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

```
#include<stdio.h>
#include<conio.h>
int main ()
{
int i, temp, j, min, n, a [10];
printf ("enter the value of n:");
scanf("%d",&n);
printf("enter the array elements:");
for (i=0; i<n;i++)
scanf("%d",&a[i]);

  for (i=0; i<=n-2; i++)
  {
     min=i;
     for (j=i+1; j<=n-1; j++)
       {
            if (a[j]<a[min])
               min=j;
       }
               temp=a[min];
              a[min]=a[i];
              a[i]=temp;
   }
printf("sorted array is \n");
for (i=0; i<n; i++)
printf("%d\t",a[i]);
return 0;
}
```

OUTPUT:

```
enter the value of n:5
enter the array elements: 7 2 9 1 4
sorted array is
    1       2      4      7      9
```

2.Write a program to perform Travelling Salesman Problem

```
#include <stdio.h>
int G[10][10] = {
        { 0, 2, 8, 5 },
        { 2, 0, 3, 4 },
        { 8, 3, 0, 7 },
        { 5, 4, 7, 0 }
        };
int visited[10], n, cost = 0;

void tsp(int c)
{
   int k, adj_vertex = 999;
   int min = 999;

   visited[c] = 1;
   printf("%d ", c + 1);

    for(k = 0; k < n; k++)
    {
     if((G[c][k] != 0) && (visited[k] == 0))
      {
        if(G[c][k] < min)
              {
           min = G[c][k];
           adj_vertex = k;
        }
      }
   }

     if(min != 999)
     {
            cost = cost + min;
          }

        if (adj_vertex == 999)
         {
           adj_vertex = 0;
           printf("%d", adj_vertex + 1);
           cost = cost + G[c][adj_vertex];
           return;
        }

        tsp(adj_vertex);
    }

    int main()
    {
        int i, j;
        n = 5;
        for(i = 0; i < n; i++)
        {
            visited[i] = 0;
        }
        printf("Shortest Path: ");
        tsp(0);
        printf("\nMinimum Cost: ");
        printf("%d\n", cost);
        return 0;
         }
```

OUTPUT:

```
Shortest Path: 1 2 3 4 1
Minimum Cost: 17
```

3. Write program to implement Dynamic Programming algorithm for the 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, j;
    int K[n+1][W+1];
      for (i = 0; i <=n; i++) {
        for (j = 0; j <= W; j++) {
            if (i == 0 || j == 0)
                K[i][j] = 0;
            else if (wt[i]<= j)
            K[i][j] = max(K[i - 1][j],K[i - 1][j - wt[i]]+val[i]) ;
            else
                K[i][j] = K[i-1][j];
        }
    }
    return K[n][W];
}

int main()
{
    int profit[] = { 0,2, 3, 4,1 };
    int weight[] = { 0,3, 4, 5,6 };
    int W = 8;
    int n= 4;
    printf("%d", knapSack(W, weight, profit, n));
    return 0;
}
```

OUTPUT:

```
6
```

4.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;
}
```

<u>OUTPUT</u>

```
Enter the number of items :5

Enter Weight and Profit for item[0] :

1 5

Enter Weight and Profit for item[1] :

3 10

Enter Weight and Profit for item[2] :

5 15

Enter Weight and Profit for item[3] :
```

```
4 7

Enter Weight and Profit for item[4] :

1 8

Enter the capacity of knapsack :

11

Knapsack problems using Greedy Algorithm:


The maximum value is :39.750000
```

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

```
#include<stdio.h>
#include<conio.h>
#define MAX 5
int visited[MAX]={0};
int G[MAX][MAX],i,j;

void DFS(int start)
{
     int stack[MAX];
     int top=-1,i,k;
     for (k=0;k<MAX;k++)
       visited[k]=0;

     stack[++top]= start;

     visited[start]=1;

     while(top!=-1)  //stack empty
     {
              start=stack[top--];
              printf("%c-", start+65);

              for(i=0;i<MAX;i++)   //adding neighbours
              {
                       if (G[start][i] && visited[i]==0)
                       {
                       stack[++top]=i;
                       visited[i]=1;
                       break;
                       }
```

```
                }

          }
    }
     void BFS(int start)
     {
          int q[MAX], rear=-1,front=-1,i,k;
          for(k=0;k<MAX;k++)
           visited[k]=0;

           q[++rear]=start;
           ++front;
           visited[start]=1;

           while(rear>=front)
           {
           start=q[front++];
           printf("%c-",start+65);

            for(i=0;i<MAX;i++)
            {
                if(G[start][i] && visited[i]==0)
              {
                 q[++rear]=i;
                visited[i]=1;
              }
            }
         }
    }

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

```
printf("Enter the adjacency matrix ");
    for(i=0;i<MAX;i++)
         for(j=0;j<MAX;j++)
            scanf("%d",&G[i][j]);
printf("\nDFS Traversal\n");
   DFS(0);
printf("\nBFS Traversal\n");
     BFS(0);
return 0;
}
```

OUTPUT:

```
Enter the adjacency matrix
0 1 1 1 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0

DFS Traversal
A-B-D-E-C-
BFS Traversal
A-B-C-D-E-
```

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

```
#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(" \n Enter the total number of numbers:");
   scanf("%d",&num);
   printf("Enter the number\n");
   for(i=1;i<=num;i++)
   scanf("%d",&a[i]);
   maxmin(1, num);
   printf("Minimum element in array:%d\n",min);
   printf("Maximum element in array:%d\n",max);
   return 0;
   }
```

<u>OUTPUT</u>

```
Enter the total number of numbers: 8
Enter the number
23 90 12 56 34 77 61 43
Minimum element in array: 12
Maximum element in array:90
```

7. 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>
#include<conio.h>

void Quicksort (int a [], int first, int last)
{
int i, j,pivot,temp;
 if(first<last)
 {
   pivot=first;
   i=first;
   j=last;
  while(i<j)
   {
     while (a[i]<=a[pivot]&&i<last)
         i++;
     while (a[j]>a[pivot])
         j--;
    if(i<j)
      {
        temp=a[i];
        a[i]=a[j];
        a[j]=temp;
      }
   }

   temp=a[pivot];
   a[pivot]=a[j];
   a[j]=temp;
   Quicksort(a,first,j-1);
   Quicksort(a,j+1,last);
  }
}

int main()
{
int i,n,a[50];
printf("Enter the number of elements");
scanf("%d",&n);
printf("enter the elements");
for(i=0;i<n;i++)
scanf("%d",&a[i]);

Quicksort(a,0,n-1);

printf("Sorted elements");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
return 0;
}
```

OUTPUT

```
Enter the number of elements 5
enter the elements
56 97 42 65 68
Sorted elements 42      56      65      68      97
```

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

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

void merge (int a [], int low, int mid, int high)
{
int temp [20];
int i, j, k;
i=low;
j=mid+1;
k=0;
 while(i<=mid && j<=high)
 {
   if(a[i]<=a[j])
      temp[k++]=a[i++];
   else
      temp[k++]=a[j++];
 }

  while(i<=mid)
   {
    temp[k++]=a[i++];
    }
  while(j<=high)
    {
     temp[k++]=a[j++];
    }
for(i=low,j=0;i<=high;i++,j++)
    a[i]=temp[j];
}
```

```
void mergesort(int a[],int low,int high)
{
int mid;
if(low<high)
 {
  mid=(low+high)/2;
  mergesort(a,low,mid);
  mergesort(a, mid+1,high);
  merge(a,low,mid,high);
 }
}

int main()
{
int i,n,a[20];
printf("Enter the number of elements :\n");
scanf("%d",&n);
printf("Enter the elements:\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);

mergesort(a,0,n-1);

printf("Sorted list as follows:\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
return 0;
}
```

OUTPUT

```
Enter the number of elements:
5
Enter the elements:
48 90 1 24 18
Sorted list as follows:
1       18      24      48      90
```

9. Sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort.

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

void merge (int a[], int low, int mid, int high)
{
int temp [30000];
int i, j, k;
i=low;
j=mid+1;
k=0;
 while(i<=mid && j<=high)
 {
   if(a[i]<=a[j])
      temp[k++]=a[i++];
   else
      temp[k++]=a[j++];
 }

  while(i<=mid)
   {
    temp[k++]=a[i++];
    }
  while(j<=high)
    {
     temp[k++]=a[j++];
    }
 for(i=low,j=0;i<=high;i++,j++)
    a[i]=temp[j];
}


void mergesort(int a[],int low,int high)
{
int mid;
if(low<high)
 {
  mid=(low+high)/2;
  mergesort(a,low,mid);
  mergesort(a, mid+1,high);
  merge(a,low,mid,high);
 }
}

int main()
{
int a[30000],i,n;
double execution_time;

struct timespec begin,end;



printf("Enter the number of elements :\n");
scanf("%d",&n);
printf("Random numbers generation: \n");

  for (i = 0; i <n; i++)
  {
     a[i]=rand()%1000;
     printf(" %d ", a[i]);
  }

 clock_gettime(CLOCK_MONOTONIC, &begin);
mergesort(a,0,n-1);
clock_gettime(CLOCK_MONOTONIC, &end);

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

execution_time = ((double)(end.tv_nsec - begin.tv_nsec)/1000000000);
printf("\n\n Time = %lf Seconds",execution_time);
printf("\n\nTime Complexity =  %lf", n * log(n));

return 0;
}
```

OUTPUT

```
Enter the number of elements :
30
Random numbers generation:
 41  467  334  500  169  724  478  358  962  464  705  145  281  827  961  491
995  942  827  436  391  604  902  153  292  382  421  716  718  895

Sorted array  :
 41  145  153  169  281  292  334  358  382  391  421  436  464  467  478  491
500  604  705  716  718  724  827  827  895  902  942  961  962  995

 Time = 0.000098 Seconds

Time Complexity =  102.035921
```

10. Sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort.

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<stdlib.h>
#include<math.h>
void Quicksort (int a [], int first, int last)
{
int i, j,pivot,temp;
 if(first<last)
 {
   pivot=first;
   i=first;
   j=last;
  while(i<j)
   {
     while (a[i]<=a[pivot]&&i<last)
         i++;
     while (a[j]>a[pivot])
         j--;
    if(i<j)
      {
        temp=a[i];
        a[i]=a[j];
        a[j]=temp;
      }
   }
   temp=a[pivot];
   a[pivot]=a[j];
   a[j]=temp;
   Quicksort(a,first,j-1);
   Quicksort(a,j+1,last);
  }
}

int main()
{
int i,n,a[30000];
double execution_time;
struct timespec begin,end;

printf("Enter the number of elements");
scanf("%d",&n);
printf("Random Numbers Generation");
  for (i = 0; i <n; i++)
  {
     a[i]=rand()%1000;
     printf(" %d ", a[i]);
  }
clock_gettime(CLOCK_MONOTONIC, &begin);
Quicksort(a,0,n-1);
clock_gettime(CLOCK_MONOTONIC, &end);

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

execution_time = ((double)(end.tv_nsec - begin.tv_nsec)/1000000000);
printf("\n\n Time = %lf Seconds",execution_time);
printf("\n\nTime Complexity =  %lf", n * log(n));
return 0;
}
```

OUTPUT:

```
Enter the number of elements 30
Random Numbers Generation 41 467 334  500  169  724  478  358  962  464
705  145  281  827  961  491  995  942  827  436  391  604  902  153  292
382  421  716  718  895

Sorted array  :
: 41  145  153  169  281  292  334  358  382  391  421  436  464  467  478
491  500  604  705  716  718  724  827  827  895  902  942  961  962  995

 Time = 0.000002 Seconds

 Time Complexity =  102.035921
```

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

```
#include <stdio.h>
int main()
{
int adj[10][10] = {0}; // Initialize the adjacency matrix with zeros
int vertex,edge;
int i, j, u, v;
printf("Enter the number of vertices in the graph: ");
scanf("%d", &vertex);

printf("Enter the number of edges in the graph: ");
scanf("%d", &edge);

printf("Enter the edges (u, v):\n");
for (i = 0; i < edge; i++)
  {
   scanf("%d %d", &u, &v);
   adj[u][v] = 1;
   adj[v][u] = 1;
  }
printf("\nAdjacency Matrix:\n");
for (i = 0; i < vertex; i++)
 {
   for (j = 0; j < vertex; j++)
    {
      printf(" %d ", adj[i][j]);
    }
  printf("\n");
 }
 return 0;
}
```

OUTPUT:

```
Enter the number of vertices in the graph: 4
Enter the number of edges in the graph: 4
Enter the edges (u, v):
0 1
0 2
0 3
1 2

Adjacency Matrix:
 0 1 1 1
 1 0 1 0
 1 1 0 0
 1 0 0 0
```

12. Implement function to print In-Degree, Out-Degree and to display that adjacency matrix.

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

int main()
{
int node,edge,a[10][10]={0}
int i, j, node1, node2, rsum, csum;
printf("Enter number of nodes \n:");
scanf("%d",&node);
printf("Enter number of edges \n");
scanf("%d",&edge);
printf("Enter node1 and node2 information of the directed edges \n");
  for (i=1; i<=edge;i++)
  {
    scanf("%d%d”, &node1,&node2);
    a[node1] [node2] =1;
  }
printf("The adjacency matrix is \n");
  for (i=1; i<=node; i++)
  {
    for (j=1; j<=node; j++)
    printf("%d\t",a[i][j]);
    printf("\n");
  }
for(i=1;i<=node;i++)
{
 rsum=0;
 csum=0;
    for(j=1;j<=node;j++)
      {
```

```
     rsum=rsum+a[i][j];
     csum=csum+a[j][i];
    }
  printf("Indegree of node %d is %d\n",i,csum);
  printf("Outdegree of node %d is %d\n",i,rsum);
printf("\n");
}
return 0;
}
```

OUTPUT

```
Enter number of nodes
4
Enter number of edges
6
Enter node1 and node2 information of the directed edges
1 2
2 1
3 2
1 3
3 4
4 3
The adjacency matrix is
0       1       1       0
1       0       0       0
0       1       0       1
0       0       1       0
Indegree of node 1 is 1
Outdegree of node 1 is 2

Indegree of node 2 is 2
Outdegree of node 2 is 1
```

Indegree of node 3 is 2

Outdegree of node 3 is 2

Indegree of node 4 is 1

Outdegree of node 4 is 1

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

```
#include <stdio.h>
void job(int profit[],int dline[], int n, int max)
{
       int total=0, i, j, result[20]={ 0 };

        for(i=0;i<n; i++)
            {
                if(result[dline[i]-1] == 0)
                {
                    result [dline[i]-1] = profit[i];
                    total=total + profit[i];
                }
            }
        printf("\n Total profit is:  %d", total) ;
        printf("\n Job sequence is \n");
            for (i=0;i<max; i++)
                     printf("%d    %d \n", i+1,result[i]);
      }
int main()
{
    int profit[20], dline[20], n, i, j, max;
    printf("\n Enter number of jobs :");
    scanf("%d", &n);
    printf("\n Enter profit in descending order and its Deadline:\n");
   for(i=0;i<n; i++)
        scanf("%d%d",&profit[i], &dline[i]);

    printf("\n Enter Maximum deadline:");
          scanf("%d",&max);
            job(profit,dline,n,max);
```

```
 }
```

OUTPUT

```
Enter number of jobs :4

 Enter profit in descending order and its Deadline:
100 2
27 1
15 2
10 1

 Enter Maximum deadline:2

 Total profit is:  127
 Job sequence is
1    27
2    100
```

17. Write a program that implements Prim's algorithm to generate minimum cost spanning Tree.

```
#include<stdio.h>
int a,b,u,v,n,i,j,ne=1;
int visited [10] = {0}, min, mincost=0, cost [10][10];
int 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 Minimum cost=%d",mincost);
}
```

OUTPUT

```
Enter the number of nodes:4

 Enter the adjacency matrix:
0 5 7 2
5 0 0 3
7 0 0 4
2 3 4 0


 Edge 1:(1 4) cost:2
 Edge 2:(4 2) cost:3
 Edge 3:(4 3) cost:4
 Minimum cost=9
```