Write a program to sort a list of N element using selection sort

```
#include<stdio.h>
void selection-sort(int a[],int n)
{
    int i,j, temp, pos=0;
    for(i=0; i<n-1;i++)
    {
        pos=i;
        for(j=i+1; j<n-1; j++)
        {
            if(a[pos]>a[j])
            pos=j;
        }
       temp=a[i];
       a[j]=a[pos];
       a[pos]=temp;
    }
}
void main()
{
   int a[10], n, i;
   printf("Enter the size of the array");
   scanf("%d", &n);
   printf("Enter array elements");
   for(i=0; i<n; i++)
   scanf("%d", &a[i])
   printf("Sorted array elements are");
   selection_sort(a,n)
   for(i=0;i<n;i++)
   printf("%d", a[i])
```

Write a program to perform travelling salesman problem

```
#include<stdio.h>
#include<stdlib.h>
#define N 4
int city[N][N];
int mincost=9999;
int bestPath[N];
int calculateCost(int path[])
{
   int i;
   int cost=0;
   for(i=0; i<n-1; i++)
   {
      cost+=city[path[i]][path[i+1]];
   }
   cost+=city[path[N-1]][path[0]];
   return cost;
}
void Generate Permutations(int path[], int start,
int end){
   int tmp, i;
   if(start==end)
   {
      int cost=calculateCost(path);
      if(cost<minCost)
      {
         mincost=cost;
         for(i=0; i<N; i++)
         {
            bestpath[i]=path[i];
```

```
    }
  }
}
else
{
  for(i=start; i<=end; i++)
  {
    temp=path[start];
    path[start]=path[i];
    path[i]=temp;
    GeneratePermutations(path, start+1, end);
    temp=path[start];
    path[start]=path[i];
    path[i]=temp;
  }
}
}
int main()
{
  int path[N], i, j;
  printf("Enter the cost matrix:\n");
  for(i=0; i<N; i++)
  {
    for(j=0; j<N; j++)
    {
      scanf("%d", &city[i][j]);
    }
  }
for(i=0; i<N; i++)
{
    path[i]=i;
```

```
    }
    GenerateePermutations(path, 1, N-1);
    printf("minimum cost: %d\n", minCost);
    printf("Best path: ");
    for (i=0; i<N; i++)
    {
        printf("%d", bestpath[i]);
    }
    printf("%d\n", bestpath[0]); //Return to starting point
//city
    return 0;
}
```

Write program to implement dynamic programming algorithm for 0/1 knapsack problem

```
#include<stdio.h>
int max(int a, int b)
{
   if(a>b)
     return a;
   else
     return b;
}
void knapsack(int n, int m , int w[10], int p[10])
{
   int i,j, v[11][11];
   for(i=0; i<=n; i++)
   {
      for(j=0; j<=m; j++)
      {
         if(i==0 || j==0)
            v[i][j]=0;
         else if(w[i]>j)
            v[i][j]= v[i-1][j];
         else
            v[i][j]=max(v[i-1][j], v[i-1][j-w[i]]+p[i]);
            //printf("profit=%d\n", v[i][j]);
      }
   }
   printf("maximum profit earned=%d", v[n][m]);
}
int main()
{
```

```
int profit[10], weight[10], m, i, n;
printf("Enter the number of objects");
scanf("%d", &n);
printf("Enter the knapsack capacity");
scanf("%d", &m);
printf("Enter the weights and profits of each objects");
for(i=1; i<=n; i++)
    scanf("%d %d", &weight[i], &profit[i]);
knapsack(n, m, weight, profit);
}
```

C program to implement DFS and BFS

```
#include<stdio.h>
#include<conio.h>
//Depth first search (DFS)
void dfs(int a[10][10], int n, int source, int *visited)
{ int i;
  visited[source]=1;
  printf(" %d", source);
  for(int i=0; i<n; i++)
  {
    if(a[source][i]==1 && !visited[i])
    {
      dfs(a, n, i, visited);
    }
  }
}
//BFS
void bfs(int a[10][10], int n, int source, int *visited)
{ int i;
  int queue[n], front=0, rear=0;
  queue[rear++]=source;
  visited[source]=1;
  while(front<rear)
  { source=queue[front++];
    //test statement
    if(a[source][i]==1 && !visited[i])
    {
      queue[rear++]=i;
      visited[i]=1;
    }
  }
}
```

```
}
void main()
{
   int a[10][10], visited[10]; i, j
   int vertices, edjes, s, d;
   printf("Enter no. of vertices(max 10);");
   scanf("%d", &vertices);
   //Initialize adjecency matrix
   for(int i=0; i<vertices; i++)
   {
      for(j=0; j<vertices; j++)
      {
         a[i][j]=0;
      }
}
printf("Enter number of edges");
scanf("%d", &edges);
printf("Enter edges one by one\n");
for(i=0; i<edges; i++)
{
   scanf("%d%d", &s, &d);
   a[s][d]=1;
   a[d][s]=1;
}
for(i=0; i<vertices; i++)
{
   visited[i]=0;
}
printf("Depth First Search (DFS):");
dfs(a, vertices, 0, visited);
for(i=0; i<vertices; i++)
```

```
{
    visited[i]=0;
}
printf("\n Breadth-First Search (BFS):");
bfs(a, vertices, 0, visited);
getch();
}
```

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

```
#include<stdio.h>
#include<conio.h>
void maxmin(int a[], int i, int j, int *max, int *min)
{
  int mid, min1, max1;
  if(i==j)
  *max=*min=a[j];
  else if (i==j-1)
  {
    if(a[i]>a[j])
    {
      *max=a[i];
      *min=a[j];
    }
  else{
        *max=a[j];
        *min=a[i];
      }
  }
else
 {
    mid=(i+j)/2;
    maxmin(a,i,mid,max,min);
    maxmin(a,mid+1,j,&max1,&min1);
    if(max1>*max)
     *max=max1;
    if(min1<*min)
     *min=min1;
    }
```

```
void main()
{
  int a[10], n, max, min, i;
  printf("Enter size of array\n");
  scanf("%d", &n);
  printf("Enter array element\n");
  for(i=0; i<n; i++)
  scanf("%d", &a[i]);
  maxmin(a, 0, n-1, &max, &min);
  printf("max=%d\n", min=%d\n", max, min);
  getch();
}
```

Write a program to implement divide and conquer strategy.
Eg: Quick sort algorithm for sorting list of integers in ascending order.

```
#include<stdio.h>
#include<conio.h>
int partition(int a[25], int low, int high);
void quick_sort(int a[25], int low, int high);
void main()
{
   int i,n, a[25];
   printf("Enter the size of array\n");
   scanf("%d", &n);
   printf("Enter the array values\n");
   for(i=0; i<n; i++)
   scanf("%d", &a[i]);
   quick_sort(a, 0, (n-n);
   printf("Sorted array is \n");
   for(i=0; i<n; i++)
   printf("%d\t", a[i]);
   getch();
}
void quick_sort(int a[25], int low, int high)
{
   int j;
   if(low<high)
   {
      j=partition(a, low, high);
      quick_sort(a, low, j-1);
      quick_sort(a, j+1, high);
   }
```

```
int partition(int a[25], int low, int high)
{
   int temp, i, j, key;
   key=a[low];
   i=low+1;
   j=high;
   while(1)
   {
      while(i<high && key>=a[i])
        i++;
      while(key<a[j])
        j--;
      if(i<j)
      {
         temp=a[i];
         a[i]=a[j];
         a[j]=temp;
      }
      else
      {
         temp=a[low];
         a[low]=a[j];
         a[j]=temp;
         return j;
      }
   }
}
```

Write a program to implement merge sort algorithm for sorting a list of integer in ascending order.

```
#include<stdio.h>
#include<conio.h>
void merge_sort(int a[20], int low, int high);
void simple_merge(int a[20], int low, int mid, int high);
void main()
{
  int a[20], n, i;
  printf("Enter size of array\n");
  scanf("%d", &n);
  printf("Enta array element\n");
  for(i=0; i<n; i++)
  scanf("%d", &a[i]);
  merge_sort(a,0,n-1);
  printf("sorted array is:\n");
  for(i=0; i<n; i++)
  printf("%d\n", a[i]);
  getch();
}
void simple_merge(int a[20], int low, int mid, int high)
{
  int c[20], i, j, k;
  i=low;
  j=mid+1;
  k=low;
  while((i<=mid) && (j<=high))
  {
    if(a[i]<a[j])
    {
      a[k++]=a[j++];
```

```
        else
        {
            c[k++] = a[j++];
        }
    }
}
while(i<=mid)
{
    c[k++] = a[i++];
}
while(j<=high)
{
    c[k++] = a[j++];
}
for(i=low; i<=high; i++)
{
    a[i] = c[i];
}
}
void merge_sort(int a[20], int low, int high)
{
    int mid;
    if(low<high)
    {
        mid = (low+high)/2;
        merge_sort(a, low, mid);
        merge_sort(a, mid+1, high);
        merge_sort(a, low, mid, high);
    }
}
```

C program that accepts vertices and edges of a graph and stores it as an adjecency matrix

```
#include<stdio.h>
#define MAX 10
int main()
{
   int a[MAX][MAX], v, e, s, d;
   print("Enter number of vertices");
   scanf("%d", &v);
   for(i=0; i<v; i++)
   {
      for(j=0; j<v; j++)
      {
         a[i][j]=0;
      }
   }
printf("Enter number of edges");
scanf("%d", &e);
printf("Enter the edges one by one from source to destination");
for(i=0; i<e; i++)
{
   scanf("%d%d", &s, &d);
   a[s][d]=1;
   a[d][s]=1;
}
printf("Adjecency matrix");
for(i=0; i<v; i++)
{
      for(j=0; j<v; j++)
```

```
{
    printf("%d", a[i][j]);
}
printf("\n");
}
```

Implementation of in-degree and out-degree and display the adjecency matrix

```
#include<stdio.h>
int main()
{
 int a[10][10], v, e, s, d, i, j, indegree[10], outdegree[10]
 printf("Enter the number of vertices");
 scanf("%d", &v);
 for(i=0; i<v; i++)
 for(j=0; j<v; j++)
    a[i][j]=0;
 printf("Enter the number of edges");
 scanf("%d", &e);
 printf("Enter edges one by one from source to its
 destination");
 for(i=0; i<e; i++)
 {
    scanf("%d%d", &s, &d);
    a[s][d]=1;
 }
 printf("Adjecency matrix");
 for(i=0; i<v; i++)
 {
    for(j=0; j<v; j++)
    {
       printf("%d\t", a[i][j]);
    }
    printf("\n");
 }
 printf("Vertex \t indeg \t outdegree\n");
```

```
for(i=0; i<V; i++)
{
    indegree[i]=0, outdegree[i]=0;
    for(j=0; j<V; j++)
    {
        if(a[j][i]==1)
            indegree[i]++;
        else if(a[i][j]==1)
            outdegree[i]++;
    }
printf("%d\t%d\t%d\n", i, indegree[i], outdegree[i])
}
}
```

Write a program to perform knapsack problem using greedy solution

```
#include<stdio.h>
void knapsack(int n, float weight[], float
profit[], float capacity)
{
   float x[20], tp=0;
   int i,j, rc;
   rc=capacity;
   for(i=0; i<n; i++)
   x[i]=0.0;
   for(i=0; i<n; i++)
   {
      if (weight[i]<rc)
      {
         x[i]=1.0;
         tp=tp+profit[i];
         rc=rc-weight[i];
      }
      else
      break;
   }
   if(i<n)
   x[i]=rc/weight[i];
   tp=tp+(x[i]*profit[i]);
   printf("\n The result vector is :");
   for(i=0; i<n; i++)
   printf("%.f\t", x[i]);
   printf("\n maximum profit is %.f", tp);
}
```

```
int main()
{
float weight[20], profit[20], capacity;
int num i, j;
float ratio[20], temp;
printf("\n Enter the number of objects:");
scanf("%d", &num);
printf("\n Enter the weights and profit of each
 object:");
for(i=0; i<num; i++)
{
  scanf("%f %f", &weight[i], &profit[i]);
}
printf("\n Enter capacity of knapsack:");
scanf("%f", &capacity);
for(i=0; i<num; i++)
{
  ratio[i]=profit[i]/weight[i];
}
for(i=0; i<num; i++)
{
  for(j=i+1; j<num; j++)
  {
    if(ratio[i]<ratio[j])
    {
      temp=ratio[j];
      ratio[j]= ratio[i];
      ratio[i]=temp;
      temp=weight[j]
      weight[i]=temp;
      weight[j]=weight[i];
```

```
temp=profit[j];
profit[j]=profit[i];
profit[i]=temp;
}
}
}
knapsack(num,weight, profit, capacity);
return 0;
}
```

Write program to implement backtracking algorithm for solving problem like NQueens

```
#include<stdio.h>
#include<math.h>
#define TRUE 1
#define FALSE 0
int i,j;
void print_solution(int n, int x[])
{
  char c[10][10];
  for(int i=1; i<=n; i++)
  {
    for(int j=1; j<=n; j++)
    {
      c[i][j] = 'X';
    }
  }
  for(i=1; i<=n; i++)
  {
    c[i][x[i]] = 'Q';
  }
  for(i=1; i<=n; i++)
  {
    for(int j=1; j<=n; i++)
    {
      printf("%c", c[i][j]);
    }
    printf("\n");
  }
}
```

```
int place(int x[], int k)
{
 for(int i=1; i<=k-1; i++)
 {
   if(x[i]==x[k] || abs(i-k)==abs(x[i]-x[k]))
   {
     return FALSE;
   }
 }
 return TRUE;
}
void nqueens(int n)
{
 int x[10];
 int count=0;
 int k=1;
 x[k]=0;
 while(k!=0)
 {
   x[k]+=1;
   while((x[k]<=n) && (!place(x,k))
   {
     x[k]+=1;
   }
   if(x[k]<=n)
   {
     if(k==n)
     {
       count++;
       printf("Solution %d is \n", count);
       print_solution(n,x)
```

```
}
else
{
    k++;
    x[k]=0;
}
}
else
k--;
}
}
void main()
{
int n;
printf("Enter the number of queens\n");
scanf("%d",&n);
nqueens(n);
}
```

C program to solve sum of subset

```
#include<stdio.h>
void backtrack(int set[], int n, subset[], int k, int
targetsum, int index, int sum);
void print_solution(int subset[], int k);
void main()
{
  int set[10], targetsum, n, i;
  printf("Enter The number of elements in the set:");
  scanf("%d", &n);
  printf("Enter The elements in the set:\n");
  for(i=0; i<n; i++)
  {
    scanf("%d", &set[i]);
  }
  printf("Enter the targetsum:");
  scanf("%d", &targetsum);
  int subset[10];
  printf("The subsets are:\n");
  backtrack(set, n, subset, 0, targetsum, 0, 0);
}
void backtrack(int set[], int n, int subset[], int k,
int targetsum, int index, int sum)
{
  if(sum == targetsum)
  {
    print_solution(subset, k);
    return;
  }
```

```
if(index >= n || sum > targetsum)
{
  return;
}
subset[k] = set[index];
backtrack(set, n, subset, k+1, targetsum, index+1, sum+
set[index]);
backtrack(set, n, subset, k, targetsum, index+1, sum);
}
void print_solution(int subset[], int k)
{
  int i;
  printf("{");
  for(i=0; i<k; i++)
  {
    printf("%d", subset);
  }
  printf("}\n");
}
```

Write program to implement greedy algorithm for job sequencing with deadlines

```
#include<stdio.h>
void jobsequencing(int n, int d[10], int p[10])
{
  int dmax, J[10], i, k, tp=0;
  dmax=d[i];
  for(i=2;i<=n;i++)
  {
    if(d[i]>dmax)
    {
      dmax=d[i];
    }
  }
  for(i=1;i<=dmax;i++)
  {
    J[i]=0;
  }
  for(i=1;i<=n;i++)
  {
    k=d[i];
    while(k!=0)
    {
      if(J[k]==0)
      {
        J[k]=i;
        tp=tp+p[i];
        break;
      }
      k--;
    }
```

```
printf("The final solution vector which gives
optimal solution is :\n");
for(i=1; i<=dmax; i++)
{
  printf("%d\n", J[i]);
}
printf("The total profit is %d", tp);
}
int main()
{
  int num, i, j, p[10], d[10], temp;
  printf("Enter the number of jobs:");
  scanf("%d", &num);
  printf("Enter the profits and deadlines of each
  objects");
  for(i=1; i<=num; i++)
  {
    scanf("%d%d", &p[i], &d[i]);
  }
  for(i=1; i<=num; i++)
  {
    for(j=i+1; j<=num; j++)
    {
      if(p[i]<p[j])
      {
        temp=p[j];
        p[j]=p[i];
        p[i]=temp;
        temp=d[j];
        d[j]=d[i];
        d[i]=temp;
```

```
}
}
jobsequencing(num, d, p);
return 0;
}
```

Write program to implement dynamic programming for the optimal binary search tree problem

```
#include<stdio.h>
#include<conio.h>
int find(int c[5][5], int r[5][5], int i, int j)
{
    int min=999, m, t;
    for(m=r[i][j-1]; m<r[i+1][j]; m++)
    {
        if((c[i][m-1]+c[m][j])<min)
        {
            min=c[i][m-1]+c[m][j];
            t=m;
        }
    }
    return t;
}
void OBST(int n, int p[5], int q[5])
{
    int i,j,k,m,w[5][5], c[5][5], r[5][5];
    for(i=0; i<=n; i++)
    {
        w[i][i]=q[i];
        r[i][i]=0;
        c[i][i]=0;
    }
    for(i=0;i<n;i++)
    {
        w[i][i+1]=p[i+1]+q[i+1]+w[i][i];
        r[i][i+1]=i+1;
        c[i][i+1]=w[i][i+1];
```

```
for(m=2; m<=n; m++)
{
   for(i=0; i<=n; i++)
   {
      j=i+m;
      w[i][j]=p[j]+q[j]+w[i][j-1];
      k=Find(c,r,i,j);
      c[i][j]=c[i][k-1]+c[k][j]+w[i][j];
      r[i][j]=k;
   }
}
printf("The cost of optimal binary search tree
whose node is %d is %d", r[0][n], c[0][n]);
}
void main()
{
  int n, p[5], q[5], i;
  printf("Enter the number of identifiers in the
  binary search tree\n");
  scanf("%d", &n);
  printf("Enter the successful search\n");
  for(i=1; i<=n; i++);
  {
    scanf("%f", &p[i]);
  }
  printf("Enter the probabilities of unsuccessful search\
  for(i=0; i<=n; i++)
  {
  scanf("%f", &q[i]); }
  OBST(n, p, q);
  getch();
}
```

C program to solve prims algorithm problem

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10

int totalweight=0
int find_source(int d[MAX], int s[MAX], int n)
{
  int min=9999 source=-1, i;
  for(i=0; i<n; i++)
  {
    if(s[i]==0 && d[i]<min)
      min=d[i];
      source=i;
  }
}
return source;
}
void prims(int n, int w[MAX][MAX])
{
  int p[MAX];
  int s[MAX];
  int d[MAX];
  int i,j;
  for(i=0;i<n;i++)
  {
    s[i]=0;
    d[i]=9999;
    p[i]=-1;
  }
```

```
for(i=0; i<n-1; i++)
{
  int u=find_source(d,s,n);
  if(u==-1)
  {
    printf("Graph is disconnected. No MST can be
    formed\n");
    return;
  }
s[u]=1;
for(j=0; j<n; j++)
{
  if(w[u][j]!=0 && s[j]==0 && w[u][j]<d[j])
  {
    p[j]=u;
    d[j]=w[u][j];
  }
}
}
printf("Edge \t weight\n");
for(i=1; i<n; i++)
{
  if(p[i]!=-1)
  {
    printf("%d - %d\t %d\n", p[i], i, w[i][p[i]]);
    totalweight+=w[i][p[i]];
  }
}
}
printf("Total weight of MST: %d\n", totalweight);
}
```

```
int main()
{
  int n, i, j;
  int w[MAX][MAX];
  printf("Enter the number of vertices:");
  scanf("%d", &n);
  if(n>MAX)
  {
    printf("Number of vertices exceeds maximum
    limit(%d)\n", MAX);
    return 1;
  }
  printf("Enter the adjeccency matrix (0 for no edge):\n");
    for(i=0; i<n; i++)
    {
      for(j=0; j<n; j++)
      {
        scanf("%d", &w[i][j]);
        if(w[i][j]<0)
        {
          printf("Negetive weights are not allowed in
          prims algorithm\n")
          return 1;
        }
      }
    }
}
```

Write an program that implements kruskals algorithm to generate minimum cost spanning tree

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
int parent[MAX];
//function to find the root of an element
int find(int i)
{
  while(parent[i]!=i)
  i=parent[i];
  return i;
}
void union1(int i, int j)
{
  int a=find(i);
  int b=find(j);
  parent[a]=b;
}
void kruskal(int n, int edges[][3], int e)
{
  int mincost=0, i, j, temp[3], u, v, w;
  int count=0;
  for(i=0; i<n; i++)
  {
    parent[i]=i;
    for(j=0; j<e; j++)
    {
      for(j=0; j<e-i-1; j++)
      {
```

```
if(edges[j][2] > edges[j+1][2])
{
    temp[0] = edges[j][0];
    temp[1] = edges[j][1];
    temp[2] = edges[j][2];
    edges[j][0] = edges[j+1][0];
    edges[j][1] = edges[j+1][1];
    edges[j][2] = edges[j+1][2];
    edges[j+1][0] = temp[0]
    edges[j+1][1] = temp[1];
    edges[j+1][2] = temp[2];
  }
 }
}
printf("Edges in the MST:\n");
for(i=0; i<e; i++)
{
 u=edges[i][0];
 v=edges[i][1];
 w=edges[i][2];
 if(find(u)!=find(v))
 {
   printf("%d-%d : %d\n", u,v,w);
   mincost+=w;
   count++;
   union1(u,v);
 }
}
if(count!=n-1)
{
   printf("Spanning tree is not possible");
```

```
}
printf("Minimum cost = %d\n", mincost);
}
int main()
{
int n, e, i;
int edges[MAX][3];
printf("Enter The number of vertices:");
scanf("%d", &n);
printf("Enter the number of edges:");
scanf("%d", &e);
printf("Enter The edges (u v w):\n");
for(i=0; i<e; i++)
{
scanf("%d %d %d", &edges[i][0], &edges[i][1], &
edges[i][2]);
}
kruskal(n, edges, e);
return 0;
}
```