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

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,position,swap,a[100],d,n;
clrscr();
printf("Enter the No. Of Elements\n");
scanf("%d",&n);
printf("Enter %d Integers\n",n);
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
for(i=0;i<(n-1);i++)
{
position=i;
for(d=i+1;d<n;d++)
{
if(a[position]>a[d])
position=d;
}
if(position!=i)
{
swap=a[i];
a[i]=a[position];
a[position]=swap;
}
}
printf("Sorted List in Ascending Order is:\n");
for(i=0;i<=n;i++)
{
printf("%d \t",a[i]);
}
printf("\n");
getch();
}
```

OUTPUT:

```
Enter the No. of Elements
5
Enter 5 Integers
34
99
12
86
25
Sorted List in Ascending Order is:
12   25    34   86   99
```

2. Write a program to perform Travelling Salesman Problem

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int a[10][10],visited[10],n,cost=0;
void get()
{
int i,j;
printf("\n\nEnter Number of Cities: ");
scanf("%d",&n);
printf("\nEnter 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",&a[i][j]);
visited[i]=0;
}
printf("\n\nThe Cost Matrix is:\n");
for(i=0;i<n;i++)
{
printf("\n\n");
for(j=0;j<n;j++)
printf("\t%d",a[i][j]);
}
}
void mincost(int city)
{
int i,ncity,least(int city);
visited[city]=1;
printf("%d ===>",city+1);
ncity=least(city);
if(ncity==999)
{
ncity=0;
printf("%d",ncity+1);
cost+=a[city][ncity];
return;
}
mincost(ncity);
}
int least(int c)
{
int i,nc=999;
```

```
int min=999,kmin;
for(i=0;i<n;i++)
{
if((a[c][i]!=0)&&(visited[i]==0))
if(a[c][i]<min)
{
min=a[i][0]+a[c][i];
kmin=a[c][i];
nc=i;
}
}
if(min!=999)
cost+=kmin;
return nc;
}
void put()
{
printf("\n\nMinimum cost:");
printf("%d",cost);
}
void main()
{
clrscr();
get();
printf("\n\nThe Path is:\n\n");
mincost(0);
put();
getch();
}
```

OUTPUT

3. **Write a program to implement Dynamic Programming algorithm for the 0/1 Knapsack problem.**

```
#include<stdio.h>
#include<conio.h>
int max(int a, int b)
{
return(a>b)?a:b;
}
int knapsack(int W,int wt[],int val[],int n)
{
if(n==0 || W==0)
return 0;
if(wt[n-1]>W)
return knapsack(W,wt,val,n-1);
else
return max(val[n-1]+knapsack(W-wt[n-1],wt,val,n-1),knapsack(W,wt,val,n-1));
}
int main()
{
int profit[]={60,100,120};
int weight[]={10,20,30};
int W=50;
int n=sizeof(profit)/sizeof(profit[0]);
printf("%d",knapsack(W,weight,profit,n));
getch();
return 0;
}
```

OUTPUT

220

4. Write a Program to Perform Knapsack Problem using Greedy Solution

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void knapsackGreedy(int M,int n,int w[20],int P[20])
{
double x[20];
double sum=0.0;
int i,j;
double ratio[20];
int order[20];
int Rc=M;
for(i=0;i<n;i++)
{
ratio[i]=(double)P[i]/w[i];
}
for(i=0;i<n;i++)
{
order[i]=i;
}
for(i=0;i<n-1;i++)
{
for(j=0;j<n-i-1;j++)
{
if(ratio[order[j]]<ratio[order[j+1]])
{
int tempIndex=order[j];
order[j]=order[j+1];
order[j+1]=tempIndex;
}
}
}
for(i=0;i<n;i++)
{
x[i]=0;
}
for(i=0;i<n;i++)
{
int currentIndex=order[i];
if(w[currentIndex]<=Rc)
{
x[currentIndex]=1;
Rc=Rc-w[currentIndex];
sum=sum+P[currentIndex];
```

```
}
else
{
x[currentIndex]=(double)Rc/w[currentIndex];
sum+=(P[currentIndex]*x[currentIndex]);
break;
}
}
printf("Solution vector 'x': ");
for(i=0;i<n;i++)
{
printf("%.21f",x[i]);
}
printf("\nMaximum profit: %.21f\n",sum);
}
int main()
{
int M,n,i,w[20],P[20];
clrscr();
printf("Enter Knapsack capacity(M): ");
scanf("%d",&M);
printf("Enter the number of objects(n): ");
scanf("%d",&n);
printf("Enter weights for each object:\n");
for(i=0;i<n;i++)
{
scanf("%d",&w[i]);
}
printf("Enter profits for each object:\n");
for(i=0;i<n;i++)
{
scanf("%d",&P[i]);
}
knapsackGreedy(M,n,w,P);
getch();
return 0;
}
```

OUTPUT

Enter Knapsack capacity (M): 40
Enter the number of objects (n): 4
Enter weights for each object:
20 25 10 15
Enter profits for each object:
20 40 35 45
Solution vector ‘x’: 0.00 0.60 1.00 1.00
Maximum Profit: 104.00

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

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 100
int c[MAX][MAX];
int visited[MAX];
int queue[MAX];
int n;
void BFS(int v)
{
int front=0,rear=-1,i;
visited[v]=1;
queue[++rear]=v;
while(front<=rear)
{
v=queue[front++];
printf("%d",v);
for(i=1;i<=n;i++)
{
if(c[v][i]==1&&visited[i]==0)
{
queue[++rear]=i;
visited[i]=1;
}
}
}
printf("\n");
}
void DFS(int v)
{
int i;
visited[v]=1;
printf("%d",v);
for(i=1;i<=n;i++)
{
if(c[v][i]==1&&visited[i]==0)
{
DFS(i);
}
}
}
int main()
{
```

```
int i,j,v;
clrscr();
printf("Enter the number of verticesin the graph; ");
scanf("%d",&n);
printf("Enter the cost matrix of the graph: \n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&c[i][j]);
for(i=1;i<=n;i++)
visited[i]=0;
printf("Enter the starting vertex for BFS: ");
scanf("%d",&v);
printf("BFS traversal of the graph is: ");
BFS(v);
for(i=1;i<=n;i++)
visited[i]=0;
printf("Enter the starting vertex for DFS: ");
scanf("%d",&v);
printf("DFS traversal of the graph is: ");
DFS(v);
printf("\n");
getch();
return 0;
}
```

OUTPUT

```
Enter the number of vertices in the graph: 6
Enter the cost matrix of the graph:
0 1 1 0 0 0
1 0 0 1 1 0
1 0 0 0 0 1
0 1 0 0 0 0
0 1 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0
Enter the starting vertex for BFS: 1
BFS traversal of the graph is: 1 2 3 4 5 6
Enter the Starting vertex for DFS: 1
DFS traversal of the graph is: 1 2 4 5 3 6
```

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

```
#include<stdio.h>
#include<conio.h>
int start,end,mid;
int min1,max1,min2,max2;
void findMinMax(int arr[],int start,int end,int *min,int *max)
{
if(start==end)
{
*min=*max=arr[start];
return;
}
if(end-start==1)
{
if(arr[start]<arr[end])
{
*min=arr[start];
*max=arr[end];
}
else
{
*min=arr[end];
*max=arr[start];
}
return;
}
mid=(start+end)/2;
findMinMax(arr,start,mid,&min1,&max1);
findMinMax(arr,mid+1,end,&min2,&max2);
*min=(min1<min2)?min1:min2;
*max=(max1<max2)?max1:max2;
}
int main()
{
int n,i;
int arr[100];
int min,max;
printf("enter the number of elements:");
scanf("%d",&n);
printf("enter the array elements:\n");
for(i=0;i<n;i++)
{
scanf("%d",&arr[i]);
```

```
}
findMinMax(arr,0,n-1,&min,&max);
printf("Minimum value in the array: %d\n",min);
printf("Maximum value in the array: %d\n",max);
getch();
return 0;
}
```

OUTPUT

```
Enter the number of elements: 8
Enter the array elements:
10 50 40 80 70 60 20 30
Minimum value in the array: 10
Maximum value in the array: 80
```

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 low, int high)
{
int j;
if(low<high)
{
j=partition(A,low,high);
quicksort(A,low,j-1);
quicksort(A,j+1,high);
}
}
int partition(int A[],int low, int high)
{
int pivot,j,temp,i;
pivot=low;
i=low;
j=high;
while(i<j)
{
while(i<high && A[i]<=A[pivot])
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;
return j;
}
void main()
{
int i,n;
int A[100];
clrscr();
printf("Enter the number of elements of array:  ");
```

```
scanf("%d",&n);
printf("Enter the elements of the array:  ");
for(i=0;i<n;i++)
scanf("%d",&A[i]);
quicksort(A,0,n-1);
printf("Sorted list of elements:  ");
for(i=0;i<n;i++)
printf("%d  ",A[i]);
getch();
}
```

OUTPUT

```
Enter the number of elements of array: 8
Enter the elements of array: 80 10 60 40 20 30 50 70
Sorted list of elements: 10 20 30 40 50 60 70 80
```

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 mergesort(int arr[],int low,int h);
void main(void)
{
int array[100],n,i=0;
clrscr();
printf("\t\t\tMerge Sort\n\n\n");
printf("Enter the number of elements to be sorted: \n");
scanf("%d",&n);
printf("\nEnter the elements to be sorted: \n");
for(i=0;i<n;i++)
{
printf("\tArray[%d]= ",i);
scanf("%d",&array[i]);
}
printf("\nBefore mergesort:");
for(i=0;i<n;i++)
{
printf("%4d",array[i]);
}
printf("\n");
mergesort(array,0,n-1);
printf("\nAfter Mergesort:");
for(i=0;i<n;i++)
{
printf("%4d",array[i]);
}
printf("\n");
getch();
}
void mergesort(int arr[],int min,int h)
{
int i=0;
int length=h-min+1;
int pivot=0;
int merge1=0;
int merge2=0;
int temp[100];
if(min==h)
return;
pivot=(min+h)/2;
```

```
mergesort(arr,min,pivot);
mergesort(arr,pivot+1,h);
for(i=0;i<length;i++)
{
temp[i]=arr[min+i];
}
merge1=0;
merge2=pivot-min+1;
for(i=0;i<length;i++)
{
if(merge2<=h-min)
{
if(merge1<=pivot-min)
{
if(temp[merge1]>temp[merge2])
{
arr[i+min]=temp[merge2++];
}
else
{
arr[i+min]=temp[merge1++];
}
}
else
{
arr[i+min]=temp[merge2++];
}
}
else
{
arr[i+min]=temp[merge1++];
}
}
}
```

OUTPUT

```
                Merge Sort
Enter the number of elements to be sorted:
5
Enter the elements to be sorted:
Array[0]=34
Array[1]=89
Array[2]=05
Array[3]=12
Array[4]=76

Before Merge Sort: 34 89 05 12 76

After Merge Sort: 5 12 34 76 89
```