1 Write a program to sort a list of N elements using Selection Sort Technique

```
#include<stdio.h>
void main()
{
int i,j,n,pos,temp,a[40];
printf(“Enter the limit:\n”);
scanf(“%d”,&n);
printf(“Enter the elements:\n”);
for(i=0;i<n;i++)
scanf(“%d”,&a[i]);
for(i=0;i<n-1;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(a[j]< a[pos])
{
pos=j;
}
}
temp=a[pos];
a[pos]=a[i];
a[i]=temp;
}
printf(“Sorted array is:\n”);
for(i=0;i<n;i++)
printf(“%d\t”,a[i]);
}
```

2 Write a program to perform Travelling Sales man Problem

```
#include<stdio.h>
int ary[10][10],completed[10],n,cost=0;
void takeInput()
{
  int i,j;
  printf("Enter the number of cities: ");
  scanf("%d",&n);
  printf("\nEnter the 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", &ary[i][j]);
    completed[i]=0;
  }
  printf("\n\nThe cost list is:");
  for( i=0;i < n;i++)
  {
    printf("\n");
    for(j=0;j < n;j++)
    printf("\t%d",ary[i][j]);
  }
}
void mincost(int city)
{
   int i, ncity;
   completed[city]=1;
   printf("%d--->",city+1);
   ncity=least(city);
   if(ncity==999)
   {
      ncity=0;
      printf("%d",ncity+1);
      cost+=ary[city][ncity];
      return;
   }
  mincost(ncity);
 }
int least(int c)
{
    int i,nc=999;
    int min=999,kmin;
    for(i=0;i < n;i++)
    {
       if((ary[c][i]!=0)&&(completed[i]==0))
if(ary[c][i]+ary[i][c] < min)
      {
     min=ary[i][0]+ary[c][i];
         kmin=ary[c][i];
         nc=i;
      }
  }
  if(min!=999)
  cost+=kmin;
  return nc;
}
int main()
{
  takeInput();
  printf("\n\nThe Path is:\n");
  mincost(0);
  printf("\n\nMinimum cost is %d\n ",cost);
  return 0;
}
#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;
}
```

4 Write program to implement the DFS and BFS algorithm for a graph.

```
   #include<stdio.h>
  #define MAX 5

  void bfs(int a[][MAX],int v[],int s)
  {
  int queue[MAX],rear=-1,front=-1,i,k;
  for(k=0;k<MAX;k++)
  v[k]=0;
queue[++rear]=s;
  ++front;
  v[s]=1;
while(rear>=front)
  {
  s=queue[front++];
  printf("%c-",s + 65);
  for(i=0; i<MAX; i++)
  {
  if(a[s][i] && v[i] == 0)
  {
  queue[++rear] = i;
  v[i] = 1;
  }
  }
  }
  }
  void dfs(int a[][MAX],int v[],int s)
  {
  int stack[MAX];
  int top=-1,i,k;
  for(k=0;k<MAX;k++)
  v[k]=0;
  stack[++top]=s;
  v[s]=1;
while(top!=-1)
{
  s=stack[top--];
  printf("%c-",s + 65);
  for(i=0;i<MAX;i++)
  {
  if(a[s][i] && v[i] == 0)
  {
  s[++top]=i;
  v[i]=1;
  break;
  }
  }
  }
  }
  int main()
  {
  int v[MAX]={0};
  int a[MAX][MAX],i,j;
  int option,size;
  do
  {
  printf(" 1. enter values in the graph");
  printf(" 2. BFS ");
  printf(" 3.dfs");
  printf(" 4. exit");
  printf(" enter your option");
  scanf("%d",&option);
  switch(option)
  {
```

```
case 1:printf(" enter matrix");
for(i=0; i<MAX; i++)
for(j=0; j<MAX; j++)
scanf("%d",&a[i][j]);
break;
case 2:printf("bfs");
bfs(a,v,0);
break;
case 3:printf("dfs");
dfs(a,v,0);
break;
}
}
while(option!=4);
return(0);
}
```

. 5 Write a program to find minimum and maximum value in an array using divide and conquer.

```
#include<stdio.h>
#include<stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j)
{
  int max1, min1, mid;
  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
    {
      mid = (i+j)/2;
      maxmin(i, mid);
      max1 = max; min1 = min;
      maxmin(mid+1, j);
      if(max <max1)
      max = max1;
      if(min > min1)
      min = min1;
    }
    }
}
int main ()
{
    int i, num;
    printf ("\nEnter the total number of numbers : ");
    scanf ("%d",&num);
    printf ("Enter the numbers : \n");
    for (i=1;i<=num;i++)
    scanf ("%d",&a[i]);

    max = a[0];
    min = a[0];
    maxmin(1, num);
    printf ("Minimum element in an array : %d\n", min);
    printf ("Maximum element in an array : %d\n", max);
    return 0;
}
```

6 Write a test program to implement Divide and Conquer Strategy. Eg: Quick sort algorithm for sorting list of integers in ascending order.

```
#include<stdio.h>
void quicksort(int number[25], int first, int last)
{
  int i, j, pivot, temp;
  if(first<last)
  {
    pivot=first;
    i=first;
    j=last;
    while(i<j)
    {
 while(number[i]<=number[pivot]&&i<last)
      i++;      while(number[j]>number[pivot])
      j--;
      if(i<j)
      {
        temp = number[i];
        number[i] = number[j];
        number[j] = temp;
      }
    }
    temp = number[pivot];
    number[pivot] = number[j];
    number[j] = temp;
    quicksort(number, first ,j-1);
    quicksort(number, j+1, last);
  }
}
```

```
int main()
{
  int i, count, number[25];
  printf("Enter the number of elements:\n ");
  scanf("%d", &count);
  printf("Enter %d elements: ", count);
  for(i=0; i<count; i++)
  scanf("%d", &number[i]);
 quicksort(number,0,count-1);
  printf("Order of Sorted elements: ");
  for(i=0; i<count; i++)
  printf(" %d", number[i]);
  return 0;
}
```

7 Write a program to implement Merge sort algorithm for sorting a list of integers in ascending order.

```
#include <stdio.h>
#include <stdlib.h>
void Merge(int arr[], int left, int mid, int right)
{
   int i, j, k;
   int size1 = mid - left + 1;
   int size2 = right - mid;
   int Left[size1], Right[size2];
      for (i = 0; i < size1; i++)
      Left[i] = arr[left + i];
   for (j = 0; j < size2; j++)
      Right[j] = arr[mid + 1 + j];
   i = 0;
   j = 0;
```

```
    k = left;
    while (i < size1 && j < size2)
    {
        if (Left[i] <= Right[j])
        {
            arr[k] = Left[i];
            i++;
        }
        else
        {
            arr[k] = Right[j];
            j++;
        }
        k++;
    }
     while (i < size1)
    {
        arr[k] = Left[i];
        i++;
        k++;
    }
    while (j < size2)
    {
        arr[k] = Right[j];
        j++;
        k++;
    }
}
void Merge_Sort(int arr[], int left, int right)
{
    if (left < right)
    {
        int mid = left + (right - left) / 2;
        Merge_Sort(arr, left, mid);
        Merge_Sort(arr, mid + 1, right);
        Merge(arr, left, mid, right);
    }
}
int main()
{
    int size;
    printf("Enter the size: ");
    scanf("%d", &size);
    int arr[size];
    printf("Enter the elements of array: ");
    for (int i = 0; i < size; i++)
    {
        scanf("%d", &arr[i]);
    }
    Merge_Sort(arr, 0, size - 1);
    printf("The sorted array is: ");
    for (int i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

8 Write C program that accepts the vertices and edges for a graph and stores it as an adjacency matrix.

```
#include <stdio.h>

#define MAX_VERTICES 100

int main() {

int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0}; // Initialize the adjacency matrix with zeros

int numVertices, numEdges;

int i, j, u, v;

printf("Enter the number of vertices in the graph: ");

scanf("%d", &numVertices);

printf("Enter the number of edges in the graph: ");

scanf("%d", &numEdges);

// Accept the edges from the user and store them in the adjacency matrix

printf("Enter the edges (u, v):\n");

for (i = 0; i < numEdges; i++) {

scanf("%d %d", &u, &v);

adjMatrix[u][v] = 1;

adjMatrix[v][u] = 1; // If the graph is undirected, set both vertices as adjacent

}

// Display the adjacency matrix

printf("\nAdjacency Matrix:\n");

for (i = 0; i < numVertices; i++) {

for (j = 0; j < numVertices; j++) {
```

```
printf("%d ", adjMatrix[i][j]);

}

printf("\n");

}

}
```

9 ImplementfunctiontoprintIn-Degree,Out-Degree and to display that adjacency matrix

```
#include<stdio.h>
#define MAX 10
void accept_graph(int G[][MAX], int n)
{
int i,j;
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("Edge (V%d,V%d) exists? (Yes=1, No=0):",i,j);
scanf("%d",&G[i][j]);
}
}
}
void disp_adj_mat(int G[][MAX], int n)
{
int i,j;
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("%4d",G[i][j]);
}
printf("\n");
}
}
void calc_out_degree(int G[][MAX], int n)
{
int i,j,sum;
for(i=0;i<n;i++)
{
sum=0;
for(j=0;j<n;j++)
{
sum += G[i][j];
}
printf("out-deg(V%d)=%d\n",i,sum);
}
}
void calc_in_degree(int G[][MAX], int n)
{
int i,j,sum;
for(i=0;i<n;i++)
{
sum=0;
for(j=0;j<n;j++)
{
sum += G[j][i];
}
printf("in-deg(V%d)=%d\n",i,sum);
}
}
void main()
{
int G[MAX][MAX],n;
printf("Enter no.of vertices:");
scanf("%d",&n);
accept_graph(G,n);
printf("Adjacency Matrix:\n");
disp_adj_mat(G,n);
printf("Out degree:\n");
calc_out_degree(G,n);
printf("In degree:\n");
calc_in_degree(G,n);
}
```

10. Write a program to perform Knapsack Problem using Greedy Solution

```
#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 = Totalvalue + profit[i];
        capacity = capacity - weight[i];
      }
    }
if (i < n)
      Totalvalue = Totalvalue + (ratio[i]*capacity);
    printf("\nThe maximum value is :%f\n",Totalvalue);
    return 0;
}
```

11 program

```
#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)
    {
      if(board[i]==j)
      printf("\tQ");
      else
      printf("\t-");
   }
 }
}
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;
}
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);
     }
  }
}
```

12. Write a program to implement the backtracking algorithm for the sum of subsets problem

```
#include<stdio.h>
#define TRUE 1
#define FALSE 0
int inc[50],w[50],sum,n;
void sumset(int i, int wt, int total);
int promising(int i,int wt,int total) {
        return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));
}
void main() {
        int i,j,n,temp,total=0;

        printf("\n Enter how many numbers:\n");
        scanf("%d",&n);
        printf("\n Enter %d numbers to th set:\n",n);
        for (i=0;i<n;i++) {
                scanf("%d",&w[i]);
                total+=w[i];
        }
    printf("\n Input the sum value to create sub set:\n");
        scanf("%d",&sum);
        for (i=0;i<=n;i++)
          for (j=0;j<n-1;j++)
           if(w[j]>w[j+1]) {
                temp=w[j];
                w[j]=w[j+1];
                w[j+1]=temp;
        }
        printf("\n The given %d numbers in ascending order:\n",n);
        for (i=0;i<n;i++)
          printf("%d \t",w[i]);
        if((total<sum))
          printf("\n Subset construction is not possible"); else {
                for (i=0;i<n;i++)
                  inc[i]=0;
                printf("\n The solution using backtracking is:\n");
                sumset(-1,0,total);
        }
}
void sumset(int i,int wt,int total) {
        int j;
      if(promising(i,wt,total)) {
                if(wt==sum) {
                     printf("\n{\t");
                       for (j=0;j<=i;j++)
                          if(inc[j])
                           printf("%d\t",w[j]);
                       printf("}\n");
                } else {
                       inc[i+1]=TRUE;
                     sumset(i+1,wt+w[i+1],total-w[i+1]);
                       inc[i+1]=FALSE;
                    sumset(i+1,wt,total-w[i+1]);
                }
        }
}
```

13 Write program to implement greedy algorithm for job sequencing with deadlines.

```
#include<stdio.h>
int n,i,j,k,t;
int check(int s[],int p)
       {  int ptr=0,i;
             for(i=0;i<n;i++)
             {if(s[i]==p)
                ptr++;
      }
              if(ptr==0)
              return 1;
              else
              return 0;
          }

void main()
```

```
{
        int slot[10],profit,p[10],d[10],max=0;

   printf("enter the no of jobs      : ");
   scanf("%d",&n);

   for(i=0;i<n;i++)
     {printf("\n enter the profit of job #%d      :",i+1);
     scanf("%d",&p[i]);
     printf("\n enter the deadline of job #%d    :",i+1);
     scanf("%d",&d[i]);
     }

   for(i=0;i<n;i++)
        for(j=i+1;j<n;j++)
         if(p[i]<p[j])
            { t=p[i];
              p[i]=p[j];
              p[j]=t;
              t=d[i];
              d[i]=d[j];
              d[j]=t;
            }

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

   for(i=0;i<n;i++)
         for(j=d[i];j>0;j--)
            {if(check(slot,j)==1)
                 {slot[i]=j;
                 break;}
            }

   printf("\n\n INDEX   PROFIT  DEADLINE  SLOT ALLOTTED ");
   for(i=0;i<n;i++)
   {if(slot[i]>0)
     {
           printf("\n\n   %d       %d        %d        [%d - %d]", i+1,p[i],d[i],(slot[i]-1),slot[i]);
           max=max+p[i];
        }
   else
   printf("\n\n   %d       %d        %d       REJECTED", i+1,p[i],d[i]);
   }

   printf("Total profit=%d",max);
}
```

14 Write program to implement Dynamic Programming algorithm for the Optimal Binary Search Tree Problem

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

// Function prototype
int sum(int freq[], int i, int j);

// Function to find the optimal binary search tree cost using dynamic programming
int optimalBST(int keys[], int freq[], int n) {
    int cost[n][n];

    for (int i = 0; i < n; i++) {
        cost[i][i] = freq[i];
    }

    for (int len = 2; len <= n; len++) {
        for (int i = 0; i <= n - len + 1; i++) {
            int j = i + len - 1;
            cost[i][j] = INT_MAX;

            for (int r = i; r <= j; r++) {
                int c = ((r > i) ? cost[i][r - 1] : 0) +
                        ((r < j) ? cost[r + 1][j] : 0) + sum(freq, i, j);

                if (c < cost[i][j]) {
                    cost[i][j] = c;
                }
            }
        }
    }

    return cost[0][n - 1];
}

// Function to calculate the sum of frequencies between indices i and j
int sum(int freq[], int i, int j) {
    int s = 0;
    for (int k = i; k <= j; k++) {
        s += freq[k];
    }
    return s;
}

int main() {

int n;

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

    int keys[n];
    int freq[n];

    printf("Enter the keys:\n");
    for (int i = 0; i < n; i++) {
        printf("Key %d: ", i + 1);
        scanf("%d", &keys[i]);
    }

    printf("Enter the frequencies:\n");
    for (int i = 0; i < n; i++) {
        printf("Frequency for key %d: ", i + 1);
        scanf("%d", &freq[i]);
    }

    printf("The cost of optimal binary search tree is: %d\n", optimalBST(keys, freq, n));
return 0;
}
```

15. Write a program that implements Prim’s algorithm to generate minimum cost spanning Tree

```
#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("\n Enter the number of nodes:");
    scanf("%d",&n);

    printf("\n Enter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=999;
        }
    visited[1]=1;
    printf("\n");

    while(ne<n)
    {
        for(i=1,min=999;i<=n;i++)
            for(j=1;j<=n;j++)
                if(cost[i][j]<min)
                    if(visited[i]!=0)
                    {               min=cost[i][j];
a=u=i           b=v=j;              }  if(visited[u]==0 || visited[v]==0)
        {           printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min)
mincost+=min;
visited[b]=1;
        }
        cost[a][b]=cost[b][a]=999;
    }
    printf("\n Minimun cost=%d",mincost);
    getch();
}
```

16 Write a program that implements Kruskal’s algorithm to generate minimum cost spanning tree

```
#include<stdio.h>
int find(int v,int s[])
{
    while(s[v]!=v)
         v=s[v];
    return v;
}
void kruskal(int n,int c[10][10])
{
   int count,i,s[10],min,j,u,v,k,t[10][2],sum;
   for(i=0;i<n;i++)
   s[i]=i;
   count=0;
   sum=0;
   k=0;
  while(count<n-1)
  {
   min=9999;
    for(i=0;i<n;i++)
     {
     for(j=0;j<n;j++)
     {
            if(c[i][j]!=0 && c[i][j]<min)
            {
                      min=c[i][j];
                      u=i,v=j;
             }
           }
     }
     if(min==9999)break;
         i=find(u,s);
         j=find(v,s);
         if(u!=v)
         {
            t[k][0]=u;
            t[k][1]=v;
            k++;
            count++;
            sum+=min;
            s[v]=u;
         }
          c[u][v]=c[v][u]=9999;
   }
   if(count==n-1)
   {
         printf("cost of spanning tree=%d\n",sum);
         printf("spanning tree is know below\n");
         for(k=0;k<n-1;k++)
         {
           printf("%d->%d\n",t[k][0],t[k][1]);
         }
      //  exit(0);
    }
   // printf("spanning tree do not exit\n");
}

int main()
{
   int n,c[10][10],i,j;
   printf("enter the number of nodes\n");
   scanf("%d",&n);
   printf("ent the cost matrix\n");
   for(i=0;i<n;i++)
   {
     for(j=0;j<n;j++)
     {
         scanf("%d",&c[i][j]);
     }
   }
   kruskal(n,c);
      return 0;
}
```