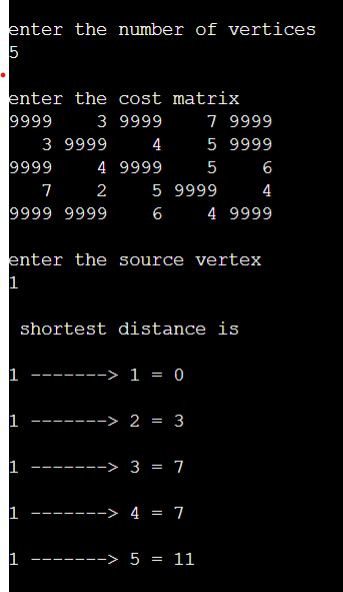
{

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

}

}

**output:**

******