1. Program for Breadth First Search (BFS) in C.

```
#include<stdio.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;
void bfs(int v) {
for(i = 1; i <= n; i++)
        if(a[v][i] && !visited[i])
                q[++r] = i;
        if(f <= r) {
                visited[q[f]] = 1;
                bfs(q[f++]);
        }
}
void main() {
        int v;
        printf("\n Enter the number of vertices:");
        scanf("%d", &n);
        for(i=1; i <= n; i++) {         q[i] = 0;
                visited[i] = 0;
        }
        printf("\n Enter graph data in matrix form:\n");
        for(i=1; i<=n; i++) {
                for(j=1;j<=n;j++) {     scanf("%d", &a[i][j]); }
        }
        printf("\n Enter the starting vertex:");
        scanf("%d", &v);
        bfs(v);
        printf("\n The node which are reachable are:\n");
        for(i=1; i <= n; i++) {         if(visited[i])
                        printf("%d\t", i);
                else {          printf("\n Bfs is not possible. Not all nodes are reachable");
                        break;          }
        }
}                                       Output –
```

```
Enter the number of vertices:4

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

Enter the starting vertex:1

The node which are reachable are:
1       2       3       4
Process returned 4 (0x4)   execution time : 23.459 s
Press any key to continue.
```

2. C Program for Depth - First Search in Graph (Adjacency Matrix).

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

Output –

```
                    Graphs
Enter the no of edges:11
Enter the no of vertices:10
Enter the edges (format: V1 V2) : 1 2
Enter the edges (format: V1 V2) : 1 3
Enter the edges (format: V1 V2) : 2 4
Enter the edges (format: V1 V2) : 2 5
Enter the edges (format: V1 V2) : 3 6
Enter the edges (format: V1 V2) : 3 7
Enter the edges (format: V1 V2) : 4 8
Enter the edges (format: V1 V2) : 5 9
Enter the edges (format: V1 V2) : 6 10
Enter the edges (format: V1 V2) : 8 9
Enter the edges (format: V1 V2) : 9 10
 0  1  1  0  0  0  0  0  0  0
 0  0  0  1  1  0  0  0  0  0
 0  0  0  0  0  1  1  0  0  0
 0  0  0  0  0  0  0  1  0  0
 0  0  0  0  0  0  0  0  1  0
 0  0  0  0  0  0  0  0  0  1
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  1  0
 0  0  0  0  0  0  0  0  0  1
 0  0  0  0  0  0  0  0  0  0
Enter the source: 1
 1-> 2-> 4-> 8-> 9-> 10-> 5-> 3-> 6-> 7->
Process returned 0 (0x0)   execution time : 42.232 s
Press any key to continue.
```

3. C IMPLEMETATION of Kruskal's Algorithm.

```
#include<stdio.h>
#include<conio.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);
void 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;                  }
                        }
                }       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);
        getch();
}
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;
}
```

Output –

```
        Implementation of Kruskal's algorithm

Enter the no. of vertices:6

Enter the cost adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0
The edges of Minimum Cost Spanning Tree are
1 edge (1,3) =1
2 edge (4,6) =2
3 edge (1,2) =3
4 edge (2,5) =3
5 edge (3,6) =4

        Minimum cost = 13
```

4. C IMPLEMETATION of Prim's Algorithm.

```
#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("\nEnter the number of nodes:");
        scanf("%d",&n);
        printf("\nEnter 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);
        getch();
}
```

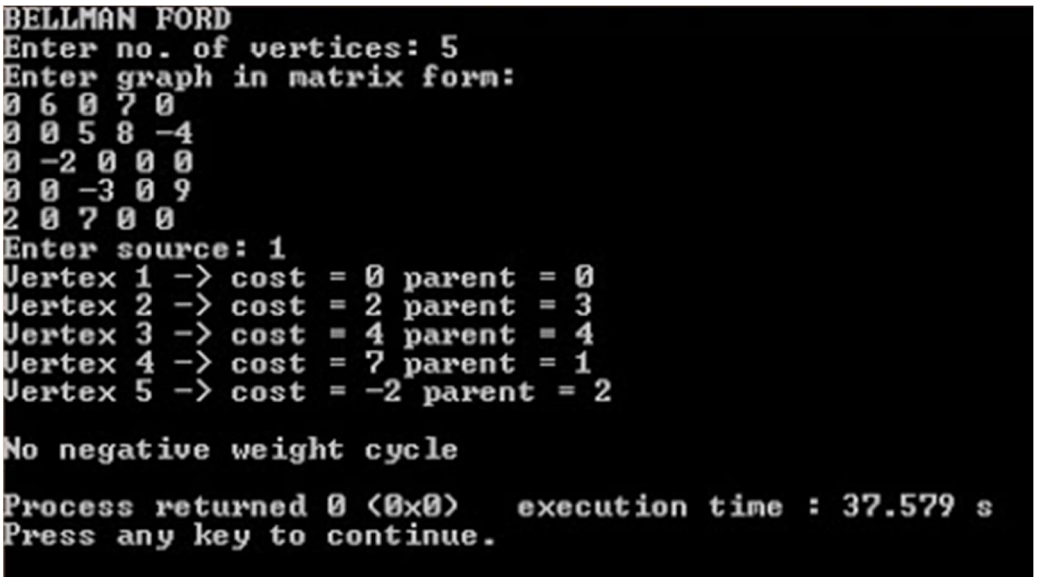
Output –

```
Enter the number of nodes:6

Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0


 Edge 1:(1 3) cost:1
 Edge 2:(1 2) cost:3
 Edge 3:(2 5) cost:3
 Edge 4:(3 6) cost:4
 Edge 5:(6 4) cost:2
 Minimun cost=13_
```

5. Bellman Ford Algorithm to find shortest path in C.

```
#include <stdio.h>
#include <stdlib.h>
int Bellman_Ford(int G[20][20] , int V, int E, int edge[20][2])
{   int i,u,v,k,distance[20],parent[20],S,flag=1;
    for(i=0;i<V;i++)
        distance[i] = 1000 , parent[i] = -1 ;
        printf("Enter source: ");
        scanf("%d",&S);
        distance[S-1]=0 ;
    for(i=0;i<V-1;i++)
    {  for(k=0;k<E;k++)
        {   u = edge[k][0] , v = edge[k][1] ;
            if(distance[u]+G[u][v] < distance[v])
                distance[v] = distance[u] + G[u][v] , parent[v]=u ;        }
    }
    for(k=0;k<E;k++)        {
            u = edge[k][0] , v = edge[k][1] ;
            if(distance[u]+G[u][v] < distance[v])
                flag = 0 ;
        }
        if(flag)
            for(i=0;i<V;i++)
                printf("Vertex %d -> cost = %d parent = %d\n",i+1,distance[i],parent[i]+1);
        return flag;
}
int main()
{  int V,edge[20][2],G[20][20],i,j,k=0;
    printf("BELLMAN FORD\n");
    printf("Enter no. of vertices: ");
    scanf("%d",&V);
    printf("Enter graph in matrix form:\n");
    for(i=0;i<V;i++)
        for(j=0;j<V;j++)
        {  scanf("%d",&G[i][j]);
            if(G[i][j]!=0)
                edge[k][0]=i,edge[k++][1]=j;
        }
    if(Bellman_Ford(G,V,k,edge))
        printf("\nNo negative weight cycle\n");
    else printf("\nNegative weight cycle exists\n");
    return 0;
}
```

Output –


BELLMAN FORD
Enter no. of vertices: 5
Enter graph in matrix form:
0 6 0 7 0
0 0 5 8 -4
0 -2 0 0 0
0 0 -3 0 9
2 0 7 0 0
Enter source: 1
Vertex 1 -> cost = 0 parent = 0
Vertex 2 -> cost = 2 parent = 3
Vertex 3 -> cost = 4 parent = 4
Vertex 4 -> cost = 7 parent = 1
Vertex 5 -> cost = -2 parent = 2
No negative weight cycle
Process returned 0 (0x0) execution time : 37.579 s
Press any key to continue.

6. Program for Dijkstra's Algorithm in C.

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);
int main()
{    int G[MAX][MAX],i,j,n,u;
   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]);
   printf("\nEnter the starting node:");
   scanf("%d",&u);
   dijkstra(G,n,u);
   return 0;                }
void dijkstra(int G[MAX][MAX],int n,int startnode)          {
   int cost[MAX][MAX],distance[MAX],pred[MAX];
   int visited[MAX],count,mindistance,nextnode,i,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];
   for(i=0;i<n;i++)     {  distance[i]=cost[startnode][i];
      pred[i]=startnode;
      visited[i]=0;       }
   distance[startnode]=0;
   visited[startnode]=1;
   count=1;
   while(count<n-1)
   {    mindistance=INFINITY;
      for(i=0;i<n;i++)
         if(distance[i]<mindistance&&!visited[i])       {
           mindistance=distance[i];
            nextnode=i;           }
         visited[nextnode]=1;
         for(i=0;i<n;i++)
            if(!visited[i])
               if(mindistance+cost[nextnode][i]<distance[i])    {
                  distance[i]=mindistance+cost[nextnode][i];
                  pred[i]=nextnode;                                    }
      count++;     }
```

```
    for(i=0;i<n;i++)
        if(i!=startnode)       {
            printf("\nDistance of node%d=%d",i,distance[i]);
            printf("\nPath=%d",i);
            j=i;
            do          {
                j=pred[j];
                printf("<-%d",j);
            }while(j!=startnode);
    }
}
```

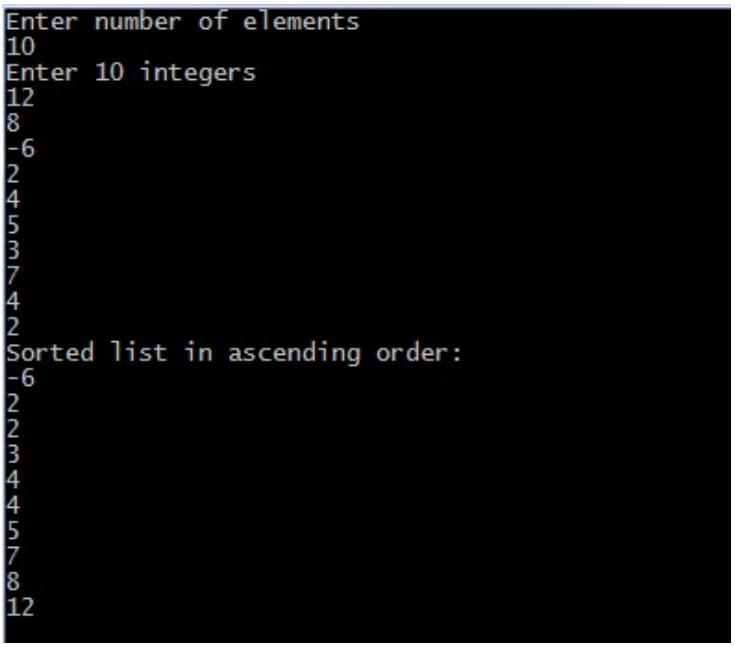
Output –

```
Enter no. of vertices:5

Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0

Enter the starting node:0

Distance of node 1=10
 Path=1<-0
Distance of node 2=50
 Path=2<-3<-0
Distance of node 3=30
 Path=3<-0
Distance of node 4=60
 Path=4<-2<-3<-0
Process returned 5 (0x5)   execution time : 47.471 s
Press any key to continue.
```

7. Selection Sort algorithm implementation in C.

```
#include <stdio.h>
int main()
{
  int array[100], n, c, d, position, swap;
  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]);
  for ( c = 0 ; c < ( n - 1 ) ; c++ )
  {     position = c;
    for ( d = c + 1 ; d < n ; d++ )
    {    if ( array[position] > array[d] )
         position = d;
    }
    if ( position != c )
    { swap = array[c];
      array[c] = array[position];
      array[position] = swap;
    }
  }
 printf("Sorted list in ascending order:\n");
  for ( c = 0 ; c < n ; c++ )
    printf("%d\n", array[c]);
  return 0;
}
```

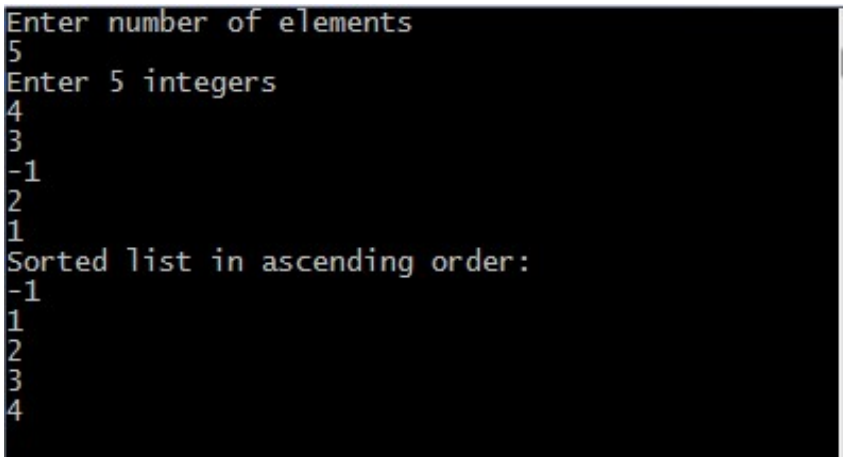
Output –



8. Insertion sort algorithm implementation in C.

```
#include <stdio.h>
int main()
{int n, array[1000], c, d, t;
  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]);
  }
  for (c = 1 ; c <= n - 1; c++) {
    d = c;
    while ( d > 0 && array[d] < array[d-1]) {
      t = array[d];
      array[d]   = array[d-1];
      array[d-1] = t;
      d--;
    }
  }
  printf("Sorted list in ascending order:\n");
  for (c = 0; c <= n - 1; c++) {
    printf("%d\n", array[c]);
  }
  return 0;
}
```

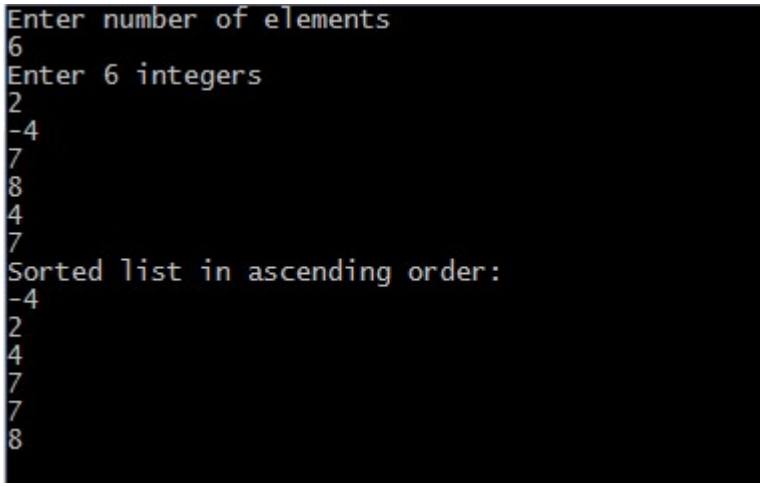
Output –



9. Bubble sort algorithm in C.

```
#include <stdio.h>
int main()
{  int array[100], n, c, d, swap;
  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]);
  for (c = 0 ; c < ( n - 1 ); c++)
  {    for (d = 0 ; d < n - c - 1; d++)
    {   if (array[d] > array[d+1]) /* For decreasing order use < */
      {        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }
  printf("Sorted list in ascending order:\n");
  for ( c = 0 ; c < n ; c++ )
     printf("%d\n", array[c]);
  return 0;
}
```

Output –

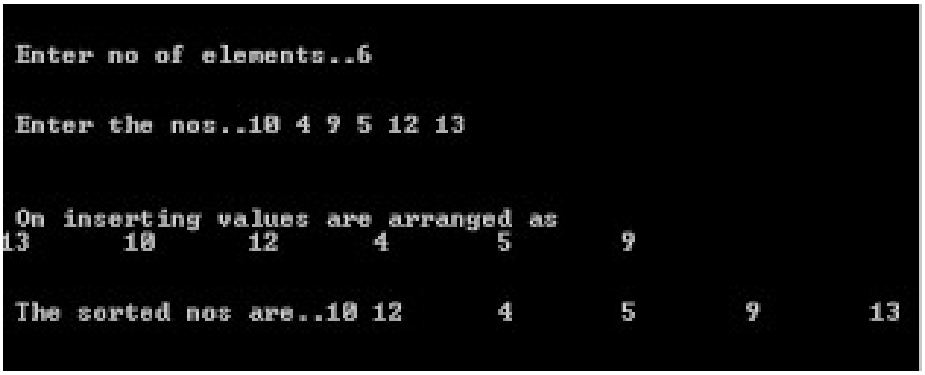


10. C program for Heap sort.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int TREE[10],N,i,j,K,p,c,temp;
    printf("\n\n Enter no of elements..");
    scanf("%d",&N);
    printf("\n\n Enter the nos..");
    for(i=1;i<=N;i++)
    scanf("%d",&TREE[i]);
    for(i=2;i<=N;i++)
    {  K=i;
       do
       {  if(TREE[K]>TREE[K/2])                    // Values are inserted in the heap
          {  temp=TREE[K];
             TREE[K]=TREE[K/2];
             TREE[K/2]=temp;
          }
          p=K/2;
          K=p;
       }
       while(K!=0);
    }
    printf("\n\n\n On inserting values are arranged as \n");
    for(i=1;i<=N;i++)                          // Displaying values in heap
    printf("%d\t",TREE[i]);
    for(j=N;j>0;j--)
    {   temp=TREE[1];
       TREE[1]=TREE[j];
       TREE[j]=temp;
       p=0;
       do
       {                                  // Heap sorting is applied
          c=2*p+2;
          if((TREE[c][/c]<TREE[c language="+1"][/c]) && c<j-1)
          c++;
          if(TREE[p]<TREE[c][/c] && c<j)
          {   temp=TREE[p];
              TREE[p]=TREE[c][/c];
              TREE[c][/c]=temp;
          }
        p=c;
        }
        while(c<(j+1));
    }
```

```
    printf("\n\n\n The sorted nos are..");
    for(i=1;i<=N;i++)                    // printing the heap in sorted order
    printf("%d\t",TREE[i]);
    getch();
}
```

Output –



11. Program for Quick Sort in C.

```
#include<stdio.h>
#include<conio.h>
void quicksort(int array[], int firstIndex, int lastIndex)
{     int pivotIndex, temp, index1, index2;
   if(firstIndex < lastIndex)
   {  pivotIndex = firstIndex;
      index1 = firstIndex;
      index2 = lastIndex;
      while(index1 < index2)
      {   while(array[index1] <= array[pivotIndex] && index1 < lastIndex)
         {           index1++;           }
         while(array[index2]>array[pivotIndex])
         {              index2--;           }
         if(index1<index2)
         {  temp = array[index1];
            array[index1] = array[index2];
            array[index2] = temp;
         }
      }
      temp = array[pivotIndex];
      array[pivotIndex] = array[index2];
      array[index2] = temp;
      quicksort(array, firstIndex, index2-1);
      quicksort(array, index2+1, lastIndex);
   }
}
int main()
{   int array[100],n,i;
   printf("Enter the number of element you want to Sort : ");
   scanf("%d",&n);
   printf("Enter Elements in the list : ");
   for(i = 0; i < n; i++)
   {       scanf("%d",&array[i]);    }
   quicksort(array,0,n-1);
   printf("Sorted elements: ");
   for(i=0;i<n;i++)
   printf(" %d",array[i]);
   getch();
   return 0;
}
```

OUTPUT:-

```
Enter the number of element you want to Sort : 10
Enter Elements in the list : 12
2
9
34
19
55
76
23
90
56
Sorted elements:  2 9 12 19 23 34 55 56 76 90
```

12. Merge Sort program using C.

```
#include <stdio.h>
void quick_sort(int[],int,int);
int partition(int[],int,int);
int main() {
   int a[50],n,i;
   printf("How many elements?");
   scanf("%d",&n);
   printf("\nEnter array elements:");

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

   quick_sort(a,0,n-1);
   printf("\nArray after sorting:");

   for(i=0;i<n;i++)
      printf("%d ",a[i]);

   return 0;
}
void quick_sort(int a[],int l,int u)
{
   int j;
   if(l<u)
   {
      j=partition(a,l,u);
      quick_sort(a,l,j-1);
      quick_sort(a,j+1,u);
   }
} int partition(int a[],int l,int u)
{
   int v,i,j,temp;
   v=a[l];
   i=l;
   j=u+1;
 do
{ do
    i++;
    while(a[i]<v&&i<=u);
      do
         j--;
      while(v<a[j]);

      if(i<j)
      {   temp=a[i];
```

```
            a[i]=a[j];
            a[j]=temp;
        }
    }while(i<j);
    a[l]=a[j];
    a[j]=v;
    return(j);
  }
```

Output-

```
How many elements?5

Enter array elements:23
45
66
788
7

Array after sorting:7 23 45 66 788
Process returned 0 (0x0)   execution time : 9.529 s
Press any key to continue.
```

13. Travelling Salesman Problem in C.

```
#include<stdio.h>
int matrix[25][25], visited_cities[10], limit, cost = 0;
int tsp(int c)
{    int count, nearest_city = 999;
    int minimum = 999, temp;
    for(count = 0; count < limit; count++)
    {        if((matrix[c][count] != 0) && (visited_cities[count] == 0))
        {            if(matrix[c][count] < minimum)
            {                minimum = matrix[count][0] + matrix[c][count];
            }
            temp = matrix[c][count];
            nearest_city = count;
        }
    }
    if(minimum != 999)
    {        cost = cost + temp;    }
    return nearest_city;
}
void minimum_cost(int city)
{    int nearest_city;
    visited_cities[city] = 1;
    printf("%d ", city + 1);
    nearest_city = tsp(city);
    if(nearest_city == 999)
    {        nearest_city = 0;
        printf("%d", nearest_city + 1);
        cost = cost + matrix[city][nearest_city];
        return;
    }
    minimum_cost(nearest_city);
}
int main()
{    int i, j;
    printf("Enter Total Number of Cities:\t");
    scanf("%d", &limit);
    printf("\nEnter Cost Matrix\n");
    for(i = 0; i < limit; i++)
    {        printf("\nEnter %d Elements in Row[%d]\n", limit, i + 1);
        for(j = 0; j < limit; j++)
        {      scanf("%d", &matrix[i][j]);          }
        visited_cities[i] = 0;
    }
    printf("\nEntered Cost Matrix\n");
    for(i = 0; i < limit; i++)
    {        printf("\n");
```

```
        for(j = 0; j < limit; j++)
        {               printf("%d ", matrix[i][j]);          }
    }
    printf("\n\nPath:\t");
    minimum_cost(0);
    printf("\n\nMinimum Cost: \t");
    printf("%d\n", cost);
    return 0;
}
```

Output –

```
Enter Total Number of Cities:   4

Enter Cost Matrix

Enter 4 Elements in Row[1]
1 2 3 4

Enter 4 Elements in Row[2]
5 6 7 8

Enter 4 Elements in Row[3]
3 4 5 6

Enter 4 Elements in Row[4]
9 8 4 3

Entered Cost Matrix

1 2 3 4
5 6 7 8
3 4 5 6
9 8 4 3

Path:   1 4 3 2 1

Minimum Cost:   17
```

14. C Program To Implement Floyd Warshall's Algorithm to Find Path Matrix.

```
#include<stdio.h>
#define LIMIT 100
void show(int mat[LIMIT][LIMIT], int n);
void new_graph();
int adjacency_matrix[LIMIT][LIMIT];
int n;
int main()
{     int P[LIMIT][LIMIT];
    int i, j, k;
    new_graph();
    printf("\nadjacency_matrixacency Matrix\n");
    show(adjacency_matrix, n);
    for(i = 0; i < n; i++)
    {           for(j = 0; j < n; j++)
        {               P[i][j] = adjacency_matrix[i][j];           }
    }
    for(k = 0; k < n; k++)
    {           for(i = 0; i < n; i++)
        {               for(j = 0; j < n; j++)
            {                   P[i][j] = (P[i][j] || (P[i][k] && P[k][j]));                }
        }
        printf("P%d is: \n", k);
        show(P, n);
    }
    printf("P%d is the path matrix of the given graph\n", k - 1);
    return 0;
}
void show(int mat[LIMIT][LIMIT], int n)
{
    int i, j;
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < n; j++)
        {
            printf("%3d", mat[i][j]);
        }
        printf("\n");
    }
}

void new_graph()
{
    int count, maximum_edges, origin, destination;
    printf("Enter Total Number of Vertices:\t");
    scanf("%d", &n);
```

```
    maximum_edges = n * (n - 1);
    for(count = 1; count <= maximum_edges; count++)
    {
        printf("\nCo - Ordinates for Edge No. %d [(-1 -1) To Quit]:\t", count);
        printf("\nEnter Origin Point:\t");
        scanf("%d", &origin);
        printf("\nEnter Destination Point:\t");
        scanf("%d", &destination);
        if((origin == -1) && (destination == -1))
        {
            break;
        }
        if(destination >= n ||  origin < 0 || origin >= n || destination < 0)
        {
            printf("Invalid Edge Input:\n");
            count--;
        }
        else
        {
            adjacency_matrix[origin][destination] = 1;
        }
    }
}
```

Output –

```
Enter Total Number of Vertices: 3

Co - Ordinates for Edge No. 1 [(-1 -1) To Quit]:
Enter Origin Point:     3

Enter Destination Point:        4
Invalid Edge Input:

Co - Ordinates for Edge No. 1 [(-1 -1) To Quit]:
Enter Origin Point:     -1 -1

Enter Destination Point:
adjacency_matrixacency Matrix
  0  0  0
  0  0  0
  0  0  0
P0 is:
  0  0  0
  0  0  0
  0  0  0
P1 is:
  0  0  0
  0  0  0
  0  0  0
P2 is:
  0  0  0
  0  0  0
  0  0  0
P2 is the path matrix of the given graph
```

15. Program for Knapsack Problem in C Using Dynamic Programming.

```
#include<stdio.h>
int max(int a, int b) { return (a > b)? a : b; }
int knapSack(int W, int wt[], int val[], int n)
{   int i, w;
   int K[n+1][W+1];
   for (i = 0; i <= n; i++)
   {       for (w = 0; w <= W; w++)
      {           if (i==0 || w==0)
                K[i][w] = 0;
           else if (wt[i-1] <= w)
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
           else
                K[i][w] = K[i-1][w];
      }
   }
   return K[n][W];
}
int main()
{    int i, n, val[20], wt[20], W;
    printf("Enter number of items:");
    scanf("%d", &n);
    printf("Enter value and weight of items:\n");
    for(i = 0;i < n; ++i){
        {   scanf("%d%d", &val[i], &wt[i]);  }
    }
    printf("Enter size of knapsack:");
    scanf("%d", &W);
    printf("%d", knapSack(W, wt, val, n));
    return 0;
}
```

Output –

```
Enter number of items:3
Enter value and weight of items:
100 20
50 10
150 30
Enter size of knapsack:50
250
```

16. C Program for Longest Common Subsequence Problem.

```
#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;
}
```

Output –

```
Enter 1st sequence:acfghd
Enter 2nd sequence:abfhd

The Longest Common Subsequence is afhd
Process returned 0 (0x0)   execution time : 18.697 s
Press any key to continue.
```

17. Activity Selection Problem in C.

```
#include<stdio.h>
#include<conio.h>
void activities(int s[], int f[], int n)
{       int i, j;
        printf ("Selected Activities are:\n");
        i = 1;
        printf("A%d ", i);
        for (j = 1; j < n; j++)
        {       if (s[j] >= f[i])
                {       printf ("A%d ", j+1);
                i = j;
                }
        }
}
void main()
{       int s[] =  {1, 3, 0, 5, 3, 5, 6, 8, 8, 2, 12};
        int f[] =  {4, 5, 6, 7, 9, 9, 10, 11, 12, 14, 16};
        int n = sizeof(s)/sizeof(s[0]);
        clrscr();
        activities(s, f, n);
        getchar();
        getch();
}
```

Output –

```
Selected Activities are:
A1 A4 A8 A11
```

18. Program for N Queens Problem in C Using Backtracking.

```
#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-"); //empty slot
  }
 }
}
 /*funtion to check conflicts
If no conflict for desired postion returns 1 otherwise returns 0*/
int place(int row,int column)
{ 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
}
 //function to check for proper positioning of queen
void queen(int row,int n)
{
 int column;
```

```
 for(column=1;column<=n;++column)
 {
  if(place(row,column))
  {
   board[row]=column; //no conflicts so place queen
   if(row==n) //dead end
    print(n); //printing the board configuration
   else //try queen with next position
    queen(row+1,n);
  }
 }
}
```

Output –

```
Enter number of Queens:4

Solution 1:

        1       2       3       4

1       -       Q       -       -

2       -       -       -       Q

3       Q       -       -       -

4       -       -       Q       -

Solution 2:

        1       2       3       4

1       -       -       Q       -

2       Q       -       -       -

3       -       -       -       Q

4       -       Q       -       -
```

19. Program for Matrix Chain Multiplication in C

```
#include<stdio.h>
#include<limits.h>
 // Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
 int MatrixChainMultiplication(int p[], int n)
{    int m[n][n];
   int i, j, k, L, q;
    for (i=1; i<n; i++)
       m[i][i] = 0;    //number of multiplications are 0(zero) when there is only one matrix
    //Here L is chain length. It varies from length 2 to length n.
   for (L=2; L<n; L++)
   {        for (i=1; i<n-L+1; i++)
      {           j = i+L-1;
         m[i][j] = INT_MAX;  //assigning to maximum value
          for (k=i; k<=j-1; k++)
          {              q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
            if (q < m[i][j])
             {             m[i][j] = q;                  }
          }
      }
   }
    return m[1][n-1];   //returning the final answer which is M[1][n]
 }
 int main()
{    int n,i;
   printf("Enter number of matrices\n");
   scanf("%d",&n);
    n++;
    int arr[n];
    printf("Enter dimensions \n");
    for(i=0;i<n;i++)
   {  printf("Enter d%d :: ",i);
      scanf("%d",&arr[i]);
   }
    int size = sizeof(arr)/sizeof(arr[0]);
    printf("Minimum number of multiplications is %d ", MatrixChainMultiplication(arr, size));
    return 0;
}
```

Output –

```
Enter number of matrices
4
Enter dimensions
Enter d0 :: 10
Enter d1 :: 100
Enter d2 :: 20
Enter d3 :: 5
Enter d4 :: 80
Minimum number of multiplications is 19000
```