1.IMPLEMENTATION OF LINEAR SEARCH:

```
#include <stdio.h>
void main()
{
	int arr[50];
	int i, key,pos,n;
	printf("enter array size:");
	scanf("%d", &n);
	printf("enter array:");
	for(i=0;i<n;i++)
	{
		scanf("%d",&arr[i]);

	}
	printf("enter key =");
	scanf("%d", &key);
	for(i=0;i<n;i++);
	{
		if(arr[i]==key)
		{
			pos=i;
			flag=1;
			break;
		}
	}
	if(flag==1)
		prinf("element found at the position %d",i);
	else
		printf("element not found");
}
```

2.IMPLEMENTATION OF BINARY SEARCH:

```
#include <stdio.h>
int binarySearch(int arr[], int l, int r, int x)
{
   if (r >= l) {
      int mid = l + (r - l) / 2;
```

```
    if (arr[mid] == x)
       return mid;
    if (arr[mid] > x)
       return binarySearch(arr, l, mid - 1, x);
    return binarySearch(arr, mid + 1, r, x);
  }

  return -1;

}

void main()
{
        int arr[50];
        int i, key,pos,n;
        printf("enter array size:");
        scanf("%d", &n);
        printf("enter array:");
        for(i=0;i<n;i++)
        {
                scanf("%d",&arr[i]);

        }
        printf("enter key =");
        scanf("%d", &key);

        int found = binarySearch(arr,0, n-1, key);
        if(found == -1)
                printf("not found");
        else
                printf("Element is found at position %d",found+1);
}
```

3.IMPLEMENTATION OF BUBBLE SORT:

```
#include <stdio.h>
void swap(int *x, int *y)
{
        int temp = *x;
        *x = *y;
        *y = temp;
```

```
}

void bubble_sort(int arr[], int n)
{
	int i, j;
	for (i = 0; i < n-1; i++)
	{
		for (j = 0; j < n-i-1; j++)
		{
			if (arr[j] > arr[j+1])
				swap(&arr[j], &arr[j+1]);
		}
	}
}

void main()
{
	int arr[50];
	int i, key,pos,n;
	printf("enter array size:");
	scanf("%d", &n);
	printf("enter array:\n");
	for(i=0;i<n;i++)
	{
		scanf("%d",&arr[i]);
	}
	bubble_sort(arr, n);
	printf("Sorted array: \n");
	for(i=0;i<n;i++)
	{
		printf("%d, ",arr[i]);
	}
}
```

4.IMPLEMENTATION OF INSERTION SORT:

```
#include<stdio.h>

void insertion_sort(int arr[], int n)
{
```

```
   int i, key, j;
   for (i = 1; i < n; i++)
   {
      key = arr[i];
      j = i - 1;

      /* Move elements of arr[0..i-1], that are greater than key, to one position ahead of their current
position */
      while (j >= 0 && arr[j] > key)
     {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
   }
}
void main()
{
            int arr[50];
            int i, key,pos,n;
            printf("enter array size:");
            scanf("%d", &n);
            printf("enter array:\n");
            for(i=0;i<n;i++)
            {
                        scanf("%d",&arr[i]);
            }
            insertion_sort(arr, n);
            printf("Sorted array: \n");
            for(i=0;i<n;i++)
            {
                        printf("%d, ",arr[i]);
            }
}
```

5.IMPLEMENTATION OF SELECTION SORT:

```
#include <stdio.h>

void swap(int *x, int *y)
{
```

```
	int temp = *x;
	*x = *y;
	*y = temp;
}

void selection_sort(int arr[], int n)
{
	int i, j, min_idx;
	for (i = 0; i < n-1; i++)
	{
		min_idx = i;
		for (j = i+1; j < n; j++)
		if (arr[j] < arr[min_idx])
			min_idx = j;
		swap(&arr[min_idx], &arr[i]);
	}
}

void main()
{
	int arr[50];
	int i, key,pos,n;
	printf("enter array size:");
	scanf("%d", &n);
	printf("enter array:\n");
	for(i=0;i<n;i++)
	{
		scanf("%d",&arr[i]);

	}
	selection_sort(arr, n);
	printf("Sorted array: \n");
	for(i=0;i<n;i++)
	{
		printf("%d, ",arr[i]);

	}
}
```

6.IMPLEMENTATION OF MERGE SORT:

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

/* Merges two subarrays of arr[].
First subarray is arr[l..m]
Second subarray is arr[m+1..r] */
void merge(int arr[], int l, int m, int r)
{
	int i, j, k;
	int n1 = m - l + 1;
	int n2 = r - m;
	int L[n1], R[n2];
	for (i = 0; i < n1; i++)
		L[i] = arr[l + i];
	for (j = 0; j < n2; j++)
		R[j] = arr[m + 1+ j];

	/* Merge the temp arrays back into arr[l..r]*/
	i = 0; // Initial index of first subarray
	j = 0; // Initial index of second subarray
	k = l; // Initial index of merged subarray
	while (i < n1 && j < n2)
	{
		if (L[i] <= R[j])
		{
			arr[k] = L[i];
			i++;
		}
		else
		{
			arr[k] = R[j];
			j++;
		}
		k++;
	}

	/* Copy the remaining elements of L[], if there
	are any */
```

```
	while (i < n1)
	{
		arr[k] = L[i];
		i++;
		k++;
	}

	/* Copy the remaining elements of R[], if there
	are any */
	while (j < n2)
	{
		arr[k] = R[j];
		j++;
		k++;
	}
}

void merge_sort(int arr[], int l, int r)
{
	if (l < r)
	{
		int m = l+(r-l)/2;
		merge_sort(arr, l, m);
		merge_sort(arr, m+1, r);
		merge(arr, l, m, r);
	}
}

void main()
{
	int arr[50];
	int i, key,pos,n;
	printf("enter array size:");
	scanf("%d", &n);
	printf("enter array:\n");
	for(i=0;i<n;i++)
	{
		scanf("%d",&arr[i]);

	}
	merge_sort(arr, 0, n-1);
	printf("Sorted array: \n");
```

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

	}
}
```

7.IMPLEMENTATION OF QUICK SORT:

```
#include<stdio.h>

void swap(int* x, int* y)
{
	int t = *x;
	*x = *y;
	*y = t;
}

int partition(int arr[], int low, int high)
{
	int pivot = arr[high]; // pivot
	int i = (low - 1);
	int j;
	for (j = low; j <= high- 1; j++)
	{
		if (arr[j] < pivot)
		{
			i++;
			swap(&arr[i], &arr[j]);
		}
	}
	swap(&arr[i + 1], &arr[high]);
	return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
```

```
void quick_sort(int arr[], int low, int high)
{
	if (low < high)
	{
		int pi = partition(arr, low, high);
		quick_sort(arr, low, pi - 1);
		quick_sort(arr, pi + 1, high);
	}
}

void main()
{
	int arr[50];
	int i, key,pos,n;
	printf("enter array size:");
	scanf("%d", &n);
	printf("enter array:\n");
	for(i=0;i<n;i++)
	{
		scanf("%d",&arr[i]);

	}
	quick_sort(arr, 0, n-1);
	printf("Sorted array: \n");
	for(i=0;i<n;i++)
	{
		printf("%d, ",arr[i]);

	}
}
```

8.IMPLEMENTATION OF HEAP SORT:

```
#include<stdio.h>

void swap(int* x, int* y)
{
	int t = *x;
	*x = *y;
	*y = t;
}
```

```
void heapify(int arr[], int n, int i)
{
   int largest = i; // Initialize largest as root
   int l = 2*i + 1; // left = 2*i + 1
   int r = 2*i + 2; // right = 2*i + 2
   if (l < n && arr[l] > arr[largest])
      largest = l;
   if (r < n && arr[r] > arr[largest])
      largest = r;
   if (largest != i)
   {
      swap(&arr[i], &arr[largest]);
      heapify(arr, n, largest);
   }
}

void heap_sort(int arr[], int n)
{
            int i;
   for (i = n / 2 - 1; i >= 0; i--)
      heapify(arr, n, i);
   for (i=n-1; i>=0; i--)
   {
      swap(&arr[0], &arr[i]);
      heapify(arr, i, 0);
   }
}

void main()
{
            int arr[50];
            int i, key,pos,n;
            printf("enter array size:");
            scanf("%d", &n);
            printf("enter array:\n");
            for(i=0;i<n;i++)
            {
                        scanf("%d",&arr[i]);

            }
            heap_sort(arr, n);
```

```
        printf("Sorted array: \n");
        for(i=0;i<n;i++)
        {
                printf("%d, ",arr[i]);

        }
}
```

9.IMPLEMENTATION OF BFS SHORTEST PATH:

```
#include<stdio.h>

int search(int arr[],int n,int key)
{
   int i;
   for(i=0;i<n;i++)
   if(arr[i]==key)
   return 0;
   //return 1;
}

int indexof(int a[],int n,int key1)
{
   int i;
   for(i=0;i<n;i++)
   if(a[i]==key1)
   return i;
   //return 0;
}
int main()
{
   int i,j,n,k=0,l=0,k1=0,flag=0,arr[10][10],p=0,start,end,root[10],bfs[10],z=-1,in=0,re[10],indo=0;
   scanf("%d",&n);
   for(i=0;i<n;i++)
   for(j=0;j<n;j++)
   scanf("%d",&arr[i][j]);
   scanf("%d",&start);
   scanf("%d",&end);

   bfs[k++] = start;
   root[k1++] = start;
   l++;

   while(k<n){
      flag = 0;
      for(i=0;i<n;i++)
```

```
        if(arr[bfs[l-1]][i]==1 && search(bfs,n,i)){
            bfs[k++] = i;
            flag = 1;
            root[k1++] = bfs[l-1];
        }
        l++;
        if(search(bfs,n,end)==0)
        break;
    }
    for(i=0;i<k;i++)
    printf("%d ",bfs[i]);
    printf("\n");
    for(i=0;i<k1;i++)
    printf("%d ",root[i]);
    printf("\n");
    z = end;
    p = k-1;
    while(z!=start && in<n){
        re[indo++] = z;
        p = indexof(bfs,n,root[p-1]);
        z = bfs[p];

        ++in;
    }
    //printf("%d",start);
    re[indo++] = start;
    printf("path length = %d",indo-1);
    printf("\npath:\n");
    for(j=indo-1;j>=0;j--)
    printf("%d ",re[j]);
    return 0;
}
```

10. IMPLEMENTATION OF DFS:

```
#include<stdio.h>
#include<conio.h>
#define max 20
int g[max][max],v[max],n;
void DFS(int);
int main()
{
int i,u,v,e,s;
printf("Enter No of Edges : ");
scanf("%d",&e);
printf("Enter No of Nodes : ");
scanf("%d",&n);
```

```
for(i=1;i<=e;i++)
{
printf("Enter Edge %d : ",i);
scanf("%d%d",&u,&v);
g[u][v]=g[v][u]=1;
}
printf("Enter Source : ");
scanf("%d",&s);
DFS(s);
getch();
}
void DFS(int s)
{
int i;
printf(" %d",s);
v[s]=1;
for(i=1;i<=n;i++)
{
if(g[s][i]!=0&&v[i]==0)
DFS(i);
}
}
```

11. IMPLEMENTATION OF PRIM'S ALGORITHM:

```
#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;

	//create cost[][] matrix,spanning[][]
	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;
		}

	//initialise visited[],distance[] and from[]
	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;			//cost of spanning tree
	no_of_edges=n-1;		//no. of edges to be added

	while(no_of_edges>0)
	{
		//find the vertex at minimum distance from the tree
```

```
                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];

                //insert the edge in spanning tree
                spanning[u][v]=distance[v];
                spanning[v][u]=distance[v];
                no_of_edges--;
                visited[v]=1;

                //updated the distance[] array
                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);
}
```

12: IMPLEMENTATION OF KRUSKAL'S ALGORITHM:

```
#include<stdio.h>
#include<stdlib.h>
#define VAL 999
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
// union - find
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;
}
int main()
{
    printf("Implementation of Kruskal's algorithm\n");
    printf("Enter the no. of vertices:");
    scanf("%d",&n);
    printf("Enter 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]=VAL;
       }
    }
    printf("The edges of Minimum Cost Spanning Tree are\n");
    while(ne < n)
    {
       for(i=1,min=VAL;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))
       {
          // printing edges
          printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
          mincost +=min;
       }
       cost[a][b]=cost[b][a]=999;
    }
    // minimum cost
```

```
    printf("\n\tMinimum cost = %d\n",mincost);
    return 0;
}
```

13: IMPLEMENTATION OF DIJKSTRA'S ALGORITHM:

```
#include<stdio.h>
#define INFINITY 9999
#define MAX 10

void dijikstra(int G[MAX][MAX], int n, int startnode,int endnode);

void main(){
        int G[MAX][MAX], i, j, n, u,v;

        printf("\nEnter the no. of vertices:: ");
        scanf("%d", &n);
        printf("\nEnter the adjacency matrix::\n");
        for(i=1;i <=n;i++)
                for(j=1;j <=n;j++)
                        scanf("%d", &G[i][j]);
        printf("\nEnter the starting node & ending node:: ");
        scanf("%d%d", &u,&v);
        dijikstra(G,n,u,v);
        getch();
}

void dijikstra(int G[MAX][MAX], int n, int startnode,int endnode)
{
        int cost[MAX][MAX], distance[MAX], pred[MAX];
        int visited[MAX], count, min_distance, nextnode, i,j;
        for(i=1;i <=n;i++)
                for(j=1;j <=n;j++)
                        if(G[i][j]==0)
                                cost[i][j]=INFINITY;
                        else
                                cost[i][j]=G[i][j];

        for(i=1;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) || (visited[endnode]!=1)){
                min_distance=INFINITY;
                for(i=1;i <=n;i++)
                        if((distance[i] < min_distance)&&(!visited[i]))
                        {
                                min_distance=distance[i];
                                nextnode=i;
                        }
                visited[nextnode]=1;
                for(i=1;i <=n;i++)
                        if(!visited[i])
                                if(min_distance+cost[nextnode][i] < distance[i])
                                {
                                        distance[i]=min_distance+cost[nextnode][i];
                                        pred[i]=nextnode;
                                }
                        count++;
        }
        i--;
                        printf("\nDistance of %d = %d", i, distance[endnode]);
                        printf("\nPath = %d", i);
                        j=endnode;
                        while(j!=startnode)
                        {
                                j=pred[j];
                                printf(" <-%d", j);
                        }

}
```

14: IMPLEMENTATION OF BELLMAN FORD'S ALGORITHM:

```
#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;
}
```

15: IMPLEMENTATION OF FLOYD WAESHALL'S ALGORITHM:

```
#include<stdio.h>

int min(int,int);
void floyds(int p[10][10],int n) {
              int i,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(int a,int b) {
              if(a<b)
                return(a); else
                return(b);
}
void 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]);
              }
```

}

16: IMPLEMENTATION OF 0-1 KNAPSACK PROBLEM:

```
#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,k;
  int K[n+1][W+1],a[n];

  for(i=0;i<n;i++)
  a[i]=0;

  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];
     }
  }


 i=n; k=W;


  while(k>=0 && i>=0)
  {
  if((K[i][k]!=K[i-1][k]))
   {
               i=i-1 ; k= k-wt[i];
               a[i]=1;
              }
              else
              {

              i=i-1;
              }
```

```
  }
  printf("\n vector for selected item is:");
   for(i=0;i<n;i++)
   printf("%d ",a[i]);


  return K[n][W];
}

int main()
{
   int i, n, val[20], wt[20], W,profit;

   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);
   profit=knapSack(W, wt, val, n);

   printf("\n maximum profit is: %d", profit );
   return 0;
}
```

17. IMPLEMENTATION OF FRACTIONAL KNAPSACK PROBLEM:

```
#include<stdio.h>
#include<conio.h>
float p[10],w[10],x[10];
void KnapSack(float m,float n);
int main()
{
int n,i;
float m;
printf("Enter no of products : ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Product %d : \n",i+1);
printf("Enter Profit : ");
scanf("%f",&p[i]);
printf("Enter Weight : ");
```

```
scanf("%f",&w[i]);
}
printf("Enter Max Load : ");
scanf("%f",&m);
KnapSack(m,n);
getch();
}
void KnapSack(float m,float n)
{
int j;
float u,c;
for(j=0;j<n;j++)
{
x[j]=0;
}
u=m;
for(j=0;j<n;j++)
{
if(w[j]>u)
{
break;
}
else
{
x[j]=1;
c+=x[j]*p[j];
u=u-w[j];
}
}
if(j<n)
{
x[j]=u/w[j];
c+=x[j]*p[j];
}
printf("Product\tQuantity\tProfit\n\n");
for(j=0;j<n;j++)
{
printf("%d\t%f\t%f\n",j+1,x[j],p[j]*x[j]);
}
printf("Total Cost : %f",c);
}
```

18: IMPLEMENTATION OF JOB SEQUENCING WITH DEADLINE PROBLEM:

```
#include<stdio.h>
int time_slot[50];
int job_id[50];
int total_profit=0;



void job_sequencing(int a[], int b[], int n)
{
	int i,j,time,temp;
	for(i=0;i<n;i++)
	{
		time_slot[i]=0;
	}
	for(i=0;i<n;i++)
	{
		job_id[i]=i+1;
	}
	for (i = 0; i < n-1; i++)
	{
		for (j = 0; j < n-i-1; j++)
		{
			if (a[j] < a[j+1])
			{
				temp = a[j];
				a[j] = a[j+1];
				a[j+1]=temp;
				temp = b[j];
				b[j] = b[j+1];
				b[j+1]=temp;
				temp = job_id[j];
				job_id[j] = job_id[j+1];
				job_id[j+1]=temp;
			}
		}
	}

	for(i=0;i<n;i++)
```

```
	{
		time = b[i];
		if(time_slot[time-1] == 0)
		{
			time_slot[time-1]=job_id[i];
			total_profit = total_profit+a[i];
			//printf("\njob at time slot %d = %d", time, time_slot[time-1]);
		}
		else if(time_slot[time-1] > 0)
		{
			 for(j=time-2;j>=0;j--)
			 {
				if(time_slot[j]==0)
				{
					time_slot[j]=job_id[i];
					//printf("\njob at time slot %d = %d", j+1, time_slot[j]);
					total_profit = total_profit+a[i];
					break;
				}

			 }
		}

	}
	printf("\nthe jobs are:\n");
	int max_d = 0;
	for(i=0;i<n;i++)
	{
		if(b[i]>max_d)
			max_d = b[i];
	}
	for(i=0;i<max_d;i++)
	{
		printf("%d ", time_slot[i]);
	}
	printf("\ntotal profit = %d", total_profit);
}

void main()
{
	int n,i,j;
	int profit[50];
```

```
        int deadline[50];
        printf("\nenter no of jobs=");
        scanf("%d", &n);
        printf("\nenter the job details:\n");
        for(i=0;i<n;i++)
        {
                printf("\nenter the profit for job%d=",i+1);
                scanf("%d",&profit[i]);
                printf("\nenter the deadline for job%d=",i+1);
                scanf("%d",&deadline[i]);
        }
        job_sequencing(profit, deadline, n);


}
```

19: IMPLEMENTATION OF STRASSEN'S MATRIX MULTIPLICATION:

```
#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 elements of first matrix: ");
  for(i = 0;i < 2; i++)
     for(j = 0;j < 2; j++)
         scanf("%d", &a[i][j]);

  printf("Enter the 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 Strassen's algorithm \n");
  for(i = 0; i < 2 ; i++){
    printf("\n");
    for(j = 0;j < 2; j++)
        printf("%d\t", c[i][j]);
  }

  return 0;
}
```

20: IMPLEMENTATION OF CHAIN MATRIX MULTIPLICATION:

```
#include<stdio.h>

/* Matrix Ai has dimension dim[i-1] x dim[i] for i = 1..n
i.e. dim is an array containing the dimensions of the chain matrices*/
int MatrixChainOrder(int dim[], int i, int j)
{
        if(i == j)
                return 0;
        int k;
        int min = 99999;
```

```
	int count;

	/*place parenthesis at different places between first and last matrix, recursively calculate count
of
	multiplications for each parenthesis placement and return the minimum count */
	for (k = i; k<j; k++)
	{
		count = MatrixChainOrder(dim, i, k) + MatrixChainOrder(dim, k+1, j) + dim[i-
1]*dim[k]*dim[j];
		printf("\ncount=%d",count);

		if (count < min)
			min = count;
	}

	// Return minimum count
	return min;
}

void main()
{
	int n,i;
	printf("enter the no of matrices=");
	scanf("%d", &n);
	int dim[50];
	printf("enter the corresponding dimensions:\n");
	for(i=0;i<=n;i++)
	{
		scanf("%d", &dim[i]);

	}

	 int mult_no=MatrixChainOrder(dim, 1, n);

	printf("Minimum number of multiplications is %d ", mult_no);

	getch();
}
```

21: IMPLEMENTATION OF TRAVELLING SALAESMAN PROBLEM:

```
#include<stdio.h>
#include<conio.h>
int adj[10][10],visited[10],n,cost=0;


void minimum_cost(int city)
{
	int i,ncity;
	visited[city]=1;
	printf("%d -->",city+1);
	ncity=least(city);
	if(ncity==999)
	{
		ncity=0;
		printf("%d",ncity+1);
		cost+=adj[city][ncity];
		return;
	}
	minimum_cost(ncity);
}

int least(int c)
{
	int i,nc=999;
	int min=999,kmin;
	for(i=0;i<n;i++)
	{
		if((adj[c][i]!=0)&&(visited[i]==0))
			if(adj[c][i] < min)
			{
				min=adj[i][0]+adj[c][i];
				kmin=adj[c][i];
				nc=i;
			}
	}
	if(min!=999)
		cost+=kmin;
	return nc;
}
```

```
void main()
{
	int i,j;
	printf("enter the no of cities=");
	scanf("%d", &n);
	printf("enter the cost matrix:\n");
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
		{
			scanf("%d",&adj[i][j]);
		}
		visited[i]=0;
	}
	printf("\n\nThe cost matrix is:\n\n");
	for( i=0;i<n;i++)
	{
		printf("\n");
		for(j=0;j<n;j++)
			printf("%d ",adj[i][j]);
	}
	printf("\n\nThe Path is:\n\n");
	minimum_cost(0);
	printf("\n\nMinimum cost=%d",cost);
	getch();
}
```

22: IMPLEMENTATION OF LONGEST COMMON SUBSEQUENCE PROBLEM:

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

int max(int a, int b);

/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs( char *X, char *Y, int m, int n )
{
if (m == 0 || n == 0)
	return 0;
if (X[m-1] == Y[n-1])
```

```
        return 1 + lcs(X, Y, m-1, n-1);
else
        return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));
}

/* Utility function to get max of 2 integers */
int max(int a, int b)
{
        return (a > b)? a : b;
}

/* Driver program to test above function */
int main()
{
char X[20],Y[20];

printf("enter 1st string = ");
scanf("%s", &X);
printf("enter 2nd string = ");
scanf("%s", &Y);
int m = strlen(X);
int n = strlen(Y);

printf("Length of LCS is %d", lcs( X, Y, m, n ) );

return 0;
}
```

23: IMPLEMENTATION OF MIN-MAX PROBLEM:

```
#include<stdio.h>
#include<conio.h>
int max=0,min=0,a[10];
void MaxMin(int,int);
int main()
{
int i,n;
printf("Enter No of Elements ( >0 ) : ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter A[%d] : ",i+1);
scanf("%d",&a[i]);
```

```
}
MaxMin(0,n-1);
printf("Max : %d\nMin : %d",max,min);
getch();
}
void MaxMin(int i,int j)
{
int max1,min1,m;
if(i==j)
max=min=a[i];
else if(i==j-1)
{
if(a[i]<a[j])
{
max=a[j];
min=a[i];
}
else
{
max=a[i];
min=a[j];
}
}
else
{
m=(i+j)/2;
MaxMin(i,m);
max1=max;
min1=min;
MaxMin(m,j);
if(max1>max)
max=max1;
if(min1<min)
min=min1;
}
```

24: IMPLEMENTATION OF GRAPH COLORING PROBLEM:

```
#include<stdio.h>
#include<conio.h>
int x[10],m,n,g[10][10],c=1;
void mcolor(int k);
void Nextcolor(int k);
void display(void);
int main()
{
int i,j,s,d,e;
```

```
printf("Enter the no of Edges : ");
scanf("%d",&e);
printf("Enter the no of Vertices : ");
scanf("%d",&n);
printf("Enter the no of Colors : ");
scanf("%d",&m);
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
g[i][j]=0;
}
for(i=1;i<=e;i++)
{
printf("Enter Edge %d : \n",i);
scanf("%d%d",&s,&d);
g[s][d]=g[d][s]=1;
}
mcolor(1);
getch();
}
void mcolor(int k)
{
while(1)
{
Nextcolor(k);
if(x[k]==0)
return;
else if(k==n)
display();
else
mcolor(k+1);
}
}
void Nextcolor(int k)
{
int i;
while(1)
{
x[k]=(x[k]+1)%(m+1);
if(x[k]==0)
return;
else
{
for(i=1;i<=n;i++)
{
if(g[i][k]==1&&x[i]==x[k])
break;
}
```

```
}
if(i==n+1)
return;
}
}
void display(void)
{
int i;
printf("\nSolution : %d\n",c++);
for(i=1;i<=n;i++)
printf("%d : %d\n",i,x[i]);
}
```

25: IMPLEMENTATION OF N-QUEEN PROBLEM:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int n,x[10],c=1;
void Nqueen(int k);
int Place(int k,int i);
void display(void);
int main()
{
int i;
printf("Enter No of Queens : ");
scanf("%d",&n);
printf("\n");
Nqueen(0);
getch();
}
void Nqueen(int k)
{
int i;
for(i=0;i<n;i++)
{
if(Place(k,i))
{
x[k]=i;
if(k==n-1)
display();
else
Nqueen(k+1);
}
}
}
```

```
int Place(int k,int i)
{
int j;
for(j=0;j<=k-1;j++)
{
if((x[j]==i)||(fabs(x[j]-i)==fabs(j-k)))
return 0;
}
return 1;
}
void display(void)
{
int i,j;
printf("\nSolution %d\n",c++);
for(i=0;i<n;i++)
{
printf("----------------\n");
for(j=0;j<n;j++)
{
if(j==x[i])
printf("| Q ");
else
printf("| ");
}
printf("|\n");
}
printf("----------------\n");
}
```