```c
#include<stdio.h>
int main()
{
 int n,i,fact=1;
 printf("Enter any number : ");
 scanf("%d", &n);
 for(i=1; i<=n; i++)
 fact = fact * i;
 printf("Factorial value of %d = %d",n,fact);
}
```

```c
#include<stdio.h>
int main(){
int i, j, count, temp, number[25];
printf("How many numbers u are going to enter?: ");
scanf("%d",&count);
printf("Enter %d elements: ", count);
// This loop would store the input numbers in array
for(i=0;i<count;i++)
scanf("%d",&number[i]);
// Implementation of insertion sort algorithm
for(i=1;i<count;i++){
temp=number[i];
j=i-1;
while((temp<number[j])&&(j>=0)){
number[j+1]=number[j];
j=j-1;
}
number[j+1]=temp;
}
printf("Order of Sorted elements: ");
for(i=0;i<count;i++)
printf(" %d",number[i]);
}

```

```c
#include<stdio.h>
int main()
{
 int i,n,temp,j,arr[25];
 printf("Enter the number of elements in the Array: ");
 scanf("%d",&n);
 printf("\nEnter the elements:\n\n");
 for(i=0 ; i<n ; i++)
 {
 printf(" Array[%d] = ",i);
 scanf("%d",&arr[i]);
 }
 for(i=0 ; i<n ; i++)
 {
 for(j=0 ; j<n-i-1 ; j++)
 {
 if(arr[j]>arr[j+1]) //Swapping Condition is Checked
 {
 temp=arr[j];
 arr[j]=arr[j+1];
 arr[j+1]=temp;
 }
 }
 }
 printf("\nThe Sorted Array is:\n\n");
 for(i=0 ; i<n ; i++)
 {
 printf(" %4d",arr[i]);
 }
}
```

```c
#include <stdio.h>
#define max 10
int a[11] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44, 0 };
int b[10];
void merging(int low, int mid, int high) {
int l1, l2, i;
for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
 if(a[l1] <= a[l2])
b[i] = a[l1++];
 else
 b[i] = a[l2++];
}
while(l1 <= mid)
b[i++] = a[l1++];
while(l2 <= high)
b[i++] = a[l2++];
for(i = low; i <= high; i++)
a[i] = b[i];
}
void sort(int low, int high) {
int mid;
if(low < high) {
mid = (low + high) / 2;
sort(low, mid);
sort(mid+1, high);
 merging(low, mid, high);
 } else
{
 return;
 }
 }
 int main()
{
 int i;
 printf("List before sorting\n");
 for(i = 0; i <= max; i++)
 printf("%d ", a[i]);
 sort(0, max);
 printf("\nList after sorting\n");
 for(i = 0; i <= max; i++)
 printf("%d ", a[i]);
 }
```

```c
#include<stdio.h>
int main(){
int list[20],size,i,sElement;
printf("Enter size of the list: ");
scanf("%d",&size);
printf("Enter any %d integer values: ",size);
for(i = 0; i < size; i++)
scanf("%d",&list[i]);
printf("Enter the element to be Search: ");
scanf("%d",&sElement);
// Linear Search Logic
for(i = 0; i < size; i++)
{
if(sElement == list[i])
{
printf("Element is found at %d index", i);
break;
}
}
if(i == size)
printf("Given element is not found in the list!!!");
}
```

```c
#include<stdio.h>
void quickSort(int [10],int,int);
int main(){
int list[20],size,i;
printf("Enter size of the list: ");
scanf("%d",&size);
printf("Enter %d integer values: ",size);
for(i = 0; i < size; i++)
scanf("%d",&list[i]);
quickSort(list,0,size-1);
printf("List after sorting is: ");
for(i = 0; i < size; i++)
printf(" %d",list[i]);
}
void quickSort(int list[10],int first,int last){
int pivot,i,j,temp;
if(first < last)
{
pivot = first;
i = first;
j = last;
while(i < j){
while(list[i] <= list[pivot] && i < last)
i++;
while(list[j] > list[pivot])
j--;
if(i <j)
{
temp = list[i];
list[i] = list[j];
list[j] = temp;
}
}
temp = list[pivot];
list[pivot] = list[j];
list[j] = temp;
quickSort(list,first,j-1);
quickSort(list,j+1,last);
}
}
```

```c
#include<stdio.h>
int main()
{
int first, last, middle, size, i, sElement, list[100];
printf("Enter the size of the list: ");
scanf("%d",&size);
printf("Enter %d integer values in Assending order\n", size);
for (i = 0; i < size; i++)
scanf("%d",&list[i]);
printf("Enter value to be search: ");
scanf("%d", &sElement);
first = 0;
last = size - 1;
middle = (first+last)/2;
while (first <= last) {
if (list[middle] < sElement)
first = middle + 1;
else if (list[middle] == sElement) {
printf("Element found at index %d.\n",middle);
break;
}
else
last = middle - 1;
middle = (first + last)/2;
}
if (first > last)
printf("Element Not found in the list.");
}
```

```c
#include<stdio.h>
int main(){
int a[2][2],b[2][2],c[2][2],i,j;
int m1,m2,m3,m4,m5,m6,m7;
printf("Enter the 4 elements of first matrix: ");
for(i=0;i<2;i++)
for(j=0;j<2;j++)
scanf("%d",&a[i][j]);
printf("Enter the 4 elements of second matrix: ");
for(i=0;i<2;i++)
for(j=0;j<2;j++)
scanf("%d",&b[i][j]);
printf("\nThe first matrix is\n");
for(i=0;i<2;i++){
printf("\n");
for(j=0;j<2;j++)
printf("%d\t",a[i][j]);
}
printf("\nThe second matrix is\n");
for(i=0;i<2;i++){
printf("\n");
for(j=0;j<2;j++)
printf("%d\t",b[i][j]);
}
m1= (a[0][0] + a[1][1])*(b[0][0]+b[1][1]);
m2= (a[1][0]+a[1][1])*b[0][0];
m3= a[0][0]*(b[0][1]-b[1][1]);
m4= a[1][1]*(b[1][0]-b[0][0]);
m5= (a[0][0]+a[0][1])*b[1][1];
m6= (a[1][0]-a[0][0])*(b[0][0]+b[0][1]);
m7= (a[0][1]-a[1][1])*(b[1][0]+b[1][1]);
c[0][0]=m1+m4-m5+m7;
c[0][1]=m3+m5;
c[1][0]=m2+m4;
c[1][1]=m1-m2+m3+m6;
printf("\nAfter multiplication using \n");
for(i=0;i<2;i++){
printf("\n");
for(j=0;j<2;j++)
printf("%d\t",c[i][j]);
}
return 0;
}
```

```c
#include <limits.h>
#include <stdio.h>
void recursiveMinMax(int arr[], int N,
int* minE, int* maxE) {
 if (N < 0) {
return; }
 if (arr[N] < *minE) {
 *minE = arr[N];  }
 if (arr[N] > *maxE) {
 *maxE = arr[N];  }
 recursiveMinMax(arr, N - 1, minE, maxE); }
void findMinimumMaximum(int arr[], int N)
{ int i;
 int minE = INT_MAX, maxE = INT_MIN;
 recursiveMinMax(arr, N - 1, &minE, &maxE);
 printf("The minimum element is %d", minE);
 printf("\n");
 printf("The maximum element is %d", maxE);
 return; }
int main()
{ int arr[] = { 1, 2, 4, -1 };
 int N = sizeof(arr) / sizeof(arr[0]);
findMinimumMaximum(arr, N);
 return 0; }
```

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
typedef struct point
{
 double x;
 double y;
}POINT,VECTOR;
POINT b[1000];
VECTOR normal;
int n;
int upper_lower(int i, VECTOR ab, double c) {
 double x, y,result;
 y = b[i].y;
 x = normal.x*b[i].x;
 result = -(x + c) / normal.y;
 if (y>result) return 1;
 if (y == result) return 0;
 else
 return -1;
}
int ccw(VECTOR v,VECTOR v2)
{
 double cp;
 cp = v2.x*v.y - v2.y*v.x;
if (cp == abs(cp)) return 1;
else
 return -1;
}
double vector_length(VECTOR v)
{
 return sqrt(pow(v.x, 2) + pow(v.y, 2));
}
int cmp_points(const void *p1, const void *p2)
{
 const POINT *pt1 = p1;
 const POINT *pt2 = p2;
if (pt1->x > pt2->x)
 return 1;
if (pt1->x < pt2->x)
 return -1;
if (pt1->y > pt2->y)
 return 1;
if (pt1->y < pt2->y)
 return -1;
return 0;
}
int main()
{
```

```
51   int i,poloha,upper[1000],lower[1000],h=0,d=0;
52   scanf("%d", &n);
53   if (n <= 0 && n > 1000) return 0;
54   for (i = 0; i < n; i++)
55   {
56   scanf("%lf %lf", &b[i].x, &b[i].y);
57   }
58   qsort(b, n, sizeof(POINT), cmp_points);
59  VECTOR ab;
60  double c;
61  ab.x = b[n - 1].x - b[0].x;
62  ab.y = b[n - 1].y - b[0].y;
63  normal.x = -ab.y;
64  normal.y = ab.x;
65  c = -normal.x*b[0].x - (normal.y*b[0].y);
66  for (i = 0; i < n; i++)
67  {
68   poloha = upper_lower(i,ab,c);
69   if (poloha == 1) upper[h++] = i;
70   if (poloha == -1) lower[d++]=i;
71   if (poloha == 0)
72   {
73   upper[h++] = i;
74   lower[d++] = i;
75   }
```

```
76  }
77  int j = 0;
78  double v, length = 0;
79  VECTOR v1, v2, v3,v4;
80  v3.x = 0; v3.y = 0;
81  for (i = 0; ; i++)
82  {
83   int in = 0;
84   if (lower[i + 2] < 0)
85   {
86   v1.x = b[lower[i + 1]].x - b[lower[0]].x;
87   v1.y = b[lower[i + 1]].y - b[lower[0]].y;
88   v2.x = b[lower[i]].x - b[lower[i + 1]].x;
89   v2.y = b[lower[i]].y - b[lower[i + 1]].y;
90   length += vector_length(v1);
91   length += vector_length(v2);
92   break;
93   }
94   v1.x = b[lower[i + 1]].x - b[lower[i]].x;
95   v1.y = b[lower[i + 1]].y - b[lower[i]].y;
96   v2.x = b[lower[i + 2]].x - b[lower[i]].x;
97   v2.y = b[lower[i + 2]].y - b[lower[i]].y;
98   in = ccw(v1, v2);
99   if (in == 1)
100  {
```

```
101   length += vector_length(v1);
102   v3 = v2;
103   v4 = v1;
104   }
105   if (in == -1)
106   {
107   length -= vector_length(v4);
108   if (v3.x != 0 && v3.y != 0)
109   {
110   length += vector_length(v3);
111   v3.x = 0; v3.y = 0;
112   }
113   else
114   {
115   length += vector_length(v2);
116   }
117   }
118 }
119 printf("%.3lf", length);
120 return 0;
121 }
```

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_TREE_HT 100
struct MinHeapNode {
 char data;
 unsigned freq;
 struct MinHeapNode *left, *right;
};
struct MinHeap {
 unsigned size;
 unsigned capacity;
 struct MinHeapNode** array;
};
struct MinHeapNode* newNode(char data, unsigned freq)
{
 struct MinHeapNode* temp = (struct MinHeapNode*)malloc(
 sizeof(struct MinHeapNode));
 temp->left = temp->right = NULL;
 temp->data = data;
 temp->freq = freq;
 return temp;
}
struct MinHeap* createMinHeap(unsigned capacity)
{
 struct MinHeap* minHeap
 = (struct MinHeap*)malloc(sizeof(struct MinHeap));
 minHeap->size = 0;
 minHeap->capacity = capacity;
 minHeap->array = (struct MinHeapNode**)malloc(
 minHeap->capacity * sizeof(struct MinHeapNode*));
 return minHeap;
}
void swapMinHeapNode(struct MinHeapNode** a,
 struct MinHeapNode** b)
{
 struct MinHeapNode* t = *a;
 *a = *b;
 *b = t;
}
 void minHeapify(struct MinHeap* minHeap, int idx)
{
 int smallest = idx;
 int left = 2 * idx + 1;
 int right = 2 * idx + 2;
 if (left < minHeap->size
 && minHeap->array[left]->freq
 < minHeap->array[smallest]->freq)
 smallest = left;
 if (right < minHeap->size
 && minHeap->array[right]->freq
```

```
  < minHeap->array[smallest]->freq)
  smallest = right;
 if (smallest != idx) {
  swapMinHeapNode(&minHeap->array[smallest],
  &minHeap->array[idx]);
  minHeapify(minHeap, smallest);
  }
 }
int isSizeOne(struct MinHeap* minHeap)
{
  return (minHeap->size == 1);
 }
struct MinHeapNode* extractMin(struct MinHeap* minHeap)
{
  struct MinHeapNode* temp = minHeap->array[0];
  minHeap->array[0] = minHeap->array[minHeap->size - 1];
  --minHeap->size;
  minHeapify(minHeap, 0);
  return temp;
 }
void insertMinHeap(struct MinHeap* minHeap,
  struct MinHeapNode* minHeapNode)

{

```

```
  ++minHeap->size;
  int i = minHeap->size - 1;
  while (i
  && minHeapNode->freq
  < minHeap->array[(i - 1) / 2]->freq) {
  minHeap->array[i] = minHeap->array[(i - 1) / 2];
  i = (i - 1) / 2;
  }
  minHeap->array[i] = minHeapNode;
 }
 void buildMinHeap(struct MinHeap* minHeap)
  {
  int n = minHeap->size - 1;
  int i;
  for (i = (n - 1) / 2; i >= 0; --i)
  minHeapify(minHeap, i);
 }
 void printArr(int arr[], int n)
{
  int i;
  for (i = 0; i < n; ++i)
  printf("%d", arr[i]);
  printf("\n");
 }
 int isLeaf(struct MinHeapNode* root)
```

```
{
 return !(root->left) && !(root->right);
}
struct MinHeap* createAndBuildMinHeap(char data[],
 int freq[], int size)
{
 struct MinHeap* minHeap = createMinHeap(size);
 for (int i = 0; i < size; ++i)
 minHeap->array[i] = newNode(data[i], freq[i]);
 minHeap->size = size;
 buildMinHeap(minHeap);
 return minHeap;
}
struct MinHeapNode* buildHuffmanTree(char data[],
 int freq[], int size)
 {
 struct MinHeapNode *left, *right, *top;
 struct MinHeap* minHeap
 = createAndBuildMinHeap(data, freq, size);
 while (!isSizeOne(minHeap)) {
```

```
 left = extractMin(minHeap);
 right = extractMin(minHeap);
 top = newNode('$', left->freq + right->freq);
 top->left = left;
 top->right = right;
 insertMinHeap(minHeap, top);
 }
 return extractMin(minHeap);
}
void printCodes(struct MinHeapNode* root, int arr[],
 int top)
{
 if (root->left) {
 arr[top] = 0;
 printCodes(root->left, arr, top + 1);
 }
 if (root->right) {
 arr[top] = 1;
 printCodes(root->right, arr, top + 1);
 }
```

```
141   if (isLeaf(root)) {
142    printf("%c: ", root->data);
143    printArr(arr, top);
144    }
145   }
146   void HuffmanCodes(char data[], int freq[], int size)
147   {
148    struct MinHeapNode* root
149    = buildHuffmanTree(data, freq, size);
150    int arr[MAX_TREE_HT], top = 0;
151    printCodes(root, arr, top);
152   }
153   int main()
154   {
155    char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
156    int freq[] = { 5, 9, 12, 13, 16, 45 };
157    int size = sizeof(arr) / sizeof(arr[0]);
158    HuffmanCodes(arr, freq, size);
159    return 0;
160   }
```

```c
#include<stdio.h>
int main()
{
 float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;
 int n,i,j;
 printf("Enter the number of items :");
 scanf("%d",&n);
 for (i = 0; i < n; i++)
 {
 printf("Enter Weight and Profit for item[%d] :\n", i);
 scanf("%f %f", &weight[i], &profit[i]);
 }
 printf("Enter the capacity of knapsack :\n");
 scanf("%f",&capacity);
 for(i=0;i<n;i++)
 ratio[i]=profit[i]/weight[i];
 for (i = 0; i < n; i++)
 for (j = i + 1; j < n; j++)
 if (ratio[i] < ratio[j])
 {
 temp = ratio[j];
 ratio[j] = ratio[i];
 ratio[i] = temp;
 temp = weight[j];
 weight[j] = weight[i];
 weight[i] = temp;
 temp = profit[j];
 profit[j] = profit[i];
 profit[i] = temp;
 }
 printf("Knapsack problems using Greedy Algorithm:\n");
 for (i = 0; i < n; i++)
 {
 if (weight[i] > capacity)
 break;
 else
 {
 Totalvalue += profit[i];
 capacity -= weight[i];
 }
 }
 if (i < n)
 Totalvalue = Totalvalue + (ratio[i]*capacity);
 printf("\nThe maximum value is :%f\n",Totalvalue);
 return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
struct node {
 int data;
 struct node* left;
 struct node* right; };
struct node* newNode(int data)
{ struct node* node
 = (struct node*)malloc(sizeof(struct node));
 node->data = data;
 node->left = NULL;
 node->right = NULL;
 return (node); }
void printPostorder(struct node* node)
{ if (node == NULL)
 return;
 printPostorder(node->left);
 printPostorder(node->right);
 printf("%d ", node->data); }
void printInorder(struct node* node)
{ if (node == NULL)
 return;
 printInorder(node->left);
 printf("%d ", node->data);
 printInorder(node->right);
}
void printPreorder(struct node* node)
{
 if (node == NULL)
 return;
 printf("%d ", node->data);
 printPreorder(node->left);
 printPreorder(node->right);
}
int main()
{
 struct node* root = newNode(1);
 root->left = newNode(2);
 root->right = newNode(3);
 root->left->left = newNode(4);
 root->left->right = newNode(5);
 printf("\nPreorder traversal of binary tree is \n");
 printPreorder(root);
 printf("\nInorder traversal of binary tree is \n");
 printInorder(root);
 printf("\nPostorder traversal of binary tree is \n");
 printPostorder(root);
 return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
int main()
{ printf("\n\tImplementation of Kruskal's Algorithm\n");
printf("\nEnter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter the cost 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;  }
} printf("The edges of Minimum Cost Spanning Tree are\n");
while(ne < n)
{ for(i=1,min=999;i<=n;i++)
{ for(j=1;j <= n;j++)
{ if(cost[i][j] < min)
{ min=cost[i][j];
a=u=i;
b=v=j; }
}
```

```c
}
u=find(u);
v=find(v);
if(uni(u,v))
{ printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
mincost +=min;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
}
int find(int i)
{
while(parent[i])
i=parent[i];
return i;
} int uni(int i,int j)
{ if(i!=j)
{
parent[j]=i;
return 1;
}
return 0;
}

```

```c
#include<stdio.h>
#include<stdlib.h>
#define infinity 9999
#define MAX 20
int G[MAX][MAX],spanning[MAX][MAX],n;
int prims();
int main()
{
 int i,j,total_cost;
 printf("Enter no. of vertices:");
 scanf("%d",&n);

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

 for(i=0;i<n;i++)
 for(j=0;j<n;j++)
 scanf("%d",&G[i][j]);

 total_cost=prims();
 printf("\nspanning tree matrix:\n");

 for(i=0;i<n;i++)
 {
 printf("\n");
 for(j=0;j<n;j++)
 printf("%d\t",spanning[i][j]);
 }

 printf("\n\nTotal cost of spanning tree=%d",total_cost);
 return 0;
}

int prims()
{
 int cost[MAX][MAX];
 int u,v,min_distance,distance[MAX],from[MAX];
 int visited[MAX],no_of_edges,i,min_cost,j;
 for(i=0;i<n;i++)
 for(j=0;j<n;j++)
 {
 if(G[i][j]==0)
 cost[i][j]=infinity;
 else
 cost[i][j]=G[i][j];
 spanning[i][j]=0;
 }
 distance[0]=0;
 visited[0]=1;

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

```
 {
 distance[i]=cost[0][i];
 from[i]=0;
 visited[i]=0;
 }

 min_cost=0;
 no_of_edges=n-1;

 while(no_of_edges>0)
 {
 min_distance=infinity;
 for(i=1;i<n;i++)
 if(visited[i]==0&&distance[i]<min_distance)
 {
 v=i;
 min_distance=distance[i];
 }

 u=from[v];
 spanning[u][v]=distance[v];
 spanning[v][u]=distance[v];
 no_of_edges--;
 visited[v]=1;
 for(i=1;i<n;i++)
```

```
 if(visited[i]==0&&cost[i][v]<distance[i])
 {
 distance[i]=cost[i][v];
 from[i]=v;
 }
min_cost=min_cost+cost[u][v];
 }
 return(min_cost);
}
```

```c
#include<stdio.h>
#include<string.h>
int i,j,m,n,c[20][20];
char x[20],y[20],b[20][20];
void print(int i,int j)
{ if(i==0 || j==0)
 return;
 if(b[i][j]=='c')
 { print(i-1,j-1);
 printf("%c",x[i-1]);
 } else if(b[i][j]=='u')
 print(i-1,j);
 else
 print(i,j-1);
}void lcs()
{ m=strlen(x);
 n=strlen(y);
 for(i=0;i<=m;i++)
 c[i][0]=0;
 for(i=0;i<=n;i++)
 c[0][i]=0;
 for(i=1;i<=m;i++)
 for(j=1;j<=n;j++)
 { if(x[i-1]==y[j-1])
 { c[i][j]=c[i-1][j-1]+1;
 b[i][j]='c';
 } else if(c[i-1][j]>=c[i][j-1])
 { c[i][j]=c[i-1][j];
 b[i][j]='u';
 }
 else
 {
 c[i][j]=c[i][j-1];
 b[i][j]='l';
 }
 }
}
int main()
{
 printf("Enter 1st sequence:");
 scanf("%s",x);
 printf("Enter 2nd sequence:");
 scanf("%s",y);
 printf("\nThe Longest Common Subsequence is ");
 lcs();
 print(m,n);
 return 0;
}
```

```c
#include<stdio.h>
#include<math.h>
 int board[20],count;
 int main()
{ int n,i,j;
 void queen(int row,int n);
 printf(" - N Queens Problem Using Backtracking -");
 printf("\n\nEnter number of Queens:");
 scanf("%d",&n);
 queen(1,n);
 return 0; }
void print(int n)
{ int i,j;
 printf("\n\nSolution %d:\n\n",++count);
 for(i=1;i<=n;++i)
 printf("\t%d",i);
 for(i=1;i<=n;++i)
 { printf("\n\n%d",i);
 for(j=1;j<=n;++j) //for nxn board
 { if(board[i]==j)
 printf("\tQ"); //queen at i,j position
 else
 printf("\t-");  }  }
}
int place(int row,int column)
```

```c
{ int i;
  for(i=1;i<=row-1;++i)
  { if(board[i]==column)
  return 0;
  else
  if(abs(board[i]-column)==abs(i-row))
  return 0;
  }
   return 1; //no conflicts
 }
 void queen(int row,int n)
{
  int column;
  for(column=1;column<=n;++column)
  {
  if(place(row,column))
  { board[row]=column;
  if(row==n)
  print(n);
  else
  queen(row+1,n);
  }
  }
 }
```

```c
#include<stdio.h>
#include<stdlib.h>
int a[10][10],visited[10],n,cost=0;
void get()
{
int i,j;
printf("\n\nEnter Number of Cities: ");
scanf("%d",&n);
printf("\nEnter Cost Matrix: \n");
for( i=0;i<n;i++)
{
printf("\n Enter Elements of Row # : %d\n",i+1);
for( j=0;j<n;j++)
scanf("%d",&a[i][j]);
visited[i]=0;
}
printf("\n\nThe Cost Matrix is:\n");
for( i=0;i<n;i++)
{
printf("\n\n");
for(j=0;j<n;j++)
printf("\t%d",a[i][j]);
}
}
void mincost(int city)
```

```c
{
int i,ncity,least(int city);
visited[city]=1;
printf("%d ===> ",city+1);
ncity=least(city);
if(ncity==999)
{
ncity=0;
printf("%d",ncity+1);
cost+=a[city][ncity];
return;
}
mincost(ncity);
}
int least(int c)
{
int i,nc=999;
int min=999,kmin;
for(i=0;i<n;i++)
{
if((a[c][i]!=0)&&(visited[i]==0))
if(a[c][i]<min)
{
min=a[i][0]+a[c][i];
kmin=a[c][i];
```

```
51  nc=i;
52  }
53  }
54  if(min!=999)
55  cost+=kmin;
56  return nc;
57  }
58  void put()
59  {
60  printf("\n\nMinimum cost:");
61  printf("%d",cost);
62  }
63  int main()
64  {
65  get();
66  printf("\n\nThe Path is:\n\n");
67  mincost(0);
68  put();
69  }
```

```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
#define initial 1
#define waiting 2
#define visited 3
 int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void BF_Traversal();
void BFS(int v);
int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();
 int main()
{
 create_graph();
 BF_Traversal();
 return 0;
}
 void BF_Traversal()
{
 int v;
```

```c
 for(v=0; v<n; v++)
 state[v] = initial;
 printf("Enter Start Vertex for BFS: \n");
 scanf("%d", &v);
 BFS(v);
}
void BFS(int v)
{
 int i;
 insert_queue(v);
 state[v] = waiting;
 while(!isEmpty_queue())
 {
 v = delete_queue( );
 printf("%d ",v);
 state[v] = visited;
 for(i=0; i<n; i++)
 {
 if(adj[v][i] == 1 && state[i] == initial)
 {
 insert_queue(i);
 state[i] = waiting;
 }
 }
 }
 printf("\n");
```

```
 printf("\n");
}
 void insert_queue(int vertex)
{
 if(rear == MAX-1)
 printf("Queue Overflow\n");
 else
 {
 if(front == -1)
 front = 0;
 rear = rear+1;
 queue[rear] = vertex ;
 }
}
 int isEmpty_queue()
{ if(front == -1 || front > rear)
 return 1;
 else
 return 0;
}
 int delete_queue()
{ int delete_item;
 if(front == -1 || front > rear)
 {
 printf("Queue Underflow\n");
```

```
 exit(1);
 }
 delete_item = queue[front];
 front = front+1;
 return delete_item;
}
 void create_graph()
{ int count,max_edge,origin,destin;
 printf("Enter number of vertices : ");
 scanf("%d",&n);
 max_edge = n*(n-1);
 for(count=1; count<=max_edge; count++)
 { printf("Enter edge %d( -1 -1 to quit ) : ",count);
 scanf("%d %d",&origin,&destin);
 if((origin == -1) && (destin == -1))
 break;
 if(origin>=n || destin>=n || origin<0 || destin<0)
 { printf("Invalid edge!\n");
 count--;
 }
 else {
 adj[origin][destin] = 1;
 }
 }
}
```

```c
#include <stdio.h>
#include <stdlib.h>
/* ADJACENCY MATRIX */
int source,V,E,time,visited[20],G[20][20];
void DFS(int i)
{
 int j;
 visited[i]=1;
 printf(" %d->",i+1);
 for(j=0;j<V;j++)
 {
 if(G[i][j]==1&&visited[j]==0)
 DFS(j);
 }
}
int main()
{
 int i,j,v1,v2;
 printf("\t\t\tGraphs\n");
 printf("Enter the no of edges:");
 scanf("%d",&E);
 printf("Enter the no of vertices:");
 scanf("%d",&V);
 for(i=0;i<V;i++)
 {
 for(j=0;j<V;j++)
 G[i][j]=0;
 }
 /* creating edges :P */
 for(i=0;i<E;i++)
 {
 printf("Enter the edges (format: V1 V2) : ");
 scanf("%d%d",&v1,&v2);
 G[v1-1][v2-1]=1;
 }
 for(i=0;i<V;i++)
 {
 for(j=0;j<V;j++)
 printf(" %d ",G[i][j]);
 printf("\n");
 }
 printf("Enter the source: ");
 scanf("%d",&source);
 DFS(source-1);
 return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
void random_shuffle(int arr[])
{
 //srand(time(NULL));
 int i, j, temp;
 for (i = MAX - 1; i > 0; i--)
 { j = rand()%(i + 1);
 temp = arr[i];
 arr[i] = arr[j];
 arr[j] = temp; }
} void swap(int *a, int *b)
{ int temp;
 temp = *a;
 *a = *b;
 *b = temp; }
int partion(int arr[], int p, int r)
{  int pivotIndex = p + rand()%(r - p + 1);
 int pivot;
 int i = p - 1;
 int j;
 pivot = arr[pivotIndex];
 swap(&arr[pivotIndex], &arr[r]);
 for (j = p; j < r; j++)
 { if (arr[j] < pivot)
 { i++;
 swap(&arr[i], &arr[j]);
 }
 } swap(&arr[i+1], &arr[r]);
 return i + 1;
}  void quick_sort(int arr[], int p, int q)
{
 int j;
 if (p < q)
 { j = partion(arr, p, q);
 quick_sort(arr, p, j-1);
 quick_sort(arr, j+1, q);
 } }
int main()
{ int i;
 int arr[MAX];
 for (i = 0; i < MAX; i++)
 arr[i] = i;
 random_shuffle(arr);
 quick_sort(arr, 0, MAX-1);
 for (i = 0; i < MAX; i++)
 printf("%d ", arr[i]);
 return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
int match(char [], char []);
int main() {
 char a[100], b[100];
 int position;
 printf("Enter some text\n");
 scanf("%[^\n]s",&a);
 printf("Enter a string to find\n");
  scanf("%s",&b);
 position = match(a, b);
 if (position != -1) {
 printf("Found at location: %d\n", position + 1);
 }
 else {
 printf("Not found.\n");
 }
 return 0;
}
int match(char text[], char pattern[]) {
 int c, d, e, text_length, pattern_length, position = -1;
 text_length = strlen(text);
 pattern_length = strlen(pattern);
 if (pattern_length > text_length) {
 return -1;
 }
 for (c = 0; c <= text_length - pattern_length; c++) {
 position = e = c;
 for (d = 0; d < pattern_length; d++) {
 if (pattern[d] == text[e]) {
 e++;
 }
 else {
 break;
 }
 }
 if (d == pattern_length) {
 return position;
 }
 }
 return -1;
}
```

```
#include <stdio.h>
 #include <stdlib.h>
 #include <sys/time.h>
 #include <omp.h>
void simplemerge(int a[], int low, int mid, int high)
{ int i,j,k,c[20000];
 i=low;
 j=mid+1;
 k=low;
 int tid;
 omp_set_num_threads(10);
 { tid=omp_get_thread_num();
 while(i<=mid&&j<=high)
 { if(a[i] < a[j])
 {
 c[k]=a[i];
 i++;
 k++;
 }
 else
 {
 c[k]=a[j];
 j++;
 k++; }
 }
```

```
 }
 while(i<=mid)
 {  c[k]=a[i];
 i++;
 k++;
 }
 while(j<=high)
 {  c[k]=a[j];
 j++;
 k++;
 }
 for(k=low;k<=high;k++)
 a[k]=c[k];
}
void merge(int a[],int low,int high)
{ int mid;
 if(low < high)
 {
 mid=(low+high)/2;
 merge(a,low,mid);
 merge(a,mid+1,high);
 simplemerge(a,low,mid,high);
 }
}
void getnumber(int a[], int n)
```

```
{ int i;
 for(i=0;i < n;i++)
 a[i]=rand()%100;
}
int main()
{  FILE *fp;
 int a[2000],i;
 struct timeval tv;
 double start, end, elapse;
 fp=fopen("mergesort.txt","w");
 for(i=10;i<=1000;i+=10)
 {
 getnumber(a,i);
 gettimeofday(&tv,NULL);
 start=tv.tv_sec+(tv.tv_usec/1000000.0);
 merge(a,0,i-1);
 gettimeofday(&tv,NULL);
 end=tv.tv_sec+(tv.tv_usec/1000000.0);
 elapse=end-start;
 fprintf(fp,"%d\t%lf\n",i,elapse);
 }
 fclose(fp);
 system("gnuplot");
 return 0;
}
```