CS3401 ALGORITHMS L T P C 3 0 2 4

PRACTICAL EXERCISES: 30 PERIODS

Searching and Sorting Algorithms

1. Implement Linear Search. Determine the time required to search for an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n
2. Implement recursive Binary Search. Determine the time required to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.
3. Given a text txt [0...n-1] and a pattern pat [0...m-1], write a function search (char pat [ ], char txt [ ]) that prints all occurrences of pat [ ] in txt [ ]. You may assume that n >m.
4. Sort a given set of elements using the Insertion sort and Heap sort methods and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

Graph Algorithms

5. Develop a program to implement graph traversal using Breadth First Search
6. Develop a program to implement graph traversal using Depth First Search
7. From a given vertex in a weighted connected graph, develop a program to find the shortest paths to other vertices using Dijkstra's algorithm.
8. Find the minimum cost spanning tree of a given undirected graph using Prim's algorithm.
9. Implement Floyd's algorithm for the All-Pairs- Shortest-Paths problem.
10. Compute the transitive closure of a given directed graph using Warshall's algorithm.

Algorithm Design Techniques

11. Develop a program to find out the maximum and minimum numbers in a given list of n numbers using the divide and conquer technique.
12. Implement Merge sort and Quick sort methods to sort an array of elements and determine the time required to sort. Repeat the experiment for different values of n,

the number of elements in the list to be sorted and plot a graph of the time taken versus n.

State Space Search Algorithms:

13. Implement N Queens problem using Backtracking. Approximation Algorithms Randomized Algorithms
14. Implement any scheme to find the optimal solution for the Traveling Salesperson problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.
15. Implement randomized algorithms for finding the kth smallest number.

The programs can be implemented in C/C++/JAVA/ Python.

Searching and Sorting Algorithms

1. Implement Linear Search. Determine the time required to search for an element. Repeat the experiment for different values of *n*, the number of elements in the list to be searched and plot a graph of the time taken versus *n*.

AIM:

To write a program for linear search algorithm.

ALGORITHM:

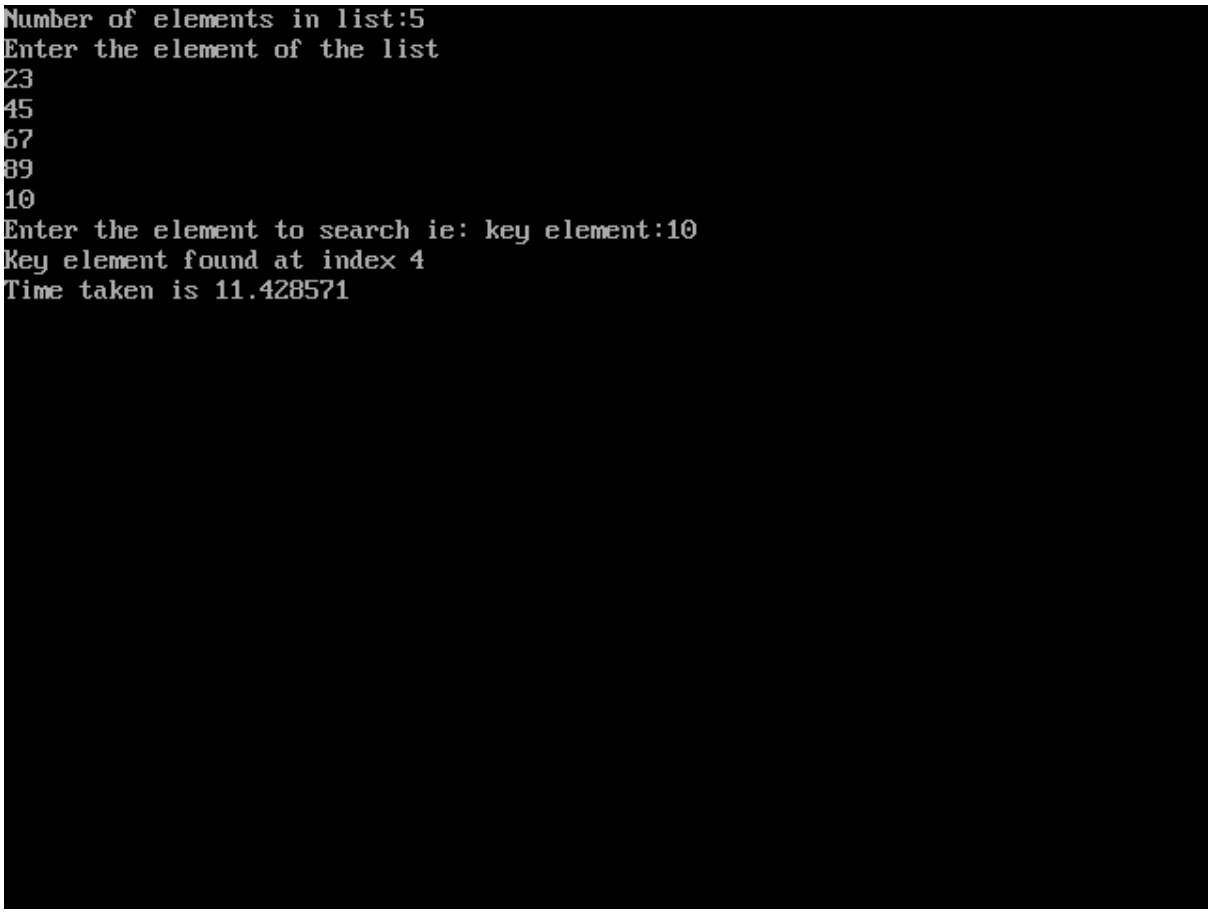
STEP1: Start
STEP2: Get the list elements from the user
STEP3: Set the target element.
STEP4: Select the first elements as the current element.
STEP5: Compare to current element with the target element
STEP6: If there is a next element,then set current element to next element and go to step2
STEP7: Target element not found, go to step9
STEP8: Target element found ,return location & print the execution time
STEP9: Stop

PROGRAM:

```
# include<stdio.h>
# include<time.h>
int main()
{
  int arr[20], size, key, i, index;
  clock_t start, end;
  clrscr();
  start = clock();
  printf("Number of elements in the list: ");
  scanf("%d", &size);
  printf("Enter elements of the list: ");
  // loop for the input of elements from 0 to number of elements-1
  for (i = 0; i < size; i++)
    scanf("%d", &arr[i]);
  printf("Enter the element to search ie. key element: ");
  scanf("%d", &key);
  // loop for traversing the array from 0 to the number of elements-1
  for (index = 0; index < size; index++)
    if (arr[index] == key) // comparing each element with the key element
      break; // cursor out of the loop when a key element found

  if (index < size) // condition to check whether previous loop partially traversed or not
    printf("Key element found at index %d /n", index); // printing the index if key found
  else
    printf("Key element not found /n");
```

```
end = clock();
double execution_time = (((double)(end - start))/CLOCKS_PER_SEC);
printf("Time taken is  %f",execution_time);
getch();
return 0;
}
```

OUTPUT:



RESULT:

Thus the program was executed successfully.

2. Implement recursive Binary Search. Determine the time required to search an element. Repeat the experiment for different values of *n*, the number of elements in the list to be searched and plot a graph of the time taken versus *n*.

AIM:

To write a program for binary search algorithm

ALGORITHM:

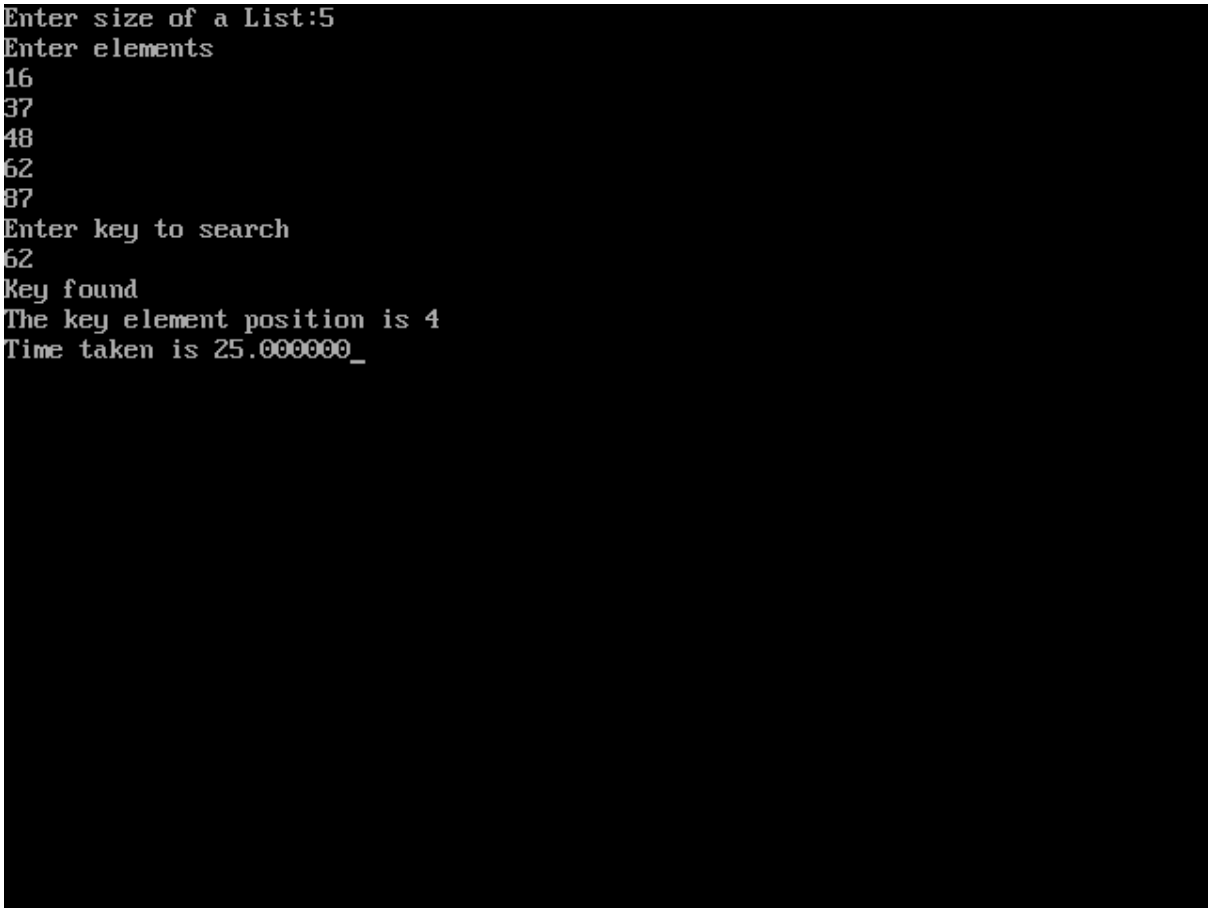
STEP1: Start .
STEP2: Get the list elements from the user.
STEP3: Set the target element.
STEP4: Find the middle element in the sorted list.
STEP5: If target & middle element are matched , then return the position & go to step9.
STEP6: If both are not matched ,then check whether the target element is smaller or larger than the middle element.
STEP7: If target is larger then middle element set low=mid+1,then repeat the step4,5,6
STEP8: If target is larger then middle element set high=mid-1,then repeat the step4,5,6
STEP6: Stop.

PROGRAM:

```
# include<stdio.h>
# include<time.h>
void binary_search(int [], int, int, int);
int main()
{
 int key, size, i;
   int list[25];
   clock_t start, end;
   clrscr();
   start = clock();
    printf("Enter size of a list: ");
   scanf("%d", &size);
   printf("Enter elements\n");
   for(i = 0; i < size; i++)
   {
      scanf("%d",&list[i]);
   }
   printf("Enter key to search\n");
   scanf("%d", &key);
   binary_search(list, 0, size, key);
   end = clock();
   double execution_time = (((double)(end - start))/CLOCKS_PER_SEC);
   printf("Time taken is  %f",execution_time);
   getch();
   return 0;
 }
 void binary_search(int list[], int lo, int hi, int key)
{
```

```
    int mid;
     if (lo > hi)
    {
       printf("Key not found\n");
       return;
    }
    mid = (lo + hi) / 2;
    if (list[mid] == key)
    {
       printf("Key found\n");
    }
    else if (list[mid] > key)
    {
       binary_search(list, lo, mid - 1, key);
    }
    else if (list[mid] < key)
    {
       binary_search(list, mid + 1, hi, key);
    }
}
```

OUTPUT:



RESULT:

Thus the program for executed successfully.

3. Given a text txt [0...n-1] and a pattern pat [0...m-1], write a function search (char pat [ ], char txt [ ]) that prints all occurrences of pat [ ] in txt [ ]. You may assume that n > m.

AIM:

To write a program for text and pattern search.

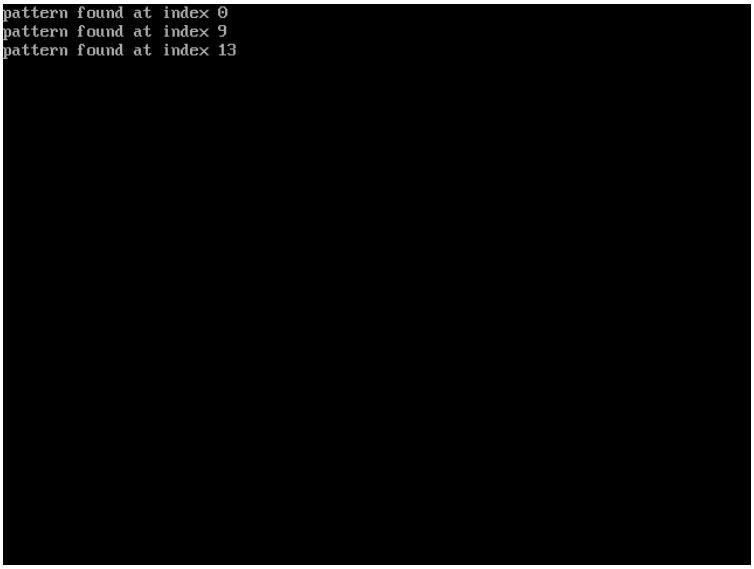
ALGORITHM:

STEP1: Start .
STEP2: Get the text from the user.
STEP3: Get the Pattern from the user
STEP4: Calculate the len of the text and the pattern given
STEP5: use for loop to find the pattern matching with the text.
STEP6: If any matching found print the result

PROGRAM:

```
#include <stdio.h>
#include <string.h>
void search(char* pat, char* txt)
{
      int M = strlen(pat);
      int N = strlen(txt);
      int  i
      for ( i = 0; i <= N - M; i++)
      {
         int j;
         for (j = 0; j < M; j++)
           if (txt[i + j] != pat[j])
             break;
           if (j == M) // if pat[0...M-1] = txt[i, i+1, ...i+M-1]
           printf("Pattern found at index %d \n", i);
      }
}
int main()
{
      char txt[] = "AABAACAADAABAAABAA";
      char pat[] = "AABA";
      search(pat, txt);
      return 0;
}
```

OUTPUT:



RESULT:

Thus the program was executed successfully.

4. Sort a given set of elements using the Insertion sort and Heap sort methods and determine the time required to sort the elements. Repeat the experiment for different values of *n*, the number of elements in the list to be sorted and plot a graph of the time taken versus *n*.

4(A) INSERTION SORT:

AIM:

To write program for insertion sort algorithm

ALGORITHM:

STEP1: Start
STEP2: Read number of array elements n
STEP3: Read array elements Ai
STEP4: Sort the elements using insertion sort
In pass p, move the element in position p left until its correct place is found among the first p + 1 elements.
Element at position p is saved in temp, and all larger elements (prior to position p) are moved one spot to the right. Then temp is placed in the correct spot.
STEP5: Stop

PROGRAM:

// INSERTION SORT

```
#include <stdio.h>
#include<time.h>
int main(void)
{
   clock_t start, end;
   start = clock();
   int n, i, j, temp;
   int arr[64];
   printf("Enter number of elements\n");
   scanf("%d", &n);
   printf("Enter %d integers\n", n);
   for (i = 0; i < n; i++)
   {
      scanf("%d", &arr[i]);
   }
   for (i = 1; i < n; i++)
   {
      j = i;
      while (j > 0 && arr[j - 1] > arr[j])
      {
         temp = arr[j];
```

```
            arr[j] = arr[j - 1];
            arr[j - 1] = temp;
            j--;
        }
    }
    printf("Sorted list in ascending order:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\n", arr[i]);
    }
end = clock();
double execution_time = (((double)(end - start))/CLOCKS_PER_SEC);
printf("Time taken is  %f",execution_time);
}
```

OUTPUT:

```
Enter the  number of elements 6
Enter 6 integers
67
89
10
34
120
12
Sorted  list in ascending order :
10
12
34
67
89
120
Time taken is 45.329670_
```

RESULT:

Thus the program was executed successfully.

4(B) HEAP SORT:

AIM:

To write program for heap sort algorithm

ALGORITHM:

STEP1: Start.

STEP2: Construct a binary tree with given list of elements.

STEP3: Delete the root element from Min Heap using heapify method.

STEP4: Put the delete element into the Sorted list.

STEP5: Repeat the same until Min heap becomes empty.

STEP6: Display the sorted List.

STEP7: Stop.

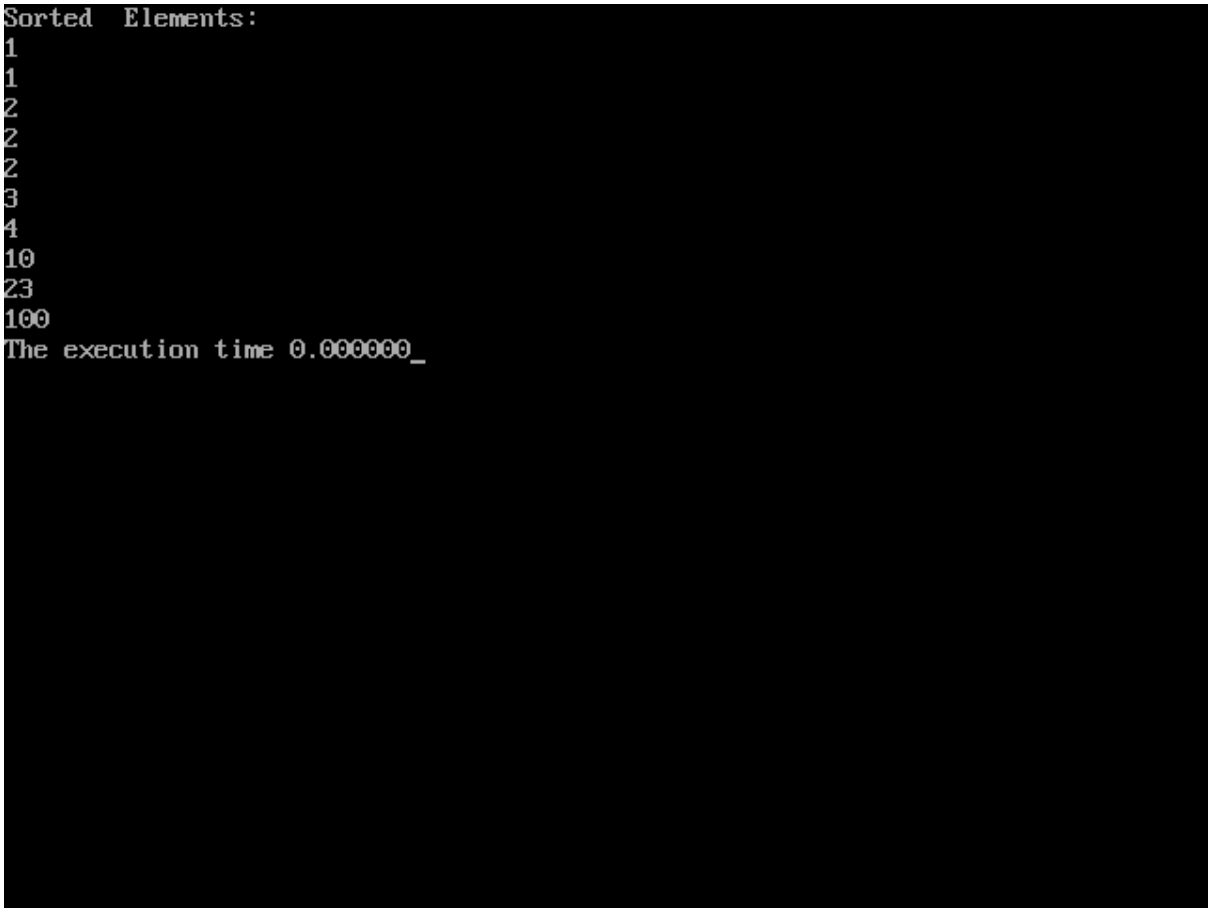
PROGRAM:

```
//HEAP SORT
#include<stdio.h>
#include<conio.h>
int temp;
void heapify (int arr[], int size, int i)
{
int largest = i;
int left = 2*i + 1;
int right = 2*i + 2;
if (left < size && arr[left] >arr[largest])
largest = left;
if (right < size && arr[right] > arr[largest])
largest = right;
if (largest != i)
```

```
{
temp = arr[i];
arr[i]= arr[largest];
arr[largest] = temp;
heapify(arr, size, largest);
}
}
void heapSort(int arr[], int size)
{
int i;
for (i = size / 2 - 1; i >= 0; i--)
heapify(arr, size, i);
for (i=size-1; i>=0; i--)
{
temp = arr[0];
arr[0]= arr[i];
arr[i] = temp;
heapify(arr, i, 0);
}
}
void main()
{
int arr[] = {1, 10, 2, 3, 4, 1, 2, 100,23, 2};
int i;
int size = sizeof(arr)/sizeof(arr[0]);
 clock_t start, end;
clrscr();
```

```
start = clock();


heapSort(arr, size);

printf("Sorted Elements:");

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

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

end = clock();

double execution_time = (((double)(end - start))/CLOCKS_PER_SEC);

printf("Time taken is  %f",execution_time);
getch();

}
```

OUTPUT:



RESULT:

Thus the program was executed successfully.

Graph Algorithms

5. Develop a program to implement graph traversal using Breadth First Search

AIM:

To write a program for graph traversal using Breadth First Search (BFS) algorithm.

ALGORITHM:

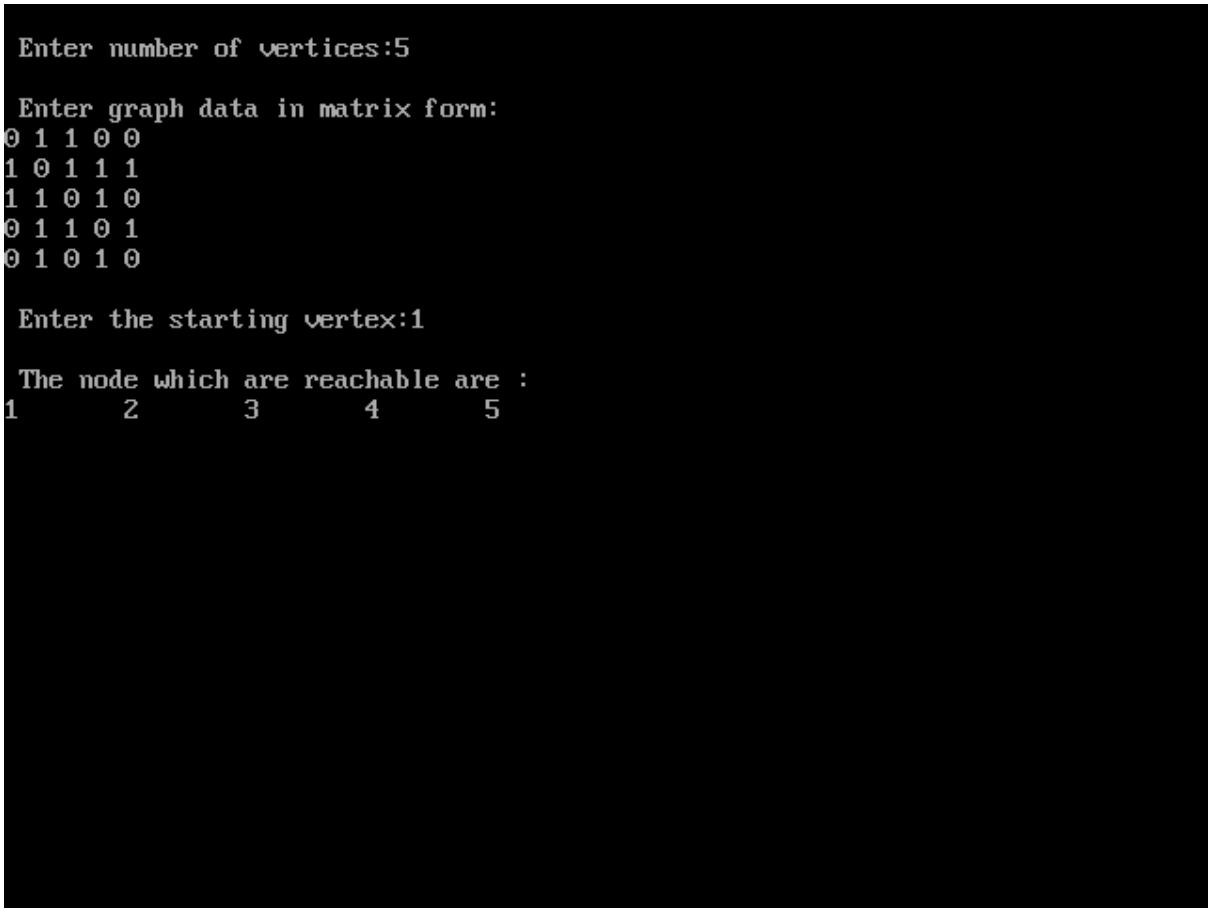
STEP1: Start.
STEP2: Choose any one node to start traversing.
STEP3: Visit its adjacent unvisited node.
STEP4: Insert the visited node into the queue.
STEP5: If there is no adjacent node,remove the first node from the queue .
STEP6: Repeat the steps3,4,5 until the queue is empty.
STEP7: Stop.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=-1,r=0;
voidbfs(int v)
{
q[++r]=v;
visited[v]=1;
while(f<=r)
{
for(i=1;i<=n;i++)
if(a[v][i] && !visited[i])
{
visited[i]=1;
q[++r]=i;
}
f++;
v=q[f];
}
}
void main()
{
int v;
clrscr();
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",q[i]);
else
printf("\n Bfs is not possible");
}
```

OUTPUT:



RESULT:

Thus the program was executed successfully.

6. Develop a program to implement graph traversal using Depth First Search

AIM:

To write a program for graph traversal using Depth First Search (DFS) algorithm.

ALGORITHM:

STEP1: Start.
STEP2: Start by declaring the structure for creating the node with vertex and edges
STEP2A: Declare function to read the graph and insert values for the graph

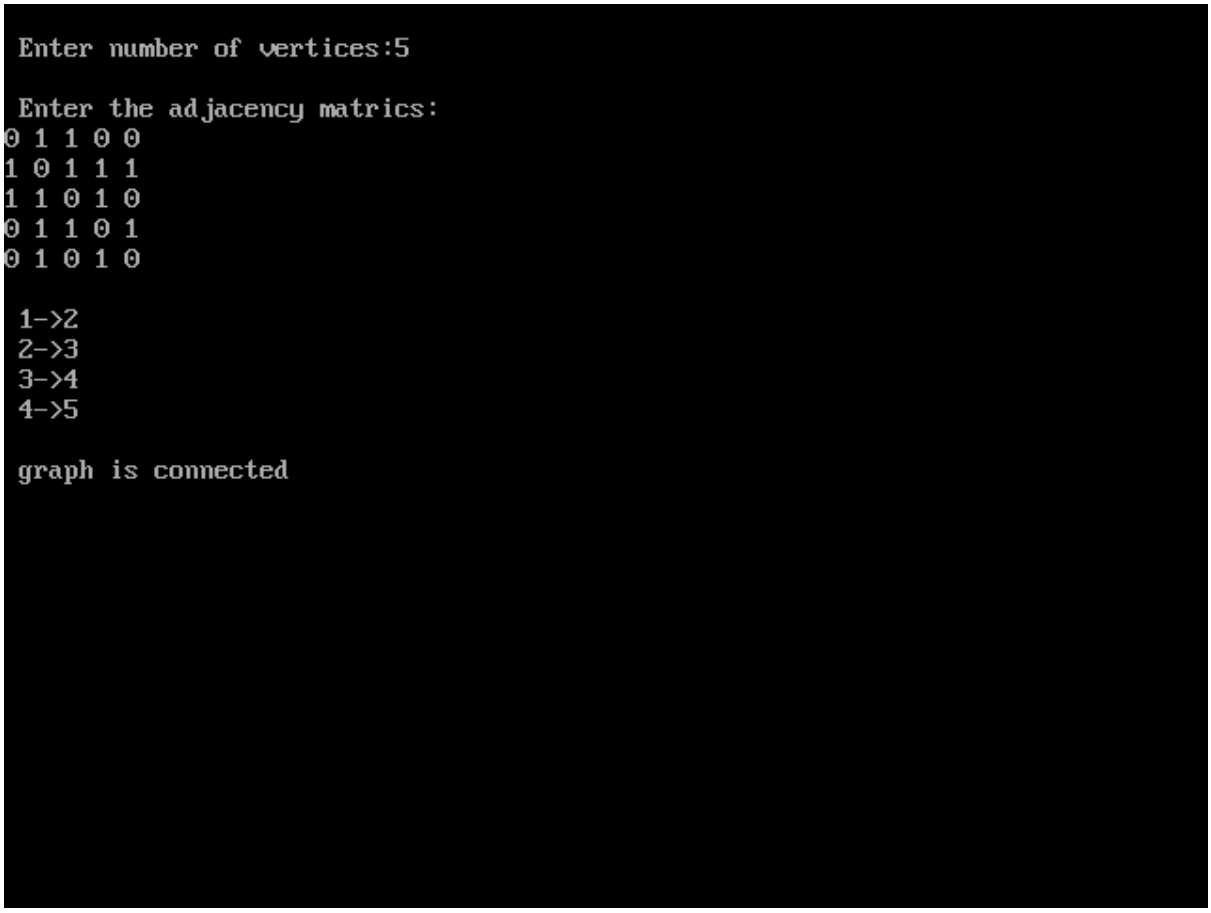
STEP2B: Traversal starts from a vertex by visiting each adjacent vertex by traversing
STEP3A: Graph can have cycles so make sure the nodes are not revisited
STEP3B: If a node is visited, mark it and store in an array and then move on to the next
STEP4: Based on the adjacency matrix, values are searched in the graph and printed
STEP7: Stop.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(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(){
int i,j,count=0;
printf("\n Enter 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("\n Enter 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("\n Graph is connected");
else
printf("\n Graph is not connected");
}
```

OUTPUT:



RESULT:

Thus the program was executed successfully.

7. From a given vertex in a weighted connected graph, develop a program to find the shortest paths to other vertices using Dijkstra's algorithm.

AIM:

To develop a program to find the shortest paths to other vertices using Dijkstra's algorithm for a given vertex in a weighted connected graph.

ALGORITHM:

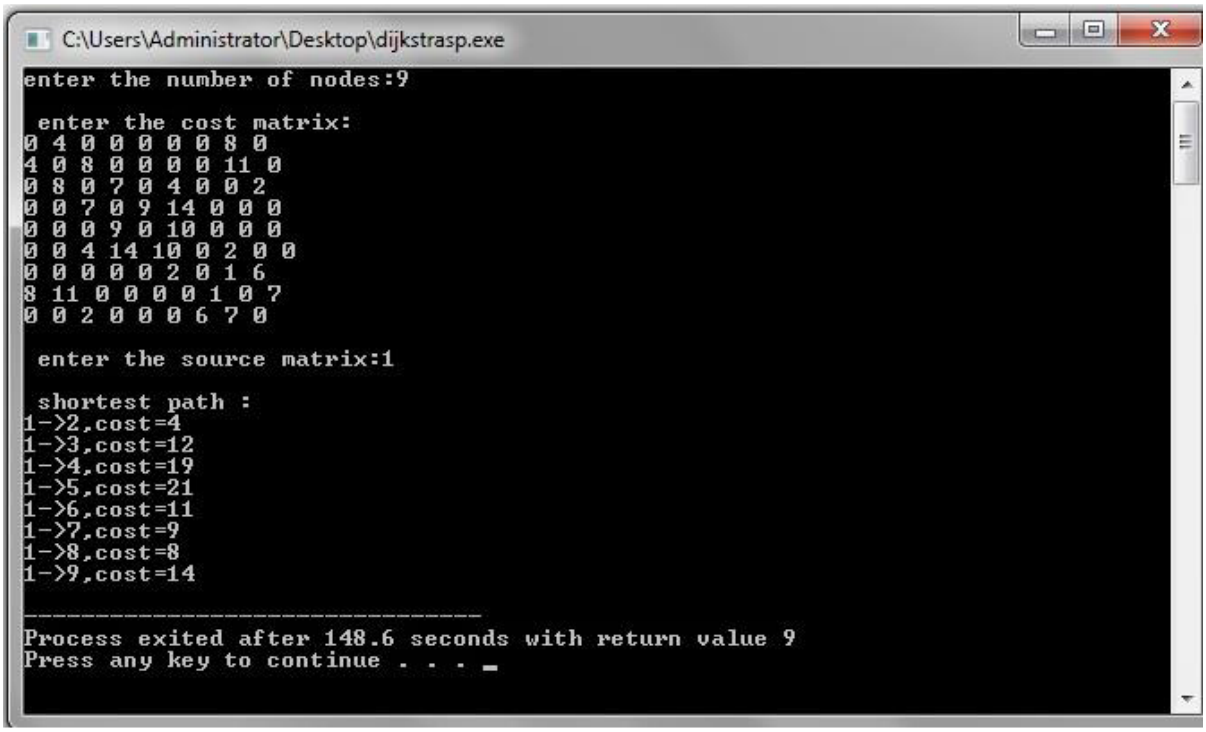
STEP1: Start
STEP2: Obtain no. of vertices and adjacency matrix for the given graph
STEP3: Create cost matrix from adjacency matrix. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity
STEP4: Initialize visited[] to zero
STEP5: Read source vertex and mark it as visited
STEP6: Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex
distance[i]=cost[0][i];
STEP7: Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.
STEP8: Recalculate the shortest distance of remaining vertices from the source.
STEP9: Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v
if(visited[v]==0)
distance[v]=min(distance[v]
distance[w]+cost[w][v])
STEP10: Stop.

PROGRAM:

```
#include<stdio.h>
#define infinity 999
void dij(int n, int v,int cost[20][20], int dist[])
{
int i,u,count,w,flag[20],min;
for(i=1;i<=n;i++)
flag[i]=0,dist[i]=cost[v][i];
count=2;
while(count<=n){
min=99;
for(w=1;w<=n;w++)
if(dist[w]<min && !flag[w]) {
min=dist[w];
u=w;
}
flag[u]=1;
count++;
for(w=1;w<=n;w++)
if((dist[u]+cost[u][w]<dist[w]) && !flag[w])
dist[w]=dist[u]+cost[u][w];
```

```
}
}
int main(){
int n,v,i,j,cost[20][20],dist[20];
printf("enter the number of nodes:");
scanf("%d",&n);
printf("\n enter the cost matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++){
scanf("%d",&cost[i][j]);
if(cost[i][j] == 0)
cost[i][j]=infinity;
}
printf("\n enter the source matrix:");
scanf("%d",&v);
dij(n,v,cost,dist);
printf("\n shortest path : \n");
for(i=1;i<=n;i++)
if(i!=v)
printf("%d->%d,cost=%d\n",v,i,dist[i]);
}
```

OUTPUT:



RESULT:

Thus the program was executed successfully.

8. Find the minimum cost spanning tree of a given undirected graph using prim's algorithm.

AIM:

To write a program to find the minimum cost spanning tree of a given undirected graph using prim's algorithm.

ALGORITHM:

STEP1: Start
STEP2: Select a starting vertex
STEP3: Select an edge connecting the tree vertex and fringe vertex that has minimum weight.
STEP4: Repeat Steps 4 and 5 until there are fringe vertices
STEP5: Add the selected edge and the vertex to the minimum spanning tree T
STEP6: [END OF LOOP]
STEP7: EXIT

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int a, b, u,v, n, i, j, ne=1;
int visited[10]={0}, min, mincost=0, cost[10][10];
void main()
{
printf("\n Enter the number of nodes:");
scanf("%d",&n);
printf("\n Enter the adjacency matrix:\n");
 for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
}
visited[1]=1;
printf("\n");
while(ne<n)
{
for(i=1,min=999;i<=n;i++)
for(j=1;j<=n;j++)
if(cost[i][j]<min)
if(visited[i]!=0)
{
min=cost[i][j];
a=u=i;
b=v=j;
}
if(visited[u]==0 || visited[v]==0)
{
```

```
printf("\n Edge %d:(%d %d) cost:%d", ne++,a, b, min);
mincost+=min;
visited[b]=1;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n Minimun cost=%d",mincost);
}
```

OUTPUT:

```
 Enter the number of nodes:5

 Enter the adjacency matrix:
999   3   2   6   999
3   999   4   999   2
2   4   999   4   999
6   999   4   999 999
999   2 999   999 999


 Edge 1:(1 3)cost=2
 Edge 2:(2 5)cost=2
 Edge 3:(1 2)cost=3
 Edge 4:(3 4)cost=4
 Minimum cost=11
```

RESULT:

Thus the program was executed successfully.

9. Implement Floyd's algorithm for the All-Pairs- Shortest-Paths problem.

AIM:

To write a program to find the all pair shortest paths problem using Floyd's algorithm

ALGORITHM:

STEP1: Start.

STEP2: Initialize the solution matrix same as the input graph matrix as a first step.

STEP3: Then we update the solution matrix by considering all vertices as an intermediate vertex.

STEP4: The ideas is to one by one pick all vertices and update all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.

STEP5: When we pick vertex number k as an intermediate vertex, we already have considered vertices

STEP6:{0, 1, 2, .. k-1} as intermediate vertices.

STEP7: For every pair (i, j) of source and destination vertices respectively, there are two possible cases.

STEP8:k is not an intermediate vertex in shortest path from i to j. Keep the value of dist[i][j] as it is.

STEP9:k is an intermediate vertex in shortest path from i to j. Update the value of dist[i][j] as dist[i][k] + dist[k][j].

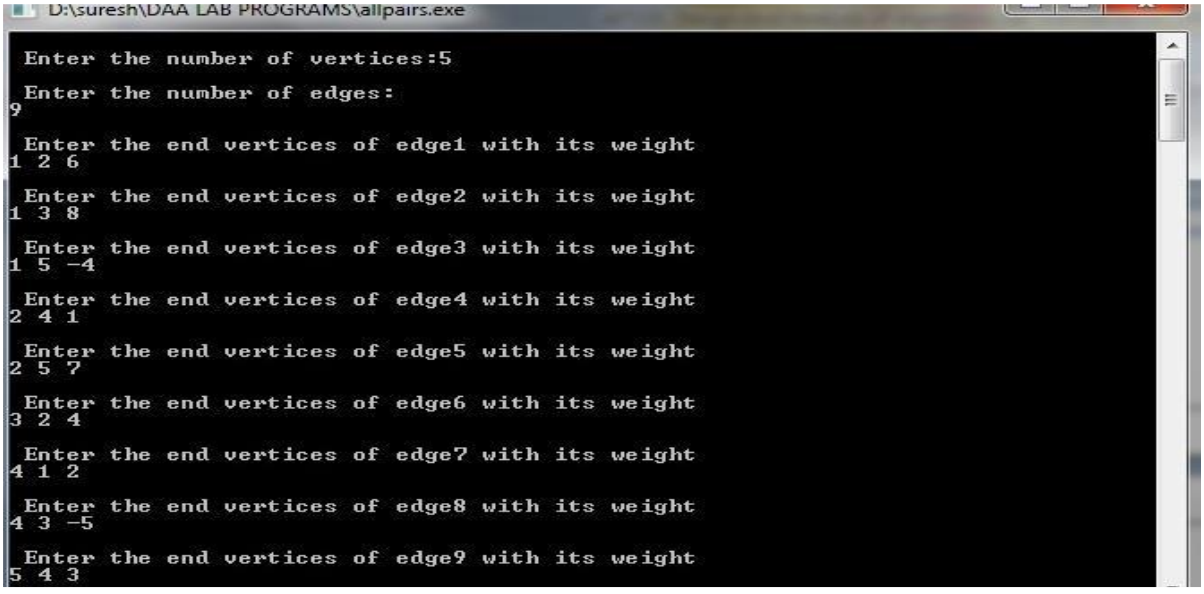
STEP10: Stop.

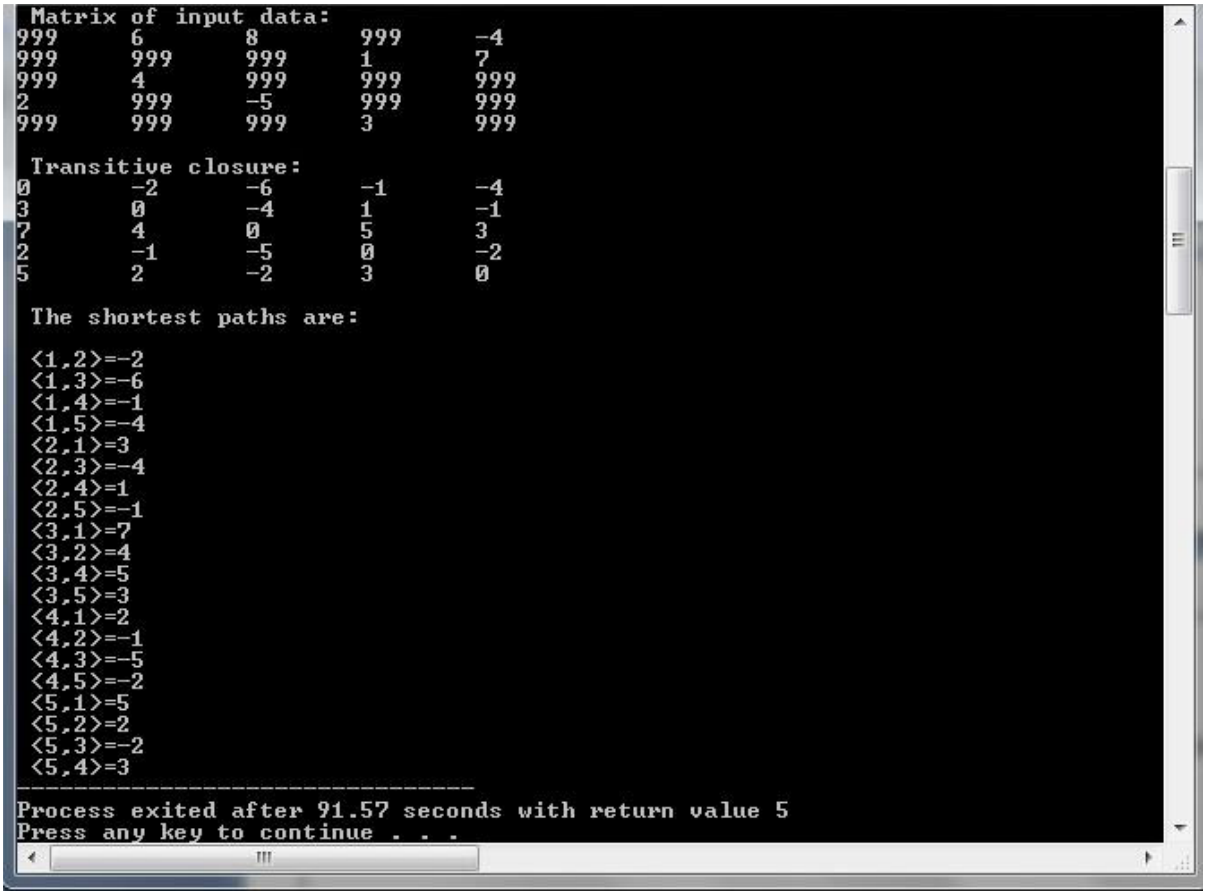
PROGRAM:

```
#include<stdio.h>
int min(int,int);
voidfloyds(int p[10][10],int n){
inti,j,k;
for(k=1;k<=n;k++)
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(i==j)
p[i][j]=0;
else
p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
int min(inta,int b){
if(a<b)
return(a);
else
return(b);
}
main(){
int p[10][10],w,n,e,u,v,i,j;
printf("\n Enter the number of vertices:");
scanf("%d",&n);
printf("\n Enter the number of edges:\n");
scanf("%d",&e);
for(i=1;i<=n;i++){
```

```
for(j=1;j<=n;j++)
p[i][j]=999;
}
for(i=1;i<=e;i++){
printf("\n Enter the end vertices of edge%d with its weight \n",i);
scanf("%d%d%d",&u,&v,&w);
p[u][v]=w;
}
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");
}
floyds(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");
}
printf("\n The shortest paths are:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++){

if(i!=j)
printf("\n <%d,%d>=%d",i,j,p[i][j]);
}
}
```

OUTPUT:





RESULT:

Thus the program was executed successfully.

10. Compute the transitive closure of a given directed graph using Warshall's algorithm.

AIM:

To write a program to compute the transitive closure of a given directed graph using Warshall's algorithm.

ALGORITHM:

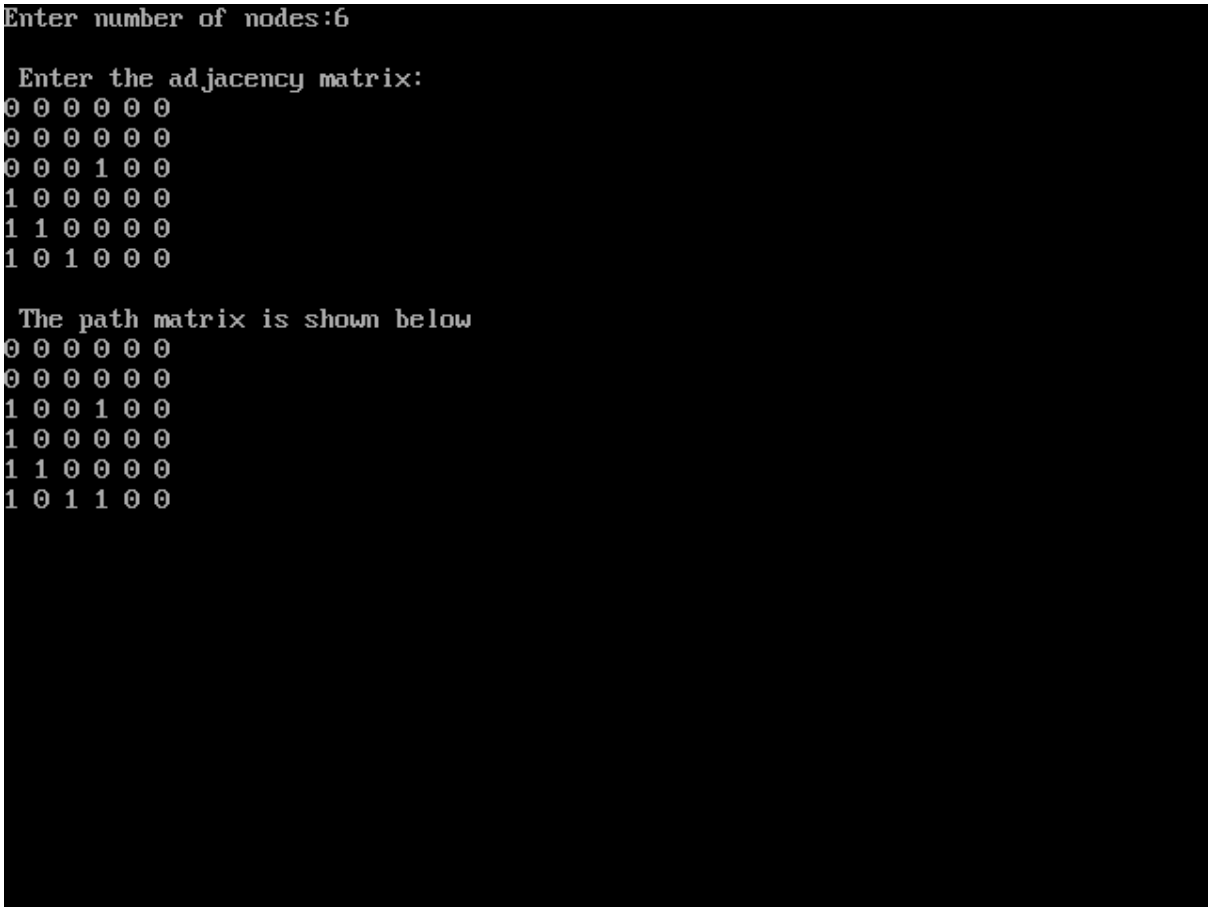
Transitive closure

1. Given a directed graph, find out if a vertex j is reachable from another vertex i for all vertex pairs (i, j) in the given graph.
2. Here reachable mean that there is a path from vertex i to j.
3. The reach-ability matrix is called transitive closure of a graph.

PROGRAM:

```
//Transitive closure of a graph using Warshall's algorithm
#include <stdio.h>
intn,a[10][10],p[10][10];
void path(){
inti,j,k;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
p[i][j]=a[i][j];
for(k=0;k<n;k++)
for(i=0;i<n;i++)

for(j=0;j<n;j++)
if(p[i][k]==1&&p[k][j]==1)
p[i][j]=1;
}
void main(){
inti,j;
printf("Enter the number of nodes:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
path();
printf("\nThe path matrix is shown below\n");
for(i=0;i<n;i++){
for(j=0;j<n;j++)
printf("%d ",p[i][j]);
printf("\n");
}
}
```

OUTPUT:



RESULT:

Thus the program was executed successfully.

Algorithm Design Techniques

11. Develop a program to find out the maximum and minimum numbers in a given list of *n* numbers using the divide and conquer technique.

AIM:

To write a program to find the maximum and minimum number in a given list of n numbers using the divide and conquer technique.

ALGORITHM:

STEP1: Start.

STEP2: The main() function calls the minimum() by passing array,array size,1 as arguments.

STEP3: Then the function minimum()

a) Checks the condition i<n, If it is true

b) Then it compares a[min]>a[i] if it is also true

c) Then min initialised to i and calls the function by itself by increasing i value until the condition a[min]>a[i] becomes false.

d)This function returns the min to the main function main() prints the a[min] value of the array.

STEP4: The main() function calls the maximum() function by passing array,array size,1 as arguments.

STEP5: Then the function maximum()

a) Checks the condition i<n, if it is true

b) Then it compares a[max]<a[i] if it is true

c) Then max initialise to i and calls the function itself by increasing i value until the condition a[max]<a[i] becomes false.

d)This function returns the max to the main function.

e)main() function prints the a[max] value of the array.

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
int minimum(int a[],int n,int i)
{
static int min=0;;
if(i<n)
{
if(a[min]>a[i])
 {
min=i;
 minimum(a,n,++i);
}
}
return min;
}
int maximum(int a[],int n,int i)
{
 static int max=0;;
if(i<n)
 {
 if(a[max]<a[i])
 {
```

```
 max=i;
 maximum(a,n,++i);
 }
 }
 return max;
}
int main()
{
 int a[1000],i,n,sum;

 printf("Enter size of the array \n : ");
 scanf("%d", &n);
 printf("Enter elements in array \n: ");
 for(i=0; i<n; i++)
 {
 scanf("%d", &a[i]);
 }
 printf("minimum of array is : %d",a[(minimum(a,n,1))]);
 printf("\nmaximum of array is : %d",a[(maximum(a,n,1))]);
}
```

OUTPUT:

RESULT:

Thus the program was executed successfully.

12. Implement Merge sort and Quick sort methods to sort an array of elements and determine the time required to sort. Repeat the experiment for different values of *n*, the number of elements in the list to be sorted and plot a graph of the time taken versus *n*.

AIM:

To sort the elements in the given array using Quick sort methods

ALGORITHM:

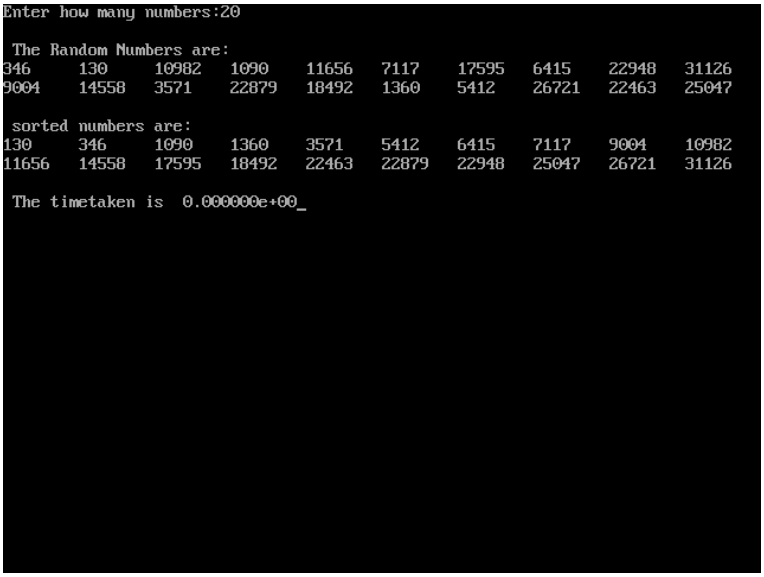
STEP1: Define the function for performing quick sort
STEP2A: Allocate a pivot element in the list and compare the first and last elements
STEP2B: If the elements are not in order, swap them and order them in list
STEP3: This is done for all the elements stored in the list
STE4A: Declare the number of elements and count them
STEP4B: Print the ordered/sorted list of elements
STEP5: Stop

PROGRAM:

```
include <stdio.h>
include <time.h>
voidExch(int *p, int *q){
int temp = *p;
*p = *q;
*q = temp;
}
voidQuickSort(int a[], int low, int high){
int i, j, key, k;
if(low>=high)
return;
key=low;

i=low+1;
j=high;
while(i<=j){
while ( a[i] <= a[key] )
i=i+1;
while ( a[j] > a[key] )
j=j -1;
if(i<j)
Exch(&a[i], &a[j]);
}
Exch(&a[j], &a[key]);
QuickSort(a, low, j-1);
QuickSort(a, j+1, high);
}
void main(){
int n, a[1000],k;
clock_tst,et; double ts; clrscr();
printf("\n Enter How many Numbers: ");
scanf("%d", &n);
printf("\nThe Random Numbers are:\n");
for(k=1; k<=n; k++){
```

```
a[k]=rand();
printf("%d\t",a[k]);
}
st=clock();
QuickSort(a, 1, n);
et=clock();
ts=(double)(et-st)/CLOCKS _PER_SEC;
printf("\nSorted Numbers are: \n ");
for(k=1; k<=n; k++)
printf("%d\t", a[k]);
printf("\nThe time taken is %e",ts);
}
```

OUTPUT:



RESULT:

Thus the program was executed successfully.

12.b. Merge sort

AIM:

To sort the elements in the given array using Merge sort methods

ALGORITHM:

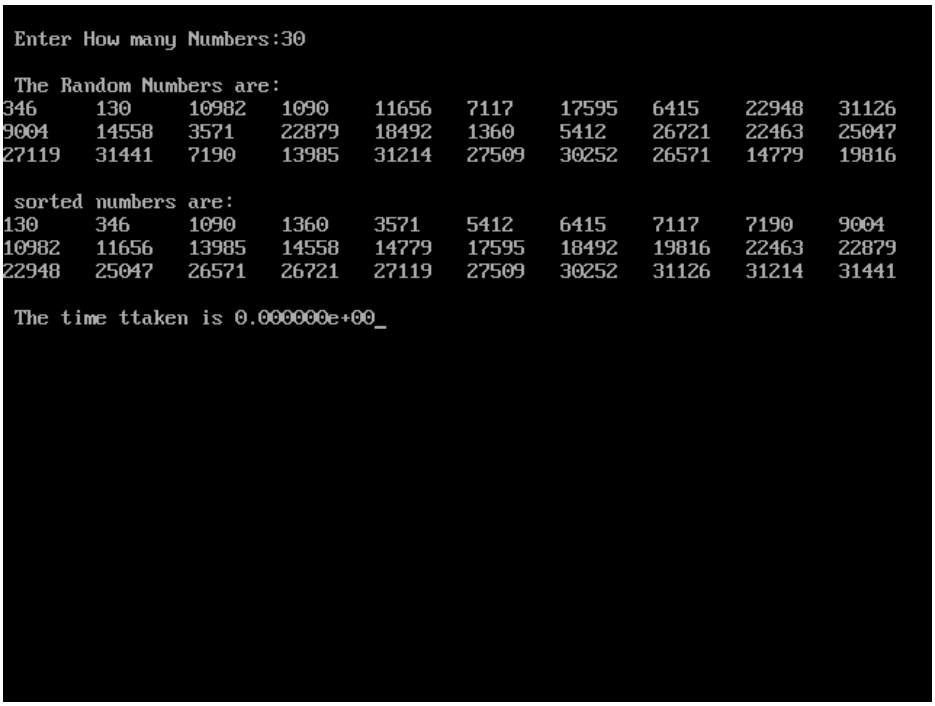
STEP1: Define the function for performing merge sort
STEP2: Obtain the number of elements for list1 and list2
STEP3: Enter the ordered elements for list 1 and list 2
STEP4: Compare the elements of both lists and merge them
STEP5: Then perform sorting of two lists after merging them together as one list
STEP6: Print the combined or merged and ordered/sorted list of elements
STEP7: Stop

PROGRAM:

```
#include <stdio.h>
#include<time.h>
int b[50000];
void Merge(int a[], int low, int mid, int high){
int i, j, k;
i=low; j=mid+1; k=low;
while ( i<=mid && j<=high ) {
if( a[i] <= a[j] )
b[k++] = a[i++] ;
else
b[k++] = a[j++] ;
}
while (i<=mid)
b[k++] = a[i++] ;
while (j<=high)
b[k++] = a[j++] ;
for(k=low; k<=high; k++)
a[k] = b[k];
}

voidMergeSort(int a[], int low, int high){
int mid;
if(low >= high)
return;
mid = (low+high)/2 ;
MergeSort(a, low, mid);
MergeSort(a, mid+1, high);
Merge(a, low, mid, high);
}
void main(){
int n, a[50000],k;
clock_tst,et;
```

```
doublets;
printf("\n Enter How many Numbers:");
scanf("%d", &n);
printf("\nThe Random Numbers are:\n");
for(k=1; k<=n; k++) {
a[k]=rand();
printf("%d\t", a[k]);
}
st=clock();
MergeSort(a, 1, n);
et=clock();
ts=(double)(et-st)/CLOCKS_PER_SEC;
printf("\n Sorted Numbers are : \n ");
for(k=1; k<=n; k++)
printf("%d\t", a[k]);
printf("\nThe time taken is %e",ts);
}
```

OUTPUT:



RESULT:

Thus the program was executed successfully.

State Space Search Algorithms

13.Implement N Queens problem using Backtracking.

AIM:

To implement the program for N-Queen problem using Backtracking.

ALGORITHM:

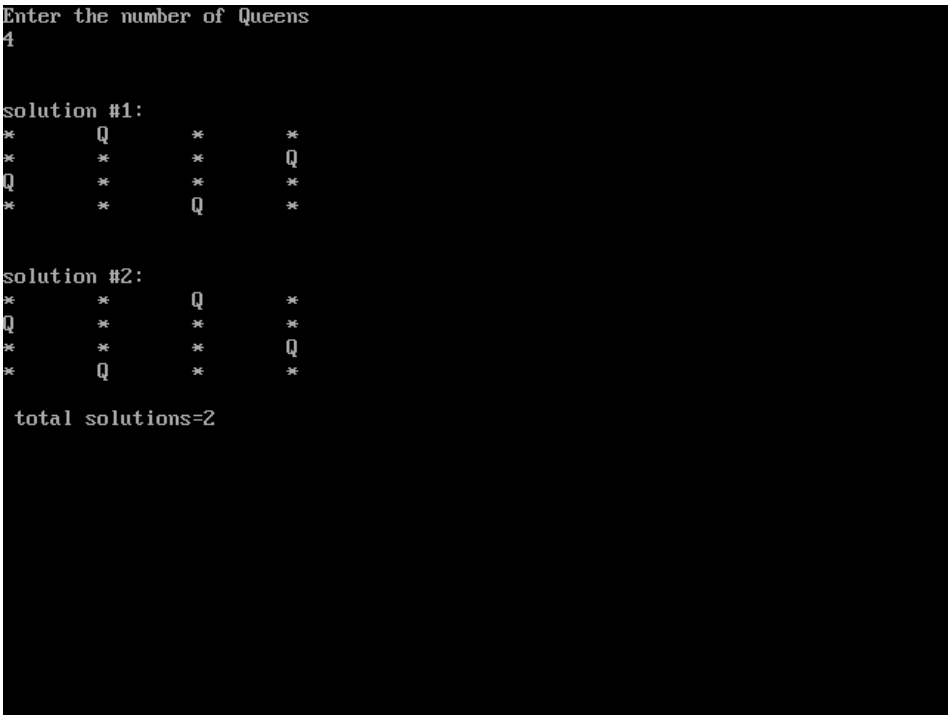
1) Start in the leftmost column
2) If all queens are placedreturn true
3) Try all rows in the current column. Do following for every tried row.
a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
b) If placing queen in [row, column] leads to a solution then return true.
c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go
to step (a) to try other rows.
3) If all rows have been tried and nothing worked, return false to trigger Backtracking.

PROGRAM:

```
#include<stdio.h>
#include<math.h>
int a[30],count=0;
int place(intpos){
int i;
for(i=1;i<pos;i++){
if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
return 0;
}
return 1;
}
voidprint_sol(int n){
inti,j; count++;
printf("\n\nSolution #%d:\n",count);
for(i=1;i<=n;i++){
for(j=1;j<=n;j++){
if(a[i]==j)
printf("Q\t");
else
printf("*\t");
}
printf("\n");
}
}
void queen(int n){
int k=1;
a[k]=0;
while(k!=0){
a[k]=a[k]+1;
while((a[k]<=n)&&!place(k))
```

```
a[k]++;
if(a[k]<=n){
if(k==n)
print_sol(n);
else{
k++;
a[k]=0;
}
}
else
k--;
}
}
void main(){
inti,n;
printf("Enter the number of Queens\n");
scanf("%d",&n);
queen(n);
printf("\nTotal solutions=%d",count);
}
```

OUTPUT:



RESULT:

Thus the program was executed successfully.

Approximation Algorithms Randomized Algorithms

14.Implement any scheme to find the optimal solution for the Travelling Salesperson problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.

AIM:

To implement the program for Travelling Salesperson problem using Backtracking.

ALGORITHM:

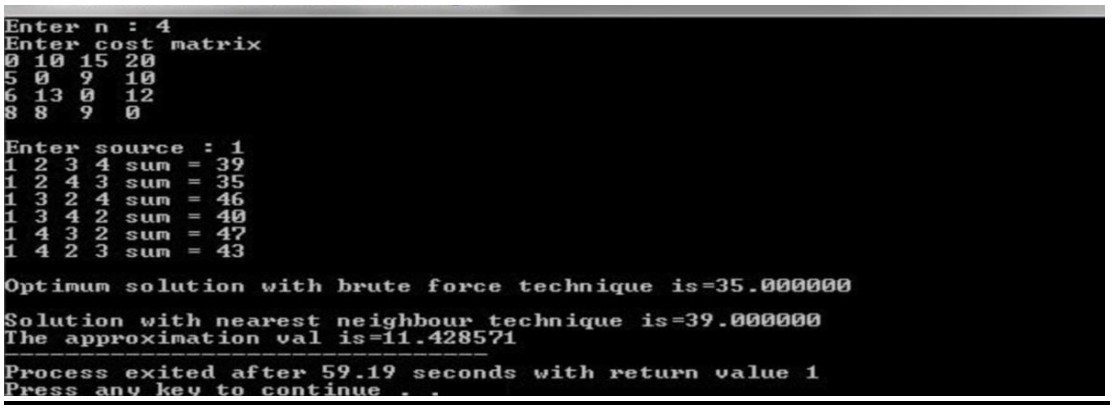
1. Check for the disconnection between the current city and the next city
2. Check whether the travelling sales person has visited all the cities
3. Find the next city to be visited
4. Find the solution and terminate

PROGRAM:

```
#include<stdio.h>
ints,c[100][100],ver;
float optimum=999,sum;
/* function to swap array elements */
void swap(int v[], int i, int j) {
int t;
t = v[i];
v[i] = v[j];
v[j] = t;
}
/* recursive function to generate permutations */
voidbrute_force(int v[], int n, int i) {
// this function generates the permutations of the array from element i to element n-1
int j,sum1,k;
//if we are at the end of the array, we have one permutation
if (i == n) {
if(v[0]==s) {
for (j=0; j<n; j++)
printf ("%d ", v[j]);
sum1=0;
for( k=0;k<n-1;k++) {
sum1=sum1+c[v[k]][v[k+1]];
}
sum1=sum1+c[v[n-1]][s];
printf("sum = %d\n",sum1);
if (sum1<optimum)
optimum=sum1;
}
}
else
// recursively explore the permutations starting at index i going through index n-1*/
for (j=i; j<n; j++) { /* try the array with i and j switched */
swap (v, i, j);
```

```
brute_force (v, n, i+1);
/* swap them back the way they were */
swap (v, i, j);
}
}
voidnearest_neighbour(intver) {
intmin,p,i,j,vis[20],from;
for(i=1;i<=ver;i++)
vis[i]=0;
vis[s]=1;
from=s;
sum=0;
for(j=1;j<ver;j++) {
min=999;
for(i=1;i<=ver;i++)
if(vis[i] !=1 &&c[from][i]<min && c[from][i] !=0 ) {
min= c[from][i];
p=i;
}
vis[p]=1;
from=p;
sum=sum+min;
}
sum=sum+c[from][s];
}
void main () {
intver,v[100],i,j;
printf("Enter n : ");
scanf("%d",&ver);
for (i=0; i<ver; i++)
v[i] = i+1;
printf("Enter cost matrix\n");
for(i=1;i<=ver;i++)
for(j=1;j<=ver;j++)
scanf("%d",&c[i][j]);
printf("\nEnter source : ");
scanf("%d",&s);
brute_force (v, ver, 0);
printf("\nOptimum solution with brute force technique is=%f\n",optimum);
nearest_neighbour(ver);
printf("\nSolution with nearest neighbour technique is=%f\n",sum);
printf("The approximation val is=%f",((sum/optimum)-1)*100);
printf(" % ");
}
```

OUTPUT:



RESULT:

Thus the program was executed successfully.

15. Implement randomized algorithms for finding the kth smallest number.

ALGORITHM:

Given an array A[] of n elements and a positive integer K, find the
Kth smallest element in the array. It is given that all array elements are distinct.
Brute force and efficient solutions
We will be discussing four possible solutions for this problem:-
1. Brute Force approach: Using sorting
2. Using Min-Heap
3. Using Max-Heap
4. Quick select: Approach similar to quick sort
Brute force approach: Using sorting
The idea is to sort the array to arrange the numbers in increasing order and then returning the Kth number from the start.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>

// Compare function for qsort
int cmpfunc(const void* a, const void* b)
{
 return (*(int*)a - *(int*)b);
}

// Function to return k'th smallest element in a given array
int kthSmallest(int arr[], int n, int k)
{
 // Sort the given array
 qsort(arr, n, sizeof(int), cmpfunc);

 // Return k'th element in the sorted array
 return arr[k - 1];
}

// Driver program to test above methods
int main()
{
 int arr[] = { 12, 3, 5, 7, 19 };
 int n = sizeof(arr) / sizeof(arr[0]), k = 2;
 printf("K'th smallest element is %d",
 kthSmallest(arr, n, k));
 return 0;
}
```

OUTPUT:

RESULT:

Thus the program was executed successfully.