1. BFS

```
#include<stdio.h>
#define MAXSIZE 20

struct queue
{
    int data[MAXSIZE];
    int front,rear;
};

void initq(struct queue *pq)
{
   pq->front=pq->rear = -1;
}
 void addq(struct queue *pq, int n)
{
    pq->data[++pq->rear]=n;

}
int removeq (struct queue *pq)
{
    return pq->data[++pq->front];
}
int isempty (struct queue *pq)
{
    return (pq->front==pq->rear);
}
int isfull(struct queue *pq)
{
    return (pq->rear==MAXSIZE-1);
}
void bfs(int m[10][10],int n)
{
    int i,j,v,w;
    int visited[20]={0};
    struct queue q;
    initq(&q);
    printf("The BFS traversal is:\n");
    v=1;
    visited[v]=1;
    addq(&q,v);
    while(!isempty(&q))
    {
        v=remove(&q);
        printf("v%d",v);
        for(w=1;w<=n;w++)
        {
            if((m[v][w]==1)&&(!visited[w]))
```

```
        {
            addq(&q,w);
            visited[w]=1;
        }
    }
    }
}

int main()
{
   int m[5][5]={{0,0,1,1,0},{0,0,1,0,1},{0,1,0,0,0},{0,0,0,0,1},{0,0,0,0,0}};
   bfs(m,5);
}
```

2. DFS

```
#include <stdio.h>
#define MAX 100

int visited[MAX] = {0};
int adjacency_matrix[MAX][MAX];
int n;

void DFS(int vertex) {
    int i;
    visited[vertex] = 1;
    printf("%d ", vertex);

    for(i = 0; i < n; i++) {
        if(adjacency_matrix[vertex][i] && !visited[i]) {
            DFS(i);
        }
    }
}

int main() {
    int i, j, start_vertex;

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    printf("Enter the adjacency matrix: \n");
    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
            scanf("%d", &adjacency_matrix[i][j]);
        }
```

```
    }

    printf("Enter the starting vertex: ");
    scanf("%d", &start_vertex);

    printf("DFS traversal: ");
    DFS(start_vertex);

    return 0;
}
```

3. Dijkstraj

```
#include <stdio.h>
#define MAX 100
#define INF 99999

int adjacency_matrix[MAX][MAX];
int num_vertices;

void dijkstra(int source_vertex) {
    int distance[MAX], visited[MAX], i, j, min_distance, next_vertex;

    // initialize all distances to infinity and visited to false
    for (i = 0; i < num_vertices; i++) {
        distance[i] = INF;
        visited[i] = 0;
    }

    // set the distance to the source vertex as 0
    distance[source_vertex] = 0;

    for (i = 0; i < num_vertices - 1; i++) {
        min_distance = INF;

        // find the next vertex to visit
        for (j = 0; j < num_vertices; j++) {
            if (!visited[j] && distance[j] < min_distance) {
                next_vertex = j;
                min_distance = distance[j];
            }
        }

        visited[next_vertex] = 1;
```

```
        // update the distances of the neighboring vertices
        for (j = 0; j < num_vertices; j++) {
            if (!visited[j] && adjacency_matrix[next_vertex][j] &&
distance[next_vertex] + adjacency_matrix[next_vertex][j] < distance[j]) {
                distance[j] = distance[next_vertex] +
adjacency_matrix[next_vertex][j];
            }
        }
    }

    // print the distances from the source vertex
    printf("Distances from source vertex %d: \n", source_vertex);
    for (i = 0; i < num_vertices; i++) {
        printf("%d: %d\n", i, distance[i]);
    }
}

int main() {
    int i, j, source_vertex;

    printf("Enter the number of vertices: ");
    scanf("%d", &num_vertices);

    printf("Enter the adjacency matrix: \n");
    for(i = 0; i < num_vertices; i++) {
        for(j = 0; j < num_vertices; j++) {
            scanf("%d", &adjacency_matrix[i][j]);
        }
    }

    printf("Enter the source vertex: ");
    scanf("%d", &source_vertex);

    dijkstra(source_vertex);

    return 0;
}
```

4. Heap Sort

```
#include<stdio.h>
void heapsort(int a[],int n);
void buildheap(int a[],int n);
void heapify(int a[],int i,int last);
void display(int a[],int n);
int main()
{
    int a[7]={17,4,19,2,16,8,28};
    heapsort(a,7);
    printf("Sorted elements are \n");
    display(a,7);
}
void heapsort(int a[7],int n)
{
    int temp,i=0,last;
    buildheap(a,n);
    printf("Initial heap= \n");
    display(a,n);
    for(last=n-1;last>=1;last--)
    {
        temp=a[i];
        a[i]=a[last];
        a[last]=temp;
        printf("After iteartion %d\n",n-last);
        display(a,n);
        heapify(a,i,last-1);
    }
}

void buildheap(int a[7],int n)
{
    int i;
    for(i=n/2-1;i>=0;i--)
        heapify(a,i,n-1);

}

void heapify(int a[],int i,int last)
{
   int l,temp,max;
    max=a[i];
    l=2*i+1;
    if((l<last)&&(a[l]<a[l+1]))
        l=l+1;
     if((l<=last)&&(max<a[l]))
     {
```

```
51.         temp=a[i];
52.         a[i]=a[l];
53.         a[l]=temp;
54.         heapify(a,l,last);
55.      }
56.}
57.
58.void display(int a[],int n)
59.{
60.     int i;
61.     for(i=0;i<n;i++)
62.         printf("%d\n",a[i]);
63.}
64.
```

5. Insertion

```
#include<stdio.h>
int main()
{
 int i,j,n,x,a[10];
 printf("\n Enter the no of elements:");
 scanf("%d",&n);
 printf("\n Enter the unsorted data:");
    for(i=0;i<n;i++)
 scanf("%d",&a[i]);
 printf("\n Display the unsorted data:");
 for(i=0;i<n;i++)
   printf("%4d",a[i]);
 for(i=1;i<n;i++)
  {
   x=a[i];
   for(j=i-1;j>=0 && x<a[j];j--)
   a[j+1]=a[j];
   a[j+1]=x;
  }
  printf("\n Display the sorted data:");
  for(i=0;i<n;i++)
     printf("%4d",a[i]);
  return 0;
}
/*
          OUTPUT
 Enter the no of elements:4

 Enter the unsorted data:8
6
```

```
4
2

 Display the unsorted data:   8   6   4   2
 Display the sorted data:   2   4   6   8
*/
```

6. Kruskal

```
#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;
            }
          }
        }
        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);
    return 0;
  }
  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;
  }
```

7. Prims

```
#include <stdio.h>

void prim(int n,int cost[10][10])
{
    int visited[10] = {0}, i, j, min_cost = 0, min, u, v,e;

    visited[1] = 1;

        for(e = 1; e <= n; e++)
            {
                for(i = 1,min=999; i <= n; i++)
                    for(j = 1;j <= n; j++)
                    {
                        if(cost[i][j]==0)
                            cost[i][j]=999;
                        if(cost[i][j]<min)
                            if(visited[i]!=0)
                        {
                            min= cost[i][j];
                            u=i;
```

```
                            v=j;
                        }
                    }

        if(visited[u]==0 || visited[v] == 0)
        {
         printf("Eage %d:(%d, %d) cost: %d\n",e, u, v, min);
         min_cost += min;
         visited[v]=1;
        }
        cost[u][v] = cost[v][u]=999;
    }

    printf("Minimum cost: %d\n", min_cost);
}

int main() {
    int i, j,n,cost[10][10];

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    printf("Enter the adjacency matrix: \n");
    for(i = 1; i <= n; i++)
        {
            for(j = 1; j <= n; j++)
            {
                scanf("%d", &cost[i][j]);
            }
        }
    printf("MST edges: \n");
    prim(n,cost);

    return 0;
}

```

8. Selection

```
#include<stdio.h>
#include<conio.h>
void selectionsort(int a[],int n);
void display(int a[],int n);
int main()
{
    int a[10],i,n;
  printf("\n Enter the number of elements:");
  scanf("%d",&n);
```

```
  printf("\n Enter array elements:");
  for(i=0;i<n;i++)
    scanf("%d",&a[i]);
  printf("\n Sorted elements are:");
  selectionsort(a,n);
  display(a,n);
}
void display(int a[],int n)
{
    int i;
    for(i=0;i<n;i++)
     printf("\t%d",a[i]);
}

void selectionsort(int a[], int n)
{
    int i, j, min,temp;
    for (i = 0; i < n-1; i++)
    {
        min =i;
        for (j = i+1; j < n; j++)
        {
           if (a[j] < a[min])

            min = j;
        }
               temp=a[j];
               a[j]=a[j+1];
               a[j+1]=temp;

    }
}
```

9. Topological Sort

```c
#include <stdio.h>
#include<stdlib.h>
#include<conio.h>
#define MAXSIZE 20
typedef struct
{
    int data[MAXSIZE];
    int top;
}STACK;
void init(STACK *ps)
{
  ps->top=-1;
}
int isempty(STACK *ps)
{
  return(ps->top==-1);
}
void push(STACK *ps, int n)
{
  ps->data[++ps->top]=n;
}
int pop(STACK *ps)
{
  return ps->data[ps->top--];
}

void topological_sort(int m[10][10],int n)
 {
    int i,j,v,w;
    int indeg[10];
    int visited[10]={0};
    printf("Enter the matrix:");
    for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                scanf("%d",&m[i][j]);
            }
        }

    for(i=0;i<n;i++)
        {
        indeg[i]=0;
        for(j=0;j<n;j++)
        {
```

```
            if(i!=j)
            indeg[i] = indeg[i]+ m[j][i];
        }

        printf("Indegree of v%d=%d\t",i+1,indeg[i]);

 }
   printf("\nTopological sort: ");
    STACK s;
    init(&s);
    while(1)
    {
     for(v=0;v<n;v++)
         if((visited[v]==0)&&(indeg[v]==0))
         {
             visited[v]=1;
             push(&s,v);
             printf("v%d",v+1);
         }
     if(isempty(&s))
        break;
     v=pop(&s);
     for(w=0;w<n;w++)
         if(m[v][w]==1)
            indeg[w]=indeg[w]-1;
   }
}
int main()
{
    //int m[4][4]={{0,1,1,0},{0,0,1,1},{0,0,0,1},{0,0,0,0}};
    int m[10][10],n;
    printf("How many vertices :");
    scanf("%d",&n);
       topological_sort(m,n);

}
```