

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

```
#include<stdio.h>

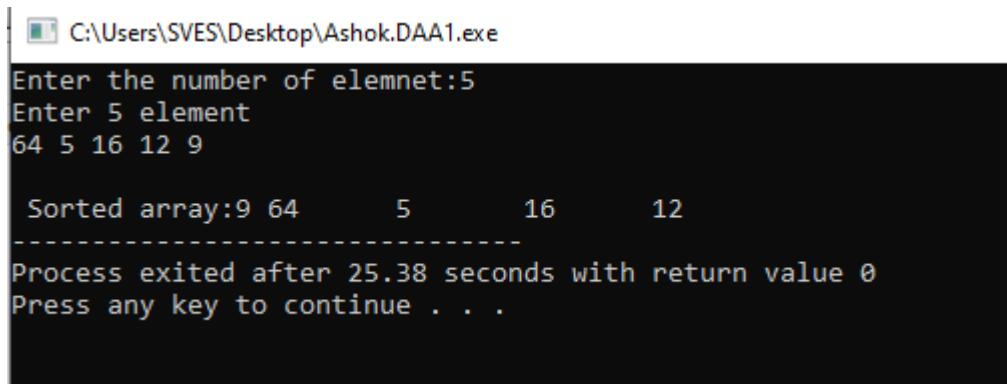
void swap(int *xp,int *yp)
{
    int temp=*xp;
    *xp=*yp;
    *yp=temp;
}

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

int main()
{
```

```
int n,i;  
printf("Enter the number of elemnet:");  
scanf("%d",&n);  
int arr[n];  
printf("Enter %d element",n);  
for(i=0;i<n;i++)  
{  
    scanf("%d",&arr[i]);  
}  
selectionSort(arr,n);  
printf("\n Sorted array:");  
for(i=0;i<n;i++)  
{  
    printf("%d\t",arr[i]);  
}  
return 0;  
}
```

Output



The screenshot shows a terminal window with the following output:

```
C:\Users\SVES\Desktop\Ashok.DAA1.exe  
Enter the number of elemnet:5  
Enter 5 element  
64 5 16 12 9  
  
Sorted array:9 64      5      16      12  
-----  
Process exited after 25.38 seconds with return value 0  
Press any key to continue . . .
```

2) Write a program to perform Traveling Salesman Problem.**Code**

```
#include<stdio.h>
#include<limits.h>
#define MAX_CITIES 10
void swap(int*a,int*b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
}
void permute(int cities[], int start,intend,int*minCost )
{
    int i,cost=0;
    if(start==end)
    {
        for(i=0;i<end-1;i++)
        {
            cost+=cities[i];
        }
        cost+=cities[i];
        *minCost=(cost<*minCost)?cost:*minCost;
    }
    else
    {
        for(i=start;i<end;i++)
```

```
    swap(&cities[start],&cities[i]);
    permute(cities,start+1,end,minCost);
    swap(&cities[start],&cities[i]);
}

}

int main()
{
    int numCities,i;
    printf("Enter the number of cities(max%d):",MAX_CITIES);
    scanf("%d",&numCities);
    if(numCities<=0||numCities>MAX_CITIES)
    {
        printf("Invalid number of cities:\n");
        return 1;
    }
    int cities[MAX_CITIES];
    printf("Enter the distances between cities:\n");
    for(i=0;i<numCities;i++)
    {
        printf("Distance from city1 to city %d: ",i+2);
        scanf("%d",&cities[i]);
    }
    int minCost=INT_MAX;
    permute(cities,1,numCities,&minCost);
    printf("\nMinimum Cost:%d\n",minCost);
```

```
    return 0;
```

```
}
```

Output

```
D:\DAA Lab\Lab Prgm 2 Travelling salesman prblm.exe"
Enter the number of cities(max10):4
Enter the distances between cities:
Distance from city1 to city 2: 10
Distance from city1 to city 3: 15
Distance from city1 to city 4: 20
Distance from city1 to city 5: 25

Minimum Cost:70

Process returned 0 (0x0)  execution time : 21.317 s
Press any key to continue.
```

3) Write a program to perform Knapsack Problem using Greedy Solution.**Code**

```
#include<stdio.h>
typedef struct
{
    int weight;
    int value;
    double valuePerWeight;
}Item;
void fractionalKnapsack(Item items[],int n,int capacity)
{
    int i,j;
    for(i=0;i<n;i++)
    {
        items[i].valuePerWeight=(double)items[i].value/items[i].weight;
    }
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(items[j].valuePerWeight<items[j+1].valuePerWeight)
            {
                Item temp=items[j];
                items[j]=items[j+1];
                items[j+1]=temp;
            }
        }
    }
}
```

```
        }
    }

    double totalValue=0.0;
    for(int i=0;i<n && capacity>0;i++)
    {
        if(items[i].weight<=capacity)
        {
            totalValue+=items[i].value;
            capacity-=items[i].weight;
        }
        else
        {
            double fraction=(double)capacity/items[i].weight;
            totalValue+=fraction*items[i].value;
            capacity=0;
        }
    }

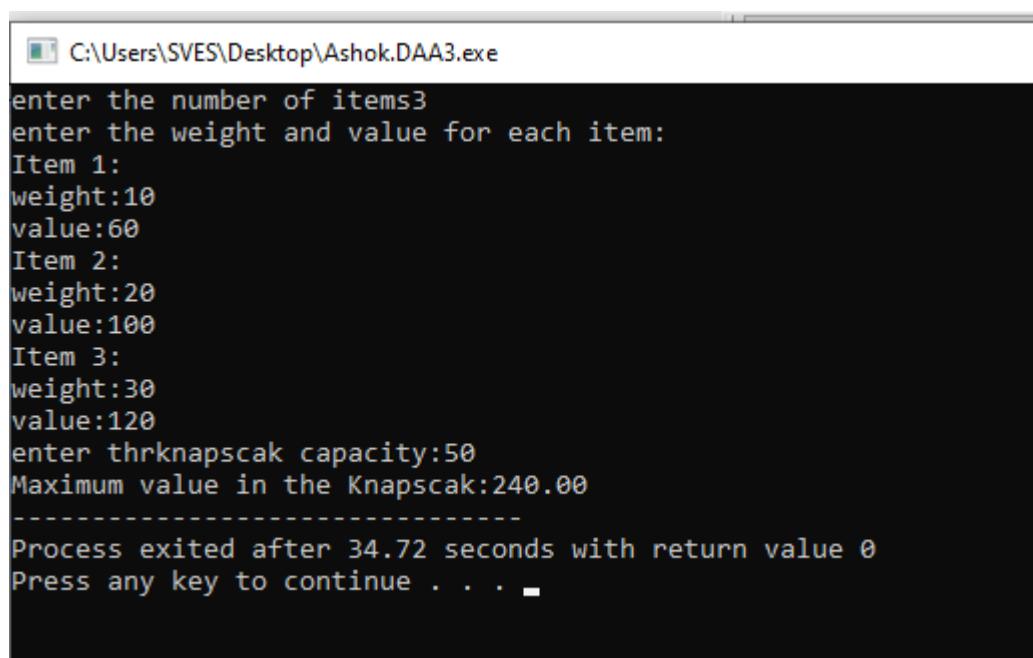
    printf("Maximum value in the Knapsack:%.2f",totalValue);
}

int main()
{
    int i, n,capacity;
    printf("enter the number of items");
    scanf("%d",&n);
    Item items[n];
    printf("enter the weight and value for each item:\n");
    for(i=0;i<n;i++)

```

```
{  
    printf("Item %d:\n",i+1);  
    printf("weight:");  
    scanf("%d",&items[i].weight);  
    printf("value:");  
    scanf("%d",&items[i].value);  
}  
  
printf("enter thrknapsack capacity:");  
scanf("%d",&capacity);  
fractionalKnapsack(items,n,capacity);  
return 0;  
}
```

Output



```
C:\Users\SVES\Desktop\Ashok.DAA3.exe  
enter the number of items3  
enter the weight and value for each item:  
Item 1:  
weight:10  
value:60  
Item 2:  
weight:20  
value:100  
Item 3:  
weight:30  
value:120  
enter thrknapsack capacity:50  
Maximum value in the Knapscak:240.00  
-----  
Process exited after 34.72 seconds with return value 0  
Press any key to continue . . .
```

4) Write program to implement the DFS algorithm for a graph.**Code**

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 100
typedef struct
{
    int vertices[MAX_VERTICES];
    int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
    int numVertices;
}
Graph;
void initGraph(Graph* g, int numVertices)
{
    int i,j;
    g->numVertices = numVertices;
    for (i = 0; i<numVertices; i++)
    {
        g->vertices[i] = 0;
        for (j = 0; j <numVertices; j++)
        {
            g->adjacencyMatrix[i][j] = 0;
        }
    }
}
void addEdge(Graph* g, int start, int end)
{
    g->adjacencyMatrix[start][end] = 1;
    g->adjacencyMatrix[end][start] = 1;
}
void DFS(Graph* g, int startVertex, bool visited[])
{
    int i;
    visited[startVertex] = true;
    printf("%d ", startVertex);
    for (i = 0; i < g->numVertices; i++)
    {
        if (g->adjacencyMatrix[startVertex][i] == 1 && !visited[i])
        {
            DFS(g, i, visited);
        }
    }
}
```

```
        }
    }
}

void performDFS(Graph* g, int startVertex)
{
    bool visited[MAX_VERTICES] = {false};
    printf("DFS traversal starting from vertex %d: ", startVertex);
    DFS(g, startVertex, visited);
}

int main()
{
    int i,j;
    Graph g;
    int numVertices, startVertex;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    initGraph(&g, numVertices);
    printf("Enter the adjacency matrix:\n");
    for( i = 0; i<numVertices; i++)
    {
        for (j = 0; j <numVertices; j++)
        {
            scanf("%d", &g.adjacencyMatrix[i][j]);
        }
    }
    printf("Enter the starting vertex for DFS: ");
    scanf("%d", &startVertex);
    performDFS(&g, startVertex);
    return 0;
}
```

Output

```
C:\Users\SVES\Desktop\Ashok.DAA4.exe
Enter the number of vertices: 4
Enter the adjacency matrix:
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
Enter the starting vertex for DFS: 0
DFS traversal starting from vertex 0: 0 1 2 3
-----
Process exited after 34.19 seconds with return value 0
Press any key to continue . . .
```

5) Write program to implement the BFS algorithm for a graph.**Code**

```
#include<stdio.h>
#include<stdbool.h>
#define MAX_VERTICES 100
typedef struct
{
    int vertices[MAX_VERTICES];
    int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
    int numVertices;
}Graph;
void initGraph(Graph*g,intnumVertices)
{
    int i,j;
    g->numVertices=numVertices;
    for(i=0;i<numVertices;i++)
    {
        g->vertices[i]=0;
        for( j=0;j<numVertices;j++)
        {
            g->adjacencyMatrix[i][j]=0;
        }
    }
}
void addEdge(Graph*g,intstart,int end)
{
```

```
g->adjacencyMatrix[start][end]=1;
g->adjacencyMatrix[start][end]=1;
}

void BFS(Graph*g,intstartVertex)
{
    bool visited[MAX_VERTICES]={false};
    int queue[MAX_VERTICES];
    int front=0,rear=0;
    visited[startVertex]=true;
    queue[rear++]=startVertex;
    while(front<rear)
    {
        int i;
        int currentVertex=queue[front++];
        printf("%d",currentVertex);
        for(i=0;i<g->numVertices;i++)
        {
            if(g->adjacencyMatrix[currentVertex][i]==1&&!visited[i])
            {
                visited[i]=true;
                queue[rear++]=i;
            }
        }
    }
}

int main()
{
```

```
int i,j;  
Graph g;  
int numVertices,startVertex;  
printf("Enter the number of vertices: ");  
scanf("%d",&numVertices);  
initGraph(&g,numVertices);  
printf("Enter the adjacency matrix:\n");  
for (i=0;i<numVertices;i++)  
{  
    for(j=0;j<numVertices;j++)  
    {  
        scanf("%d",&g.adjacencyMatrix[i][j]);  
    }  
}  
printf("enter the starting vertex for BFS: ");  
scanf("%d",&startVertex);  
printf("BFS traversal starting from vertex %d:",startVertex);  
BFS(&g,startVertex);  
return 0;  
}
```

Output

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

Code

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

```
typedef struct
{
    int min;
    int max;
}MinMax;

MinMax findMinMax(int arr[], int low, int high)
{
    MinMax result, leftResult, rightResult;
    int mid;
    if(low==high)
    {
        result.min=arr[low];
        result.max=arr[low];
        return result;
    }
    if(high==low+1)
    {
        if(arr[low]<arr[high])
        {
            result.min=arr[low];
            result.max=arr[high];
        }
        else
        {
            result.min=arr[high];
            result.max=arr[low];
        }
    }
}
```

```
        return result;  
    }  
    mid=(low+high)/2;  
    leftResult=findMinMax(arr,low,mid);  
    rightResult=findMinMax(arr,mid+1,high);  
    result.min=(leftResult.min<rightResult.min)?leftResult.min:rightResult.m  
in;  
    result.max=(leftResult.max>rightResult.max)?leftResult.max:rightResult.  
max;  
    return result;  
}  
  
int main()  
{  
    int n ,i;  
    printf("enter the number of element in the array:");  
    scanf("%d",&n);  
  
    int arr[n];  
    printf("enter the elemnet of the array:\n");  
    for( i=0;i<n;i++)  
    {  
        scanf("%d",&arr[i]);  
    }  
    MinMax result=findMinMax(arr,0,n-1);  
    printf("Minmum value in the array:%\n", result.min);  
    printf("Maximum value in the array:%d\n",result.max);  
    return 0;  
}
```

Output

```
[1] "D:\A DAA lab\MinMax using divide and conquer.exe"
enter the number of element in the array:5
enter the elemnet of the array:
12 5 23 8 18
Minimum value in the array:
Maximum value in the array:23

Process returned 0 (0x0)  execution time : 21.172 s
Press any key to continue.
```

7) Write a test program to implement Divide and Conquer Strategy for Quick sort algorithm .

Code

```
#include<stdio.h>

void swap(int*a, int*b)

{
    int temp=*a;
    *a=*b;
    *b=temp;
}

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

```
void quickSort(int arr[],int low,int high)
{
    if(low<high)
    {
        int pi=partition(arr,low,high);
        quickSort(arr,low,pi-1);
        quickSort(arr,pi+1,high);
    }
}
int main()
{
    int n,i;
    printf("Enter the number of element in the array:");
    scanf("%d",&n);
    int arr[n];
    printf("Enetr the element of the array:\n");
    for( i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    quickSort(arr,0,n-1);
    printf("Sorted array using Quick Sort:\n");
    for(i=0;i<n;i++)
    {
        printf("%d\n",arr[i]);
    }
    return 0;
```

}

Output

```
D:\A DAA lab\quick sort using divide and conquer.exe"
Enter the number of element in the array:5
Enetr the element of the array:
12 5 23 8 18
Sorted array using Quick Sort:
5
8
12
18
23

Process returned 0 (0x0)  execution time : 24.320 s
Press any key to continue.
```

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

Code

```
#include<stdio.h>

void merge(int arr[],int left,int mid,int right)
{
    int i,j,k;
    int n1=mid-left+1;
    int n2=right-mid;
    int leftArr[n1],rightArr[n2];
    for(i=0;i<n1;i++)
    {
        leftArr[i]=arr[left+i];
    }
    for(j=0;j<n2;j++)
    {
        rightArr[j]=arr[mid+1+j];
    }
    i=0,j=0,k=left;
    while(i<n1 && j<n2)
    {
        if(leftArr[i]<=rightArr[j])
        {
            arr[k++]=leftArr[i++];
        }
        else
```

```
{  
    arr[k++]=rightArr[j++];  
}  
}  
while(i<n1)  
{  
    arr[k++]=leftArr[i++];  
}  
while(i<n2)  
{  
    arr[k++]=rightArr[j++];  
}  
}  
void mergeSort(int arr[],int left,int right)  
{  
    if(left<right)  
{  
        int mid=left+(right-left)/2;  
        mergeSort(arr,left,mid); mergeSort(arr,mid+1,right);  
        merge(arr,left,mid,right);  
    }  
}  
int main()  
{  
    int n,i,j;  
    printf("Enter the number of element in the array:");  
    scanf("%d",&n);
```

```
int arr[n];
printf("Enter the element of array:\n");
for(i=0;i<n;i++)
{
    scanf("%d",&arr[i]);
}
mergeSort(arr,0,n-1);
printf("Sorted array using Merge Sort:\n");
for(i=0;i<n;i++)
{
    printf("%d\t",arr[i]);
}
return 0;
}
```

Output

```
D:\A DAA lab\MergeSort.exe"
Enter the number of element in the array:5
Enter the element of array:
12 5 23 8 18
Sorted array using Merge Sort:
5     8      12      18      23
Process returned 0 (0x0)  execution time : 20.535 s
Press any key to continue.
```

SECTION B

- 1) Write C program that accepts the vertices and edges for a graph and stores it as an adjacency matrix.**

Code

```
#include <stdio.h>
#define MAX_VERTICES 100
typedef struct
{
    int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
    int numVertices;
}
Graph;
void initGraph(Graph* g, int numVertices)
{
    int i,j;
    g->numVertices = numVertices;
    for (i = 0; i<numVertices; i++)
    {
        for (j = 0; j <numVertices; j++)
        {
            g->adjacencyMatrix[i][j] = 0;
        }
    }
}
void addEdge(Graph* g, int start, int end)
{
    g->adjacencyMatrix[start][end] = 1;
}
void displayGraph(Graph* g)
{
    int i,j;
    printf("Adjacency Matrix:\n");
    for (i = 0; i< g->numVertices; i++)
    {
        for (j = 0; j < g->numVertices; j++)
        {
            printf("%d ", g->adjacencyMatrix[i][j]);
        }
        printf("\n");
    }
}
```

```
        }
    }
int main()
{
    int i,j;
    Graph g;
    int numVertices, numEdges;
printf("Enter the number of vertices: ");
scanf("%d", &numVertices);
initGraph(&g, numVertices);
printf("Enter the number of edges: ");
scanf("%d", &numEdges);
printf("Enter the edges (start vertex end vertex):\n");
for (i = 0; i<numEdges; i++)
{
    int start, end;
scanf("%d %d", &start, &end);
addEdge(&g, start, end);
}
displayGraph(&g);
return 0;
}
```

Output

```
S:\lab1b.exe
Enter the number of vertices: 5
Enter the number of edges: 7
Enter the edges (start vertex end vertex):
0 1
0 2
1 2
1 3
2 3
3 4
4 0
Adjacency Matrix:
0 1 1 0 0
0 0 1 1 0
0 0 0 1 0
0 0 0 0 1
1 0 0 0 0
Process returned 0 (0x0)  execution time : 64.363 s
Press any key to continue.
```

2) Implement function to print In-Degree and to display that adjacency matrix.**Code**

```
#include<stdio.h>
#define MAX_VERTICES 100
typedef struct
{
    int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
    int numVertices;
}Graph;

void initGraph(Graph*g,intnumVertices)
{
    int i,j;
    g->numVertices=numVertices;
    for(i=0;i<numVertices;i++)
    {
        for( j=0;j<numVertices;j++)
        {
            g->adjacencyMatrix[i][j]=0;
        }
    }
}

void addEdge(Graph*g,intstart,int end)
{
    g->adjacencyMatrix[start][end]=1;
}

void displayGraph(Graph*g)
{
    int i,j;

    printf("Adjacency Matrix:\n");
    for(i=0;i<g->numVertices;i++)
    {
        for( j=0;j<g->numVertices;j++)
        {
            printf("%d",g->adjacencyMatrix[i][j]);
        }
        printf("\n");
    }
}
```

```
}

void calculateDegrees(Graph*g)
{
int i,j;
printf("Vertex\t In-Degree\t Out-Degree\n");
for(i=0;i<g->numVertices;i++)
{
    int inDegree=0, outDegree=0;
    for( j=0;j<g->numVertices;j++)
    {
        inDegree += g->adjacencyMatrix[j][i];
        outDegree += g->adjacencyMatrix[i][j];
    }
    printf("%d\t%d\t%d\n", i, inDegree,outDegree);
}
}

int main()
{
Graph g;
int i,j;
int numVertices,numEdges;
printf("Enter the number of vertices:");
scanf("%d",&numVertices);
initGraph(&g,numVertices);
printf("Enter the number og edges:");
scanf("%d",&numEdges);
printf("Enter the edges (start vertex end vertex):\n");
for( i=0;i<numEdges;i++)
{
    int start,end;
    scanf("%d %d",&start,&end);
    addEdge(&g,start,end);
}
displayGraph(&g);
calculateDegrees(&g);
return 0;
}
```

Output

```
D:\A DAA lab\Part-B, P2.exe"
Enter the number of vertices:5
Enter the number og edges:7
Enter the edges (start vertex end vertex):
0 1
0 2
1 2
1 3
2 3
3 4
4 0
Adjacency Matrix:
01100
00110
00010
00001
10000
Vertex   In-Degree      Out-Degree
0        1              2
1        1              2
2        2              1
3        2              1
4        1              1

Process returned 0 (0x0)  execution time : 49.186 s
Press any key to continue.
```

3) Write program to implement backtracking algorithm for solving problems like N queens.

Code

```
#include<stdio.h>
#include<stdbool.h>
#define MAX_N 10
void printSolution(int board[MAX_N][MAX_N],int N)
{
    int i,j;
    printf("Solution for N-Queens Problem:\n");
    for(i=0;i<N;i++)
    {
        for( j=0;j<N;j++)
        {
            printf("%c",board[i][j]?'Q':'-');
        }
        printf("\n");
    }
}
bool isSafe(int board[MAX_N][MAX_N],int row,int col,int N)
{
    int i,j;
    for(i=0;i<col;i++)
    if(board[row][i])
        return false;
    for(i=row,j=col;i>=0&&j>=0;i--,j--)

```

```
if(board[i][j])
    return false;
for(i=row,j=col;i<N&&j>=0;i++,j--)
    if(board[i][j])
        return false;
    return true;
}
bool solveNQueensUtil(int board[MAX_N][MAX_N],int col,int N)
{
    int i,j;

    if(col>=N)
        return true;
    for(i=0;i<N;i++)
    {
        if(isSafe(board,i,col,N))
        {
            board[i][col]=1;
            if(solveNQueensUtil(board,col+1,N))
                return true;
            board[i][col]=0;
        }
    }
    return false;
}
void solveNQueens(int N)
{
```

```
int board[MAX_N][MAX_N]={ {0} };  
if(!solveNQueensUtil(board,0,N))  
printf("Solution does not exists.\n");  
else  
printSolution(board,N);  
}  
int main()  
{  
int N;  
printf("Enter the value of N for N-Queens: ");  
scanf("%d",&N);  
solveNQueens(N);  
return 0;  
}
```

Output

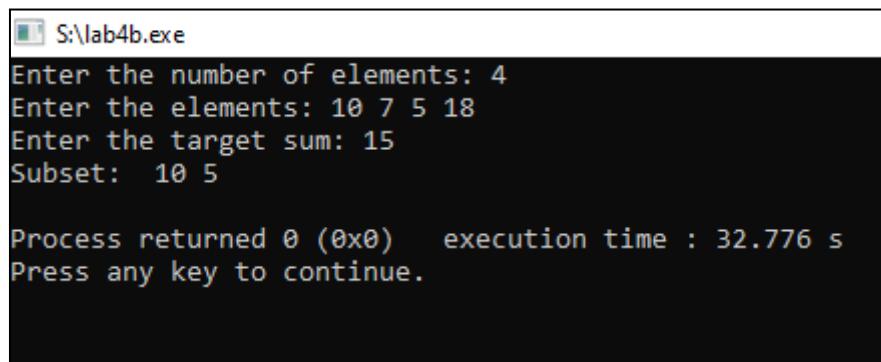
```
"D:\DAA Lab\Part - B, Prgm 3.exe"  
Enter the value of N for N-Queens: 4  
Solution for N-Queens Problem:  
--Q-  
Q---  
---Q  
-Q--  
  
Process returned 0 (0x0)  execution time : 2.673 s  
Press any key to continue.
```

4) Write a program to implement the backtracking algorithm for the sum of subsets problems.**Code**

```
#include <stdio.h>
#include <stdbool.h>
void printSubset(int subset[], int size)
{
    int i,j;
    printf("Subset: ");
    for (i = 0; i< size; i++)
    {
        printf("%d ", subset[i]);
    }
    printf("\n");
}
void sumOfSubsetsUtil(int weights[], int targetSum, int n, int subset[], int
subsetSize, int sum, int index)
{
    int i;
    if (sum == targetSum)
    {
        printSubset(subset, subsetSize);
        return;
    }
    for (i = index; i< n; i++)
    {
        if (sum + weights[i] <= targetSum)
        {
            subset[subsetSize] = weights[i];
            sumOfSubsetsUtil(weights, targetSum, n, subset, subsetSize + 1, sum
+weights[i], i + 1);
        }
    }
}
void sumOfSubsets(int weights[], int targetSum, int n)
{
    int subset[n];
    sumOfSubsetsUtil(weights, targetSum, n, subset, 0, 0, 0);
}
int main()
```

```
{  
    int n,i, targetSum;  
    printf("Enter the number of elements: ");  
    scanf("%d", &n);  
    int weights[n];  
    printf("Enter the elements: ");  
    for (i = 0; i< n; i++)  
    {  
        scanf("%d", &weights[i]);  
    }  
    printf("Enter the target sum: ");  
    scanf("%d", &targetSum);  
    sumOfSubsets(weights, targetSum, n);  
    return 0;  
}
```

Output



```
S:\lab4b.exe  
Enter the number of elements: 4  
Enter the elements: 10 7 5 18  
Enter the target sum: 15  
Subset: 10 5  
  
Process returned 0 (0x0)  execution time : 32.776 s  
Press any key to continue.
```

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

Code

```
#include <stdio.h>

#define MAX_ITEMS 100

int max(int a, int b)

{

    return (a > b) ? a : b;

}

int knapsack(int values[], int weights[], int n, int capacity)

{

    int i,w;

    int dp[MAX_ITEMS + 1][capacity + 1];

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

    {

        for (w = 0; w <= capacity; w++)

        {

            if (i == 0 || w == 0)

            {

                dp[i][w] = 0;

            } else if (weights[i - 1] <= w)

            {

                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i-1][w]);

            }

            else

            {

                dp[i][w] = dp[i - 1][w];

            }

        }

    }

}
```

```
        }
    }
}

return dp[n][capacity];
}

int main()
{
    int i;
    int n, capacity;
    printf("Enter the number of items: ");
    scanf("%d", &n);

    int values[MAX_ITEMS], weights[MAX_ITEMS];
    printf("Enter the values of items:\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &values[i]);
    }

    printf("Enter the weights of items:\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &weights[i]);
    }

    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &capacity);

    int result = knapsack(values, weights, n, capacity);

    printf("Maximum value in the knapsack: %d\n", result);

    return 0;
}
```

Output

```
S:\lab5b.exe
Enter the number of items: 4
Enter the values of items:
2 3 1 4
Enter the weights of items:
3 4 2 5
Enter the capacity of the knapsack: 5
Maximum value in the knapsack: 4

Process returned 0 (0x0)  execution time : 35.989 s
Press any key to continue.
```

- 6) Write a program to implement Dynamic Programming algorithm for the Optimal Binary Search Tree.**

Code

```
#include <stdio.h>

#define MAX_KEYS 100

int optimalBST(int keys[], int freq[], int n){

    int i,j,len,r;

    int dp[MAX_KEYS + 1][MAX_KEYS + 1];

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

        for (j = 0; j <= n; j++)      {

            dp[i][j] = 0;
        }
    }

    for (len = 1; len<= n; len++)  {

        for ( i = 0; i<= n - len + 1; i++)      {

            j = i + len - 1;

            dp[i][j] = MAX_KEYS;

            for ( r = i; r <= j; r++)      {

                int c = ((r >i) ?dp[i][r - 1] : 0) +
                    ((r <j) ?dp[r + 1][j] : 0) +
                    freq[r];

                if (c <dp[i][j])          {

                    dp[i][j] = c;
                }
            }
        }
    }
}
```

```
        }

    return dp[0][n - 1];

}

int main(){

    int n,i;

    printf("Enter the number of keys: ");

    scanf("%d", &n);

    int keys[MAX_KEYS], freq[MAX_KEYS];

    printf("Enter the keys:\n");

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

        scanf("%d", &keys[i]);

    }

    printf("Enter the frequencies:\n");

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

        scanf("%d", &freq[i]);

    }

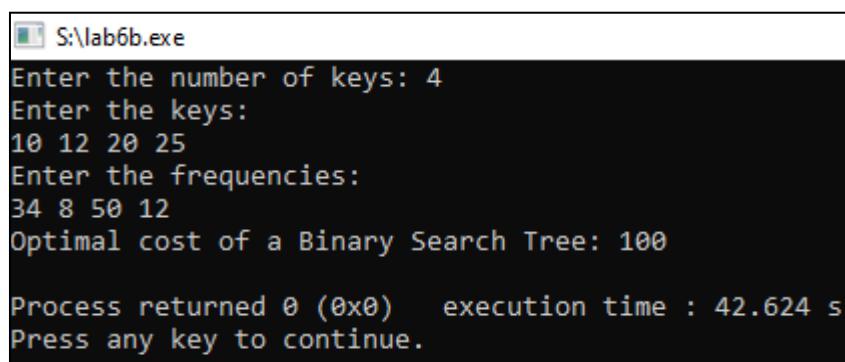
    int result = optimalBST(keys, freq, n);

    printf("Optimal cost of a Binary Search Tree: %d\n", result);

    return 0;

}
```

Output



```
S:\lab6b.exe
Enter the number of keys: 4
Enter the keys:
10 12 20 25
Enter the frequencies:
34 8 50 12
Optimal cost of a Binary Search Tree: 100

Process returned 0 (0x0)  execution time : 42.624 s
Press any key to continue.
```

7) Write a program that implement Prim's algorithm to generate minimum cost spanning tree.

Code

```
#include <stdio.h>
#include<stdbool.h>
#include <limits.h>
#define MAX_VERTICES 100
int minKey(int key[], bool mstSet[], int V)
{
    int v;
    int min = INT_MAX, min_index;
    for ( v = 0; v < V; v++)
    {
        if (mstSet[v] == false && key[v] < min)
        {
            min = key[v];
            min_index = v;
        }
    }
    return min_index;
}
void printMST(int parent[], int
graph[MAX_VERTICES][MAX_VERTICES], int V)
{
    int i;
    printf("Edge \tWeight\n");
    for (i = 1; i < V; i++)

```

```
{  
printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);  
}  
}  
void primMST(int graph[MAX_VERTICES][MAX_VERTICES], int V)  
{  
    int i, count;  
    int parent[V];  
    int key[V];  
    bool mstSet[V];  
    for (i = 0; i < V; i++)  
    {  
        key[i] = INT_MAX;  
        mstSet[i] = false;  
    }  
    key[0] = 0;  
    parent[0] = -1; for (count = 0; count < V - 1; count++)  
    {  
        int v;  
        int u = minKey(key, mstSet, V);  
        mstSet[u] = true;  
        for (v = 0; v < V; v++)  
        {  
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])  
            {  
                parent[v] = u;  
                key[v] = graph[u][v];  
            }  
        }  
    }  
}
```

```
        }
    }
}

printMST(parent, graph, V);
}

int main()
{
    int V, i, j;
    printf("Enter the number of vertices: ");
    scanf("%d", &V);

    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the adjacency matrix for the graph:\n");
    for (i = 0; i < V; i++)
    {
        for (j = 0; j < V; j++)
        {
            scanf("%d", &graph[i][j]);
        }
    }

    primMST(graph, V);

    return 0;
}
```

Output

```
S:\lab7b.exe
Enter the number of vertices: 5
Enter the adjacency matrix for the graph:
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
Edge    Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5

Process returned 0 (0x0)  execution time : 51.167 s
Press any key to continue.
```

- 8) Write a program that implements Kruskal's algorithm to generate minimum cost spanning tree .**

Code

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
typedef struct
{
    int start;
    int end;
    int weight;
}
Edge;
typedef struct
{
    Edge edges[MAX_VERTICES * MAX_VERTICES];
    int numEdges;
}
Graph;
void initGraph(Graph* g)
{
    g->numEdges = 0;
}
void addEdge(Graph* g, int start, int end, int weight)
{
    g->edges[g->numEdges].start = start;
```

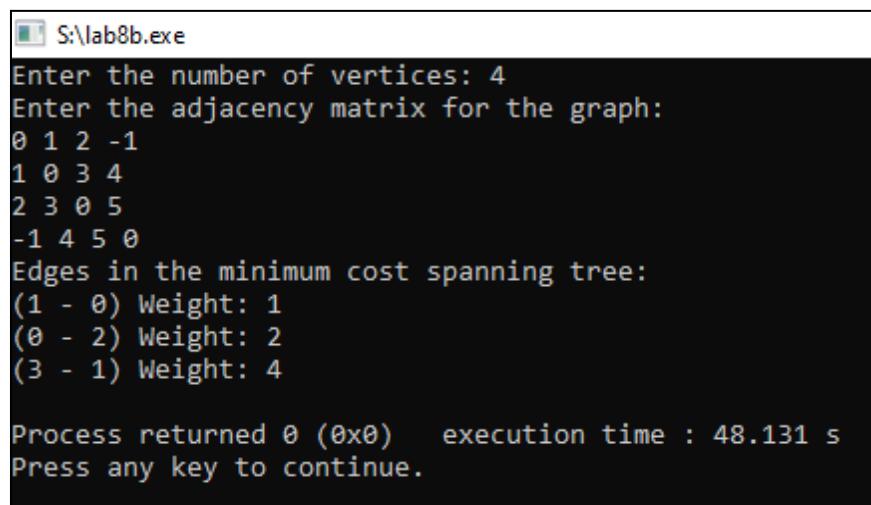
```
g->edges[g->numEdges].end = end;  
g->edges[g->numEdges].weight = weight;  
g->numEdges++;  
}  
  
int compareEdges(const void* a, const void* b)  
{  
    return ((Edge*)a)->weight - ((Edge*)b)->weight;  
}  
  
int find(int parent[], int i)  
{  
    if (parent[i] == -1)  
        return i;  
    return find(parent, parent[i]);  
}  
  
void Union(int parent[], int x, int y)  
{  
    int xset = find(parent, x);  
    int yset = find(parent, y);  
    parent[xset] = yset;  
}  
  
void kruskalMST(Graph* g)  
{  
    int v;  
    Edge result[g->numEdges];  
    int e = 0;  
    int i = 0;  
    qsort(g->edges, g->numEdges, sizeof(g->edges[0]), compareEdges);
```

```
int parent[MAX_VERTICES];
for (v = 0; v < MAX_VERTICES; v++)
    parent[v] = -1;
while (e < MAX_VERTICES - 1 &&i< g->numEdges)
{
    Edge nextEdge = g->edges[i++];
    int x = find(parent, nextEdge.start);
    int y = find(parent, nextEdge.end);
    if (x != y)
    {
        result[e++] = nextEdge;
        Union(parent, x, y);
    }
}
printf("Edges in the minimum cost spanning tree:\n");
for (i = 0; i < e; i++)
{
    printf("(%d - %d) Weight: %d\n", result[i].start, result[i].end,
    result[i].weight);
}
int main()
{
    int i,j;
    Graph g;
    initGraph(&g);
    int V;
```

```
printf("Enter the number of vertices: ");
scanf("%d", &V);

int graph[MAX_VERTICES][MAX_VERTICES];
printf("Enter the adjacency matrix for the graph:\n");
for (i = 0; i < V; i++)
{
    for (j = 0; j < V; j++)
    {
        scanf("%d", &graph[i][j]);
        if (graph[i][j] != -1)
        {
            addEdge(&g, i, j, graph[i][j]);
        }
    }
}
kruskalMST(&g);
return 0;
}
```

Output



```
S:\lab8b.exe
Enter the number of vertices: 4
Enter the adjacency matrix for the graph:
0 1 2 -1
1 0 3 4
2 3 0 5
-1 4 5 0
Edges in the minimum cost spanning tree:
(1 - 0) Weight: 1
(0 - 2) Weight: 2
(3 - 1) Weight: 4

Process returned 0 (0x0)  execution time : 48.131 s
Press any key to continue.
```