SCAD COLLEGE OF ENGINEERING & TECHNOLOGY
Cheranmahadevi, Tirunelveli

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

ACADEMIC YEAR 2023-2024

(REGULATION-2021)

PROGRAMME: UG

SEMESTER: IV

COURSE CODE: CS3401

COURSE NAME: ALGORITHM LABORATORY

BY,

Mrs.K.ARUN PRIYA M.E,

(AP/CSE)

LIST OF EXPERIMENTS

PRACTICAL EXERCISES **PERIODS:30**

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 agraph of the time taken versus *n*.
2. Implement recursive Binary Search. Determine the time required to search an element. Repeatthe 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

1. Develop a program to implement graph traversal using Breadth First Search
2. Develop a program to implement graph traversal using Depth First Search
3. From a given vertex in a weighted connected graph, develop a program to find theshortest pathsto other vertices using Dijkstra’s algorithm.
4. Find the minimum cost spanning tree of a given undirected graph using Prim’salgorithm.
5. Implement Floyd’s algorithm for the All-Pairs- Shortest-Paths problem.
6. Compute the transitive closure of a given directed graph using Warshall's algorithm.

Algorithm Design Techniques

1. Develop a program to find out the maximum and minimum numbers in a given listof *n* numbersusing the divide and conquer technique.

State Space Search Algorithms

1. Implement N Queens problem using Backtracking.

Approximation Algorithms Randomized Algorithms

1. 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.
2. Implement randomized algorithms for finding the kth smallest number.

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*.

Algorithm:
Linear Search is the simplest searching algorithm.
It traverses the array sequentially to locate the required element.
It searches for an element by comparing it with each element of the array one by one.
So, it is also called as Sequential Search.
There is a linear array 'a' of size 'n'.
Linear search algorithm is being used to search an element 'item' in this linear array.
If search ends in success, it sets loc to the index of the element otherwise it sets loc to -1.
Program:

```
#include <stdio.h>
int main()
{
 int array[100], search, c, n;
 printf("Enter the number of elements in array\n");
 scanf("%d",&n);
 printf("Enter %d integer(s)\n", n);
for (c = 0; c < n; c++)
 scanf("%d", &array[c]);
 printf("Enter the number to search\n");
 scanf("%d", &search);
 for (c = 0; c < n; c++)
 {
 if (array[c] == search) /* if required element found */
 {
 printf("%d is present at location %d.\n", search, c+1);
 break;
 }
 }
 if (c == n)
 printf("%d is not present in array.\n", search);
 return 0;
}
```

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*.

Algorithm:

There is a linear array ‘a’ of size ‘n’. Binary search algorithm is being used to search an element ‘item’ in this linear array. If search ends in success, it sets loc to the index of the element otherwise it sets loc to -1. Variables beg and end keeps track of the index of the first and last element of the array or sub array in which the element is being searched at that instant. Variable mid keeps track of the index of the middle element of that array or sub array in which the element is being searched at that instant.

Program:

```
#include <stdio.h>
int main()
{
 int c, first, last, middle, n, search, array[100];
 printf("Enter number of elements\n");
 scanf("%d",&n);
 printf("Enter %d integers\n", n);
 for (c = 0; c < n; c++)
 scanf("%d",&array[c]);
 printf("Enter value to find\n");
 scanf("%d", &search);
 first = 0;
 last = n - 1;
 middle = (first+last)/2;
 while (first <= last) {
 if (array[middle] < search)
 first = middle + 1;
 else if (array[middle] == search) {
 printf("%d found at location %d.\n", search, middle+1);
 break;
 }
 else
 last = middle - 1;
 middle = (first + last)/2;
 }
 if (first > last)
 printf("Not found! %d is not present in the list.\n", search);
 return 0;
}
```

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.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
void search(char* pat, char* txt)
{
     int M = strlen(pat);
```

```
    int N = strlen(txt);
    for (int 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:

Input:

```
 txt[] = "THIS IS A TESTTEXT"
  pat[] = "TEST"
```

Output:

```
Pattern found at index 10
```

Input:

```
 txt[]="AABAACAADAABAABA"
 pat[]="AABA"
```

Output:

```
Pattern found at index 0
Pattern found at index 9
```

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*.

Algorithm:

//INSERTION SORT

It is an sorting algorithm in which first element in the array is assumed as sorted, even if it is an unsorted . Then, each element in the array is checked with the previous elements, resulting in a growing sorted output list. With each iteration, the sorting algorithm removes one element at a time and finds the appropriate location within the sorted array and inserts it there. The iteration continues until the whole list is sorted

// HEAP SORT

Heap sort processes the elements by creating the min heap or max heap using the elements of the given array. Min heap or max heap represents the ordering of the array in which root element represents the minimum or maximum element of the array. At each step, the root element of the heap gets deleted and stored into the sorted array and the heap will again be heapified.

Program:

// INSERTION SORT

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i, j,n,key,arr[20];
printf("Enter number of elements to use:");
scanf("%d",&n);
printf("Enter %d elements: ",n);

// This loop would store the input numbers in array
for(i=0;i<n;i++)
scanf("%d",&arr[i]);
// Implementation of insertion sort algorithm
for(i=1;i<n;i++){
key=arr[i];
j=i-1;
while((key<arr[j])&&(j>=0)){
arr[j+1]=arr[j];
j=j-1;
}
arr[j+1]=key;
}
printf("Sorted elements: ");
for(i=0;i<n;i++)
printf(" %d",arr[i]);
getch();
}
```

//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()
{
clrscr();
int arr[] = {1, 10, 2, 3, 4, 1, 2, 100,23, 2};
int i;
int size = sizeof(arr)/sizeof(arr[0]);
heapSort(arr, size);
printf("Sorted Elements:");
for (i=0; i<size; ++i)
printf(" %d",arr[i]);
getch();
}
```

Graph Algorithms

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

ALGORITHM:

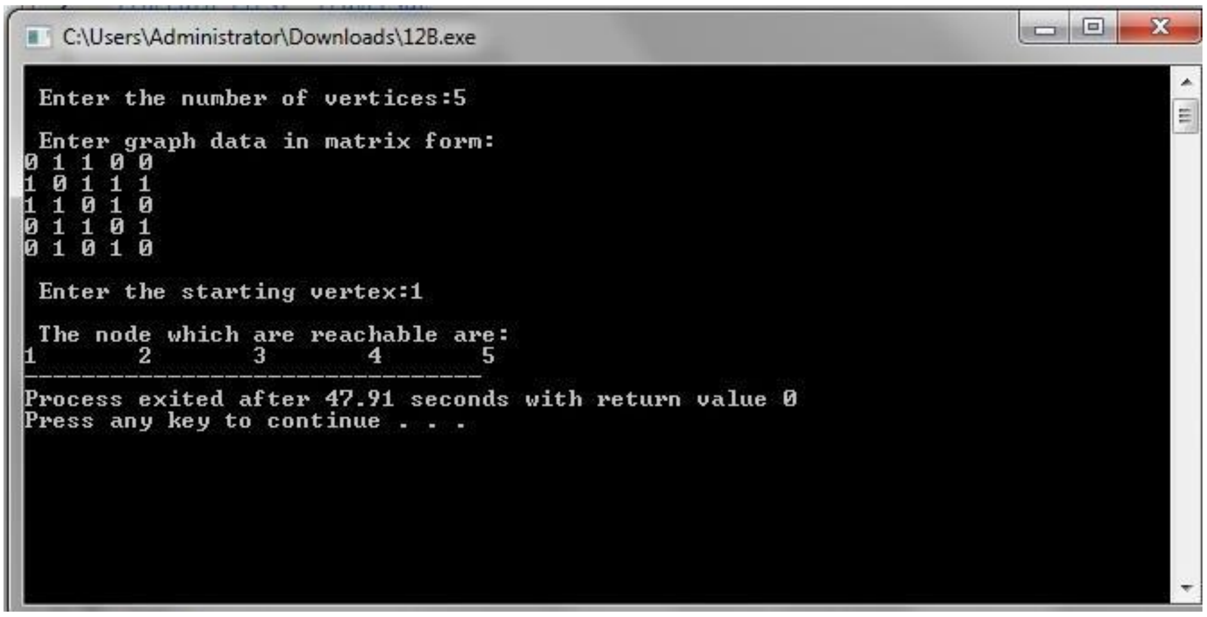
Breadth First Search (BFS) algorithm traverses a graph in a breadth ward motion and uses a queue to remember to get the next vertex to start a search.

1. Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
2. If no adjacent vertex is found, remove the first vertex from the queue.
3. Repeat Rule 1 and Rule 2 until the queue is empty.

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;
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:



```
 Enter the number of vertices:5

 Enter graph data in matrix form:
0 1 1 0 0
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
0 1 0 1 0

 Enter the starting vertex:1

 The node which are reachable are:
1       2       3       4       5
--------------------------------
Process exited after 47.91 seconds with return value 0
Press any key to continue . . .
```

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

ALGORITHM:

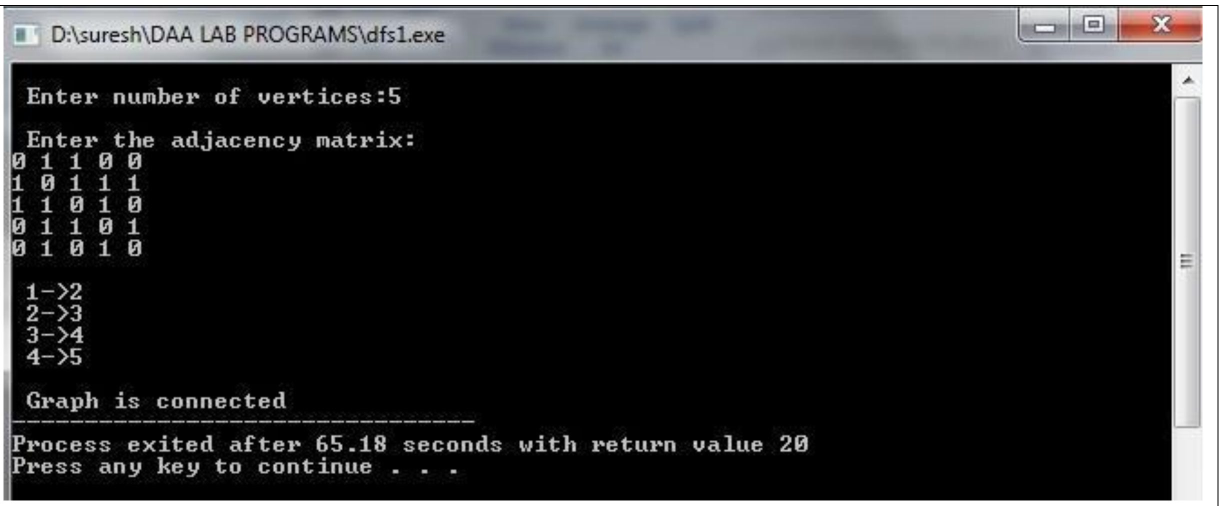
Depth First Search (DFS) algorithm traverses a graph in a depth ward motion and uses a stack to remember to get the next vertex to start a search.

1. Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
2. If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
3. Repeat Rule 1 and Rule 2 until the stack is empty.

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:


D:\suresh\DAA LAB PROGRAMS\dfs1.exe

```
 Enter number of vertices:5

 Enter the adjacency matrix:
0 1 1 0 0
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
0 1 0 1 0

 1->2
 2->3
 3->4
 4->5

 Graph is connected
--------------------------------
Process exited after 65.18 seconds with return value 20
Press any key to continue . . .
```

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

ALGORITHM

1) Create a set S that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty. **2)** Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE.Assign distance value as 0 for the source vertex so that it is picked first. **3)** While *S* doesn't include all vertices

a) Pick a vertex u which is not there in *S* and has minimum distance value. **b)**Include u to *S*.

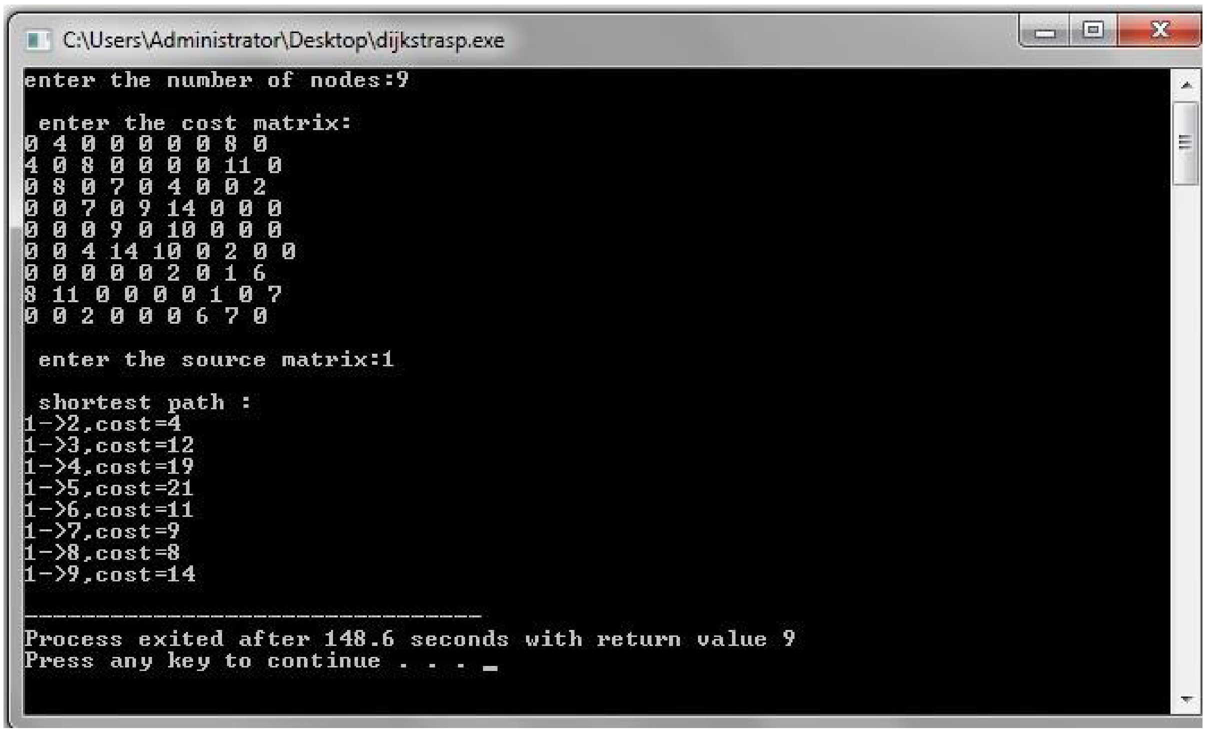
c) Update distance value of all adjacent vertices of u.

To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

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:



C:\Users\Administrator\Desktop\dijkstrasp.exe

```
enter the number of nodes:9

 enter the cost matrix:
0 4 0 0 0 0 0 8 0
4 0 8 0 0 0 0 11 0
0 8 0 7 0 4 0 0 2
0 0 7 0 9 14 0 0 0
0 0 0 9 0 10 0 0 0
0 0 4 14 10 0 2 0 0
0 0 0 0 0 2 0 1 6
8 11 0 0 0 0 1 0 7
0 0 2 0 0 0 6 7 0

 enter the source matrix:1

 shortest path :
1->2,cost=4
1->3,cost=12
1->4,cost=19
1->5,cost=21
1->6,cost=11
1->7,cost=9
1->8,cost=8
1->9,cost=14

--------------------------------
Process exited after 148.6 seconds with return value 9
Press any key to continue . . . _
```

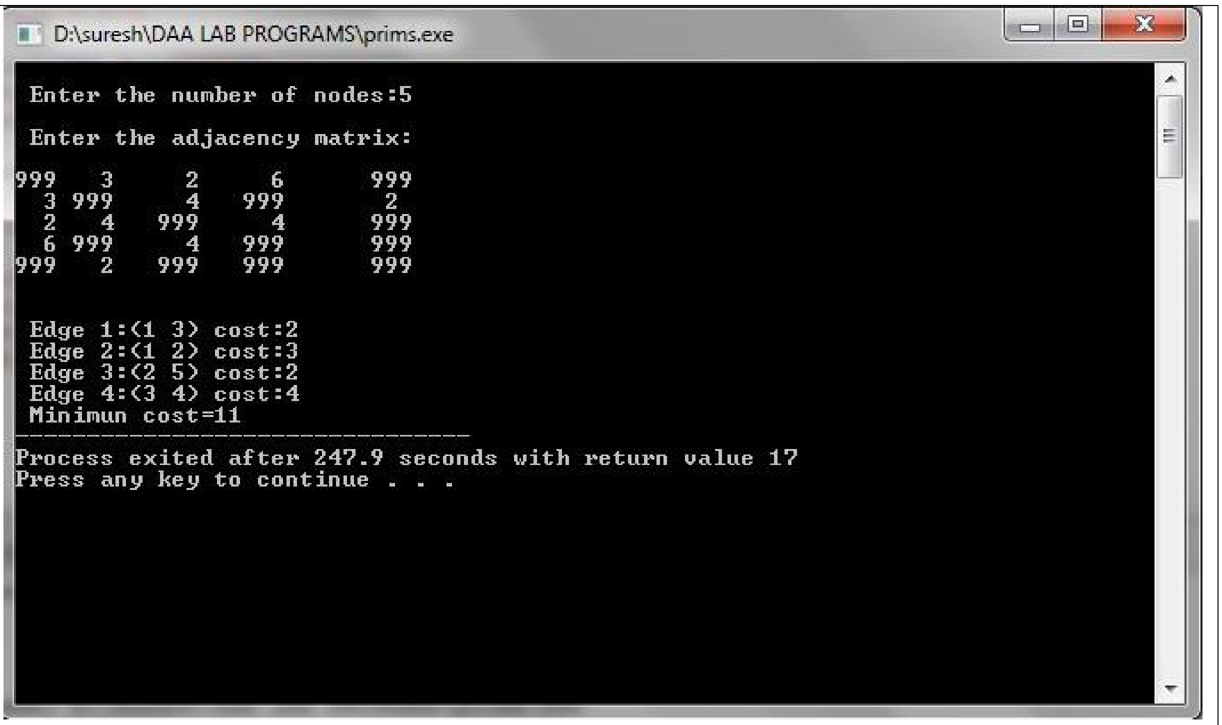
4. Find the minimum cost spanning tree of a given undirected graph using Prim’s algorithm.

ALGORITHM:

1) Create a set Sthat keeps track of vertices already included in MST. **2)** Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first. **3)** While S doesn't include all vertices.
a) Pick a vertex *u* which is not there in Sand has minimum key value. **b)** Include *u* to S.
c) Update key value of all adjacent vertices of u.
To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u-v* is less than the previous key value of *v*, update the key value as weight of *u-v*
The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.
PROGRAM

```
#include<stdio.h>
inta,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:(1 2) cost:3
 Edge 3:(2 5) cost:2
 Edge 4:(3 4) cost:4
 Minimun cost=11
--------------------------------
Process exited after 247.9 seconds with return value 17
Press any key to continue . . .
```

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

ALGORITHM:

Initialize the solution matrix same as the input graph matrix as a first step. Then we update the solution matrix by considering all vertices as an intermediate vertex. 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.

When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices.

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

1)k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.

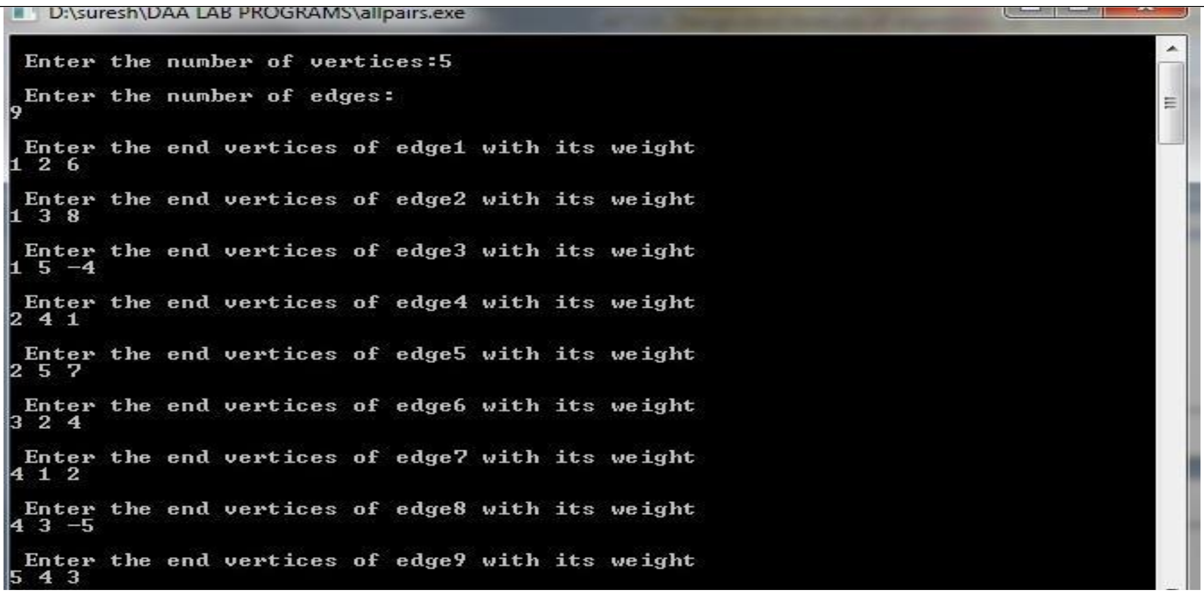
2)k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j].

PROGRAM:

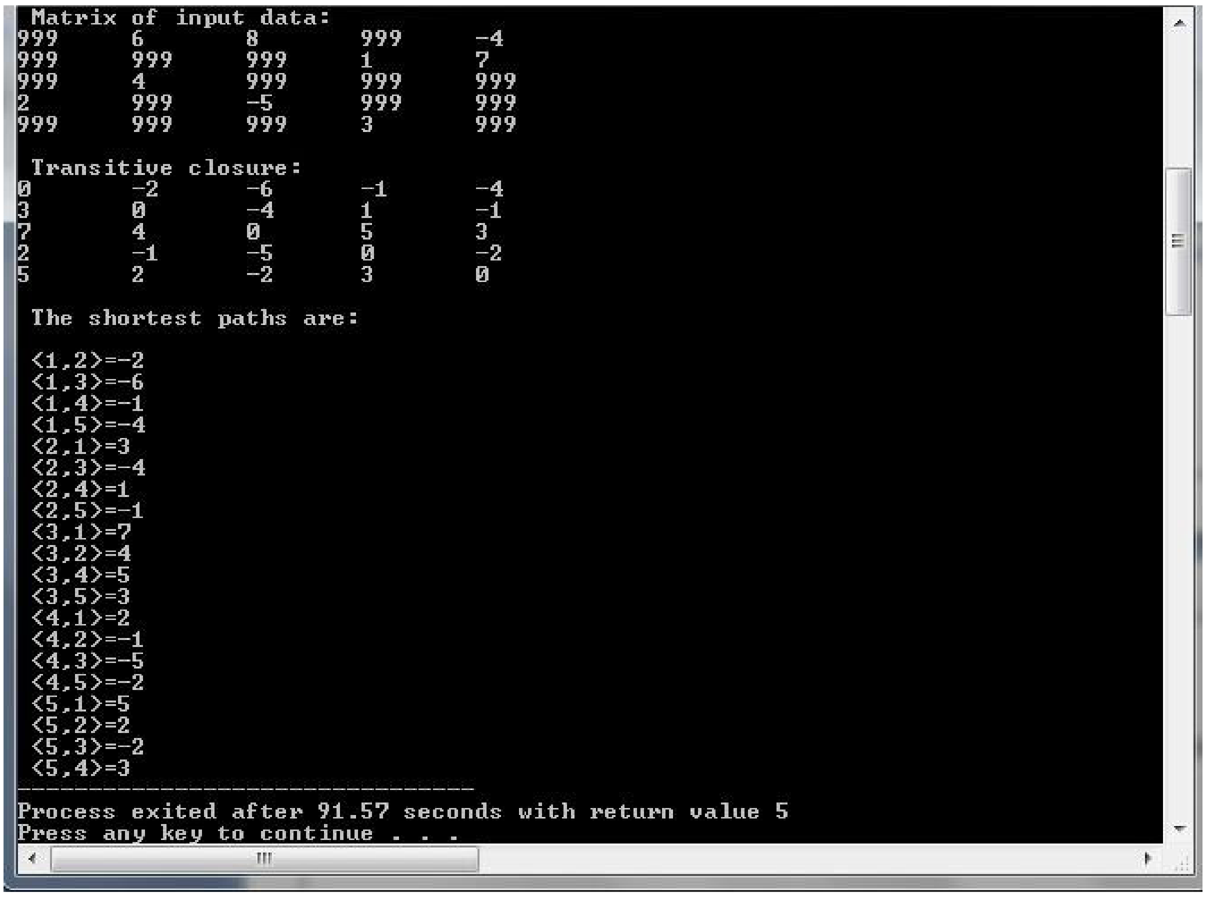
```
#include<stdio.h>
int min(int,int);
voidfloyds(int p[10][10],int n){
inti,j,k;
for(k=1;k<=n;k++)
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(i==j)
p[i][j]=0;
else
p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
```

```
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:



```
 Enter the number of vertices:5
 Enter the number of edges:
9
 Enter the end vertices of edge1 with its weight
1 2 6
 Enter the end vertices of edge2 with its weight
1 3 8
 Enter the end vertices of edge3 with its weight
1 5 -4
 Enter the end vertices of edge4 with its weight
2 4 1
 Enter the end vertices of edge5 with its weight
2 5 7
 Enter the end vertices of edge6 with its weight
3 2 4
 Enter the end vertices of edge7 with its weight
4 1 2
 Enter the end vertices of edge8 with its weight
4 3 -5
 Enter the end vertices of edge9 with its weight
5 4 3
```



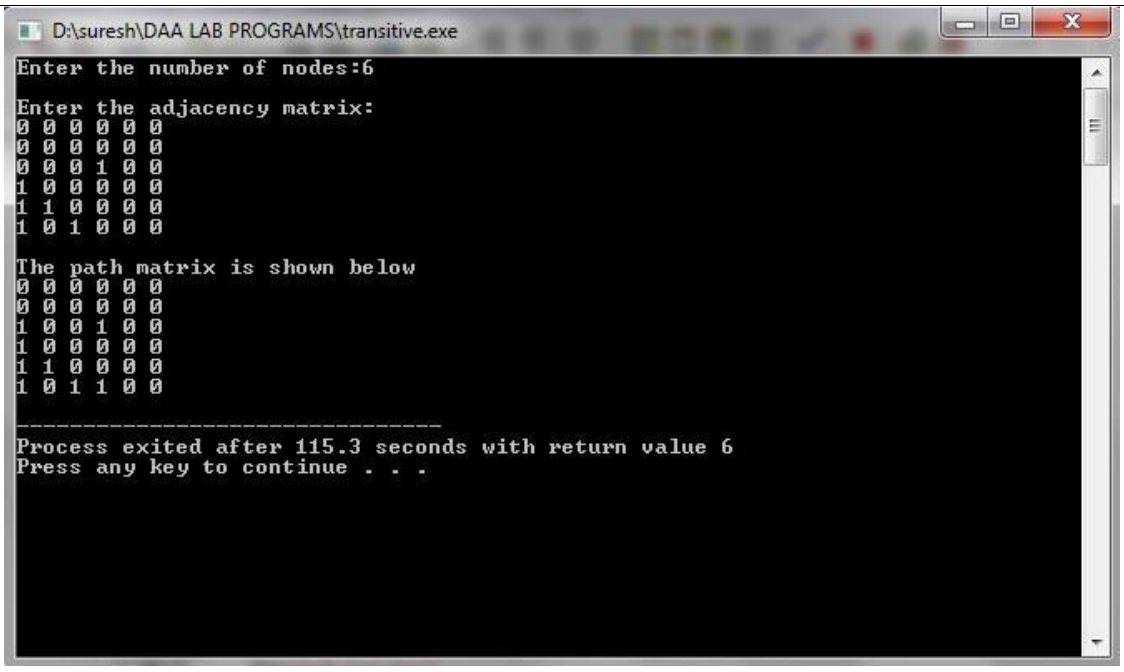
```
 Matrix of input data:
999     6       8       999     -4
999     999     999     1       7
999     4       999     999     999
2       999     -5      999     999
999     999     999     3       999

 Transitive closure:
0       -2      -6      -1      -4
3       0       -4      1       -1
7       4       0       5       3
2       -1      -5      0       -2
5       2       -2      3       0

 The shortest paths are:

 <1,2>=-2
 <1,3>=-6
 <1,4>=-1
 <1,5>=-4
 <2,1>=3
 <2,3>=-4
 <2,4>=1
 <2,5>=-1
 <3,1>=7
 <3,2>=4
 <3,4>=5
 <3,5>=3
 <4,1>=2
 <4,2>=-1
 <4,3>=-5
 <4,5>=-2
 <5,1>=5
 <5,2>=2
 <5,3>=-2
 <5,4>=3
--------------------------------
Process exited after 91.57 seconds with return value 5
Press any key to continue . . .
```

6. Compute the transitive closure of a given directed graph using Warshall's algorithm.
ALGORITHM:
Transitive closure
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. Here reachable mean that there is a path from vertex i to j. 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:


D:\suresh\DAA LAB PROGRAMS\transitive.exe

```
Enter the number of nodes:6

Enter the adjacency matrix:
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 0
1 1 0 0 0 0
1 0 1 0 0 0

The path matrix is shown below
0 0 0 0 0 0
0 0 0 0 0 0
1 0 0 1 0 0
1 0 0 0 0 0
1 1 0 0 0 0
1 0 1 1 0 0

--------------------------------
Process exited after 115.3 seconds with return value 6
Press any key to continue . . .
```

Algorithm Design Techniques

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

ALGORITHM:

Using Recursion

1) A function which calls itself until some condition is called recursive function.
2) In this program, we have two recursive functions available.one is minimum() and another one is maximum(). Both these functions call by itself.
3) The main() function calls the minimum() by passing array,array size,1 as arguments.
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. This function returns the min to the main function. main() function prints the a[min] value of the array.
4) The main() function calls the maximum() function by passing array,array size,1 as arguments.
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. This function returns the max to the main function. 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 : ");
 scanf("%d", &n);
 printf("Enter elements in array : ");
 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))]);
}
```

2. 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*.

Quick sort

ALGORITHM:

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

There are many different versions of QuickSort that pick pivot in different ways.

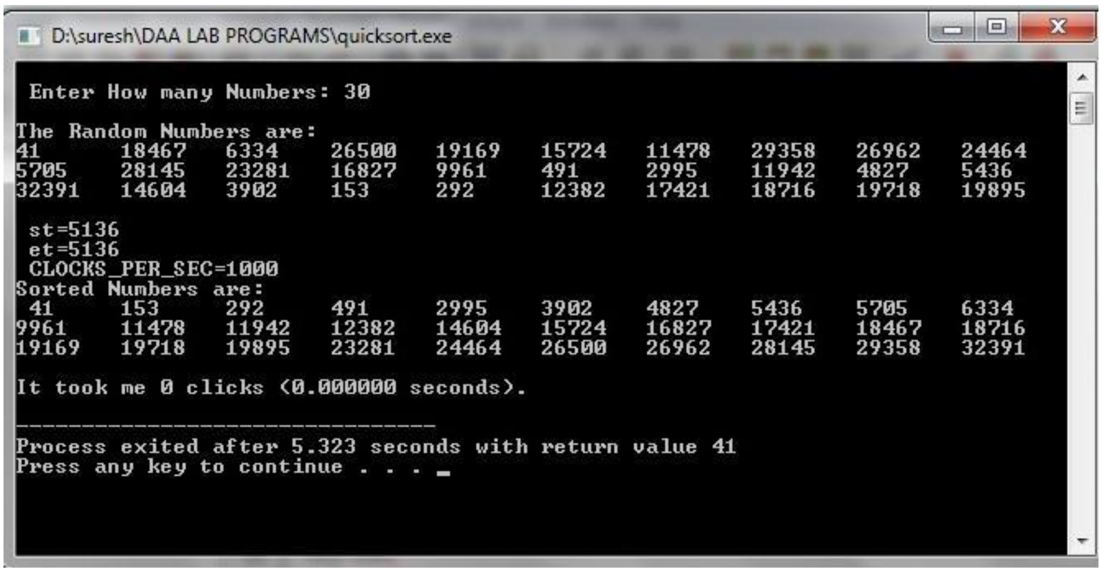
1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)

3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in QuickSort is partition. Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.

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:



```
 Enter How many Numbers: 30

The Random Numbers are:
41      18467   6334    26500   19169   15724   11478   29358   26962   24464
5705    28145   23281   16827   9961    491     2995    11942   4827    5436
32391   14604   3902    153     292     12382   17421   18716   19718   19895

 st=5136
 et=5136
 CLOCKS_PER_SEC=1000
Sorted Numbers are:
 41     153     292     491     2995    3902    4827    5436    5705    6334
9961    11478   11942   12382   14604   15724   16827   17421   18467   18716
19169   19718   19895   23281   24464   26500   26962   28145   29358   32391

It took me 0 clicks (0.000000 seconds).

--------------------------------
Process exited after 5.323 seconds with return value 41
Press any key to continue . . .
```

1. **Merge sort**

ALGORITHM:

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.

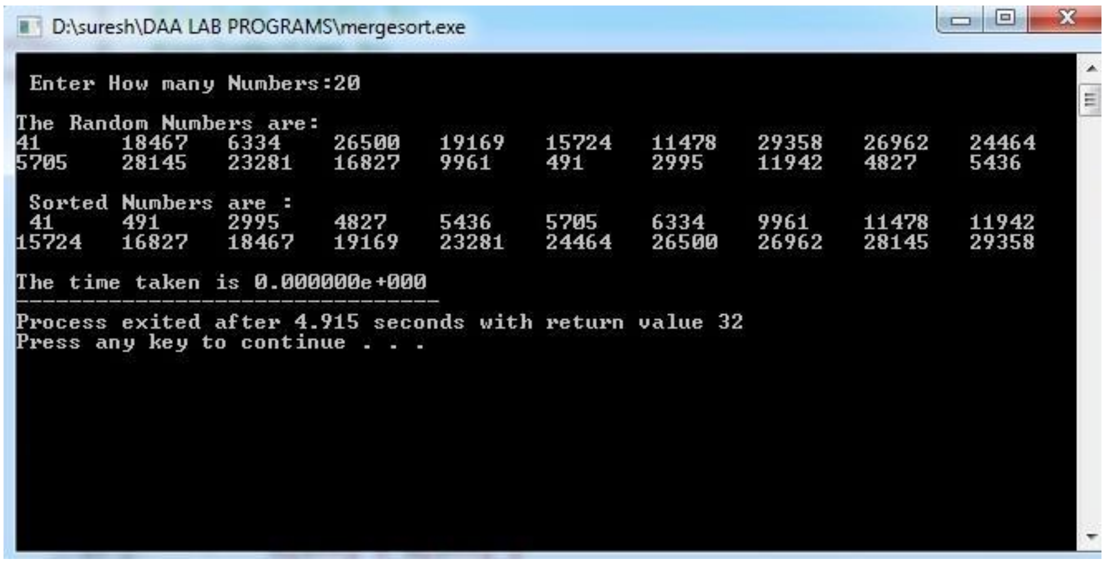
The merge() function is used for merging two halves. The merge(a, low, mid, high) is key process that assumes that a[low..mid] and a[mid+1..high] are sorted and merges the two sorted sub-arrays into one.

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:


D:\suresh\DAA LAB PROGRAMS\mergesort.exe

```
 Enter How many Numbers:20

The Random Numbers are:
41      18467   6334    26500   19169   15724   11478   29358   26962   24464
5705    28145   23281   16827   9961    491     2995    11942   4827    5436

 Sorted Numbers are :
 41     491     2995    4827    5436    5705    6334    9961    11478   11942
15724   16827   18467   19169   23281   24464   26500   26962   28145   29358

The time taken is 0.000000e+000
--------------------------------
Process exited after 4.915 seconds with return value 32
Press any key to continue . . .
```

State Space Search Algorithms

1. Implement N Queens problem using Backtracking.

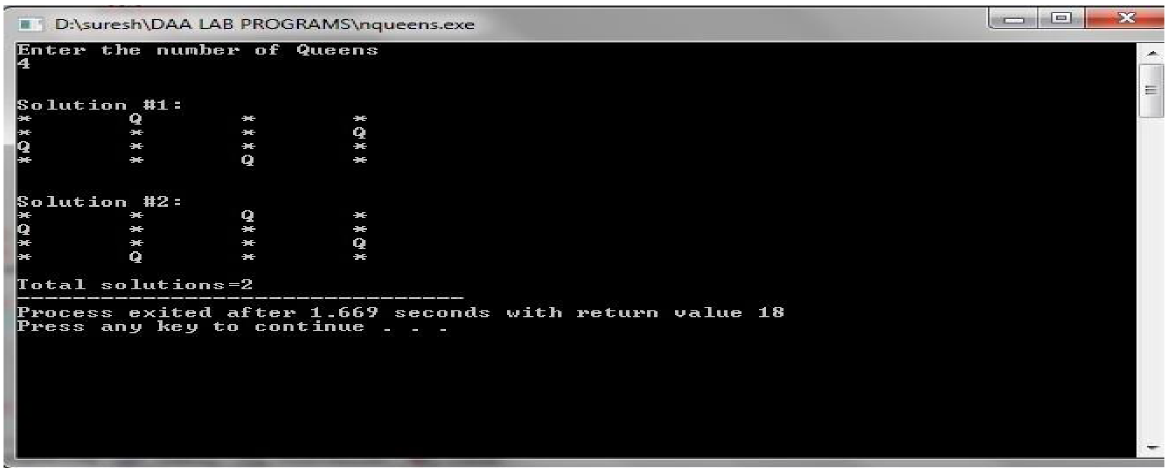
ALGORITHM:

```
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:



```
Enter the number of Queens
4

Solution #1:
*       Q       *       *
*       *       *       Q
Q       *       *       *
*       *       Q       *

Solution #2:
*       *       Q       *
Q       *       *       *
*       *       *       Q
*       Q       *       *

Total solutions=2
--------------------------------
Process exited after 1.669 seconds with return value 18
Press any key to continue . . .
```

Approximation Algorithms Randomized Algorithms

1. 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.

ALGORITHM:

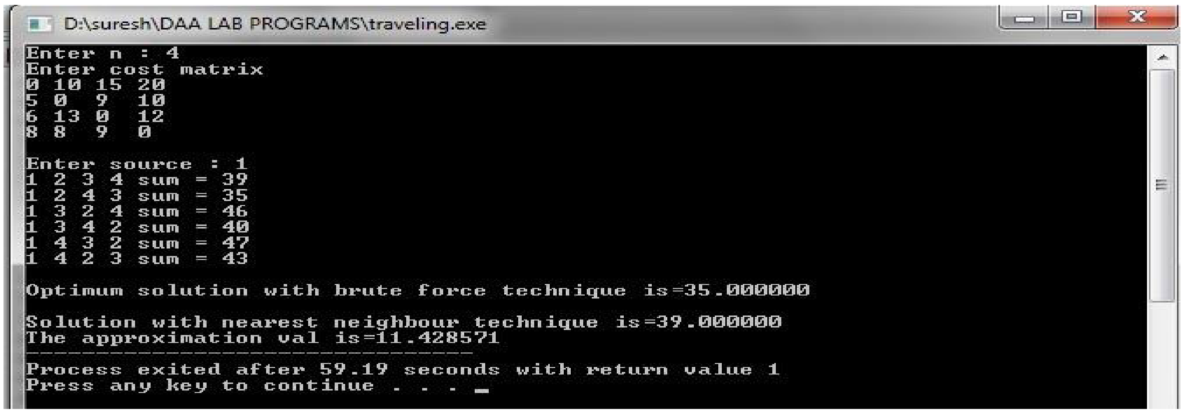
1. Check for the disconnection between the current city and the next city
2. Check whether the travelling sales person has visited all the cities
3. Find the next city to be visited
4. Find the solution and terminate

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:



```
Enter n : 4
Enter cost matrix
0 10 15 20
5 0  9  10
6 13 0  12
8 8  9  0

Enter source : 1
1 2 3 4 sum = 39
1 2 4 3 sum = 35
1 3 2 4 sum = 46
1 3 4 2 sum = 40
1 4 3 2 sum = 47
1 4 2 3 sum = 43

Optimum solution with brute force technique is=35.000000

Solution with nearest neighbour technique is=39.000000
The approximation val is=11.428571
--------------------------------
Process exited after 59.19 seconds with return value 1
Press any key to continue . . .
```

2. 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:

```c
#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;
}
```