```c
//Program -1 – 8 Queen
#define N 8
#include <stdbool.h>
#include <stdio.h>

int count=0;

int printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++) {
        count++;
        for (int j = 0; j < N; j++){
            count++;
            printf(" %d ", board[i][j]);
        }count++;

        printf("\n");
    }count++;
    return count;
}

bool isSafe(int board[N][N], int row, int col)
{
    int i, j;


    for (i = 0; i < col; i++){
        count++;
        if (board[row][i]){
            count++;
            return false;
        }

    }count++;



    for (i = row, j = col; i >= 0 && j >= 0; i--, j--){
        count++;
        if (board[i][j]){
            count++;
            return false;
        }

    }count++;
```

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

    }count++;

    return true;
}


bool solveNQUtil(int board[N][N], int col)
{

    if (col >= N){
        count++;
        return true;
    }



    for (int i = 0; i < N; i++) {
        count++;
        if (isSafe(board, i, col)) {
            count++;

            board[i][col] = 1;
            count++;

            if (solveNQUtil(board, col + 1)){
                count++;
                return true;
            }

            board[i][col] = 0;
            count++;
        }
    }

    return false;
}


bool solveNQ()
{
    int board[N][N] = { { 0, 0, 0, 0 },
```

```
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 } };


    if (solveNQUtil(board, 0) == false) {
        printf("Solution does not exist");
        return false;
    }
    printSolution(board);
    return true;
}

int main()
{
    solveNQ();
    printf("time complexity: %d",count);
    return 0;
}
```

```
PS C:\c_prg\daa_prg\day_4> gcc e8qun_1.c
PS C:\c_prg\daa_prg\day_4> ./a.exe
 1  0  0  0  0  0  0  0
 0  0  0  0  0  0  1  0
 0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  1
 0  1  0  0  0  0  0  0
 0  0  0  1  0  0  0  0
 0  0  0  0  0  1  0  0
 0  0  1  0  0  0  0  0
time complexity: 7670
PS C:\c_prg\daa_prg\day_4>
```

```
//Program -2 - Floyd warshal
#include <stdio.h>


#define V 4


#define INF 99999

int count=0;

void printSolution(int dist[][V]);
```

```
void floydWarshall(int dist[][V])
{
int i, j, k;
    for (k = 0; k < V; k++) {
    count++;
        for (i = 0; i < V; i++) {
            count++;
            for (j = 0; j < V; j++) {
                count++;
                if (dist[i][k] + dist[k][j] < dist[i][j]){
                    count++;
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }count++;
        }count++;
    }count++;

    printSolution(dist);
    count++;
}

void printSolution(int dist[][V])
{
    printf(
        "The following matrix shows the shortest distances"
        " between every pair of vertices \n");
    for (int i = 0; i < V; i++) {
        count++;
        for (int j = 0; j < V; j++) {
            count++;
            if (dist[i][j] == INF){
                count++;
                printf("%7s", "INF");
            }

            else{
                count++;
                printf("%7d", dist[i][j]);
            }
        }count++;
        printf("\n");
    }count++;
}

// driver's code
int main()
{
```

```
    int graph[V][V] = { { 0, 5, INF, 10 },
                        { INF, 0, 3, INF },
                        { INF, INF, 0, 1 },
                        { INF, INF, INF, 0 } };

    floydWarshall(graph);
    printf("Time complexity: %d",count);
    return 0;
}
```

```
PS C:\c_prg\daa_prg\day_4> gcc flyd_warshl_2.c
PS C:\c_prg\daa_prg\day_4> ./a.exe
The following matrix shows the shortest distances between every pair of vertices
      0      5      8      9
    INF      0      3      4
    INF    INF      0      1
    INF    INF    INF      0
Time complexity: 150
PS C:\c_prg\daa_prg\day_4>
```

```
//Program -3 -- Knapsack
#include <stdio.h>
int main()
{
    int capacity, no_items, cur_weight, item,count=0;
    int used[10];
    float total_profit;
    int i;
    int weight[10];
    int value[10];

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

    printf("Enter the number of items: ");
    scanf("%d", &no_items);

    printf("Enter the weight and value of %d item: \n", no_items);
    for (i = 0; i < no_items; i++)
    {
        count++;
        printf("Weight[%d]:\t", i);
        scanf("%d", &weight[i]);
        printf("Value[%d]:\t", i);
```

```
        scanf("%d", &value[i]);
    }count++;

    for (i = 0; i < no_items; ++i){
        count++;
        used[i] = 0;
    }count++;


    cur_weight = capacity;
    count++;
    while (cur_weight > 0)
    {
        count++;
        item = -1;
        count++;
        for (i = 0; i < no_items; ++i){
            count++;
            if ((used[i] == 0) &&((item == -1) || ((float) value[i] /
weight[i] > (float) value[item] / weight[item]))){
                count++;
                item = i;
                count++;
            }
        }count++;


        used[item] = 1;
        count++;
        cur_weight -= weight[item];
        count++;
        total_profit += value[item];
        count++;
        if (cur_weight >= 0){
            count++;
            printf("Added object %d (%d Rs., %dKg) completely in the bag.
Space left: %d.\n", item + 1, value[item], weight[item], cur_weight);
        }

        else
        {
            count++;
            int item_percent = (int) ((1 + (float) cur_weight / weight[item])
* 100);
            count++;
            printf("Added %d%% (%d Rs., %dKg) of object %d in the bag.\n",
item_percent, value[item], weight[item], item + 1);
            total_profit -= value[item];
```

```
            count++;
            total_profit += (1 + (float)cur_weight / weight[item]) *
value[item];
            count++;
        }
    }count++;

    printf("Filled the bag with objects worth %.2f Rs.\n", total_profit);
    printf("Time complexity: %d",count);
}
```

```
Enter the capacity of knapsack: 100
Enter the number of items: 4
Enter the weight and value of 4 item:
Weight[0]:      40
Value[0]:       80
Weight[1]:      30
Value[1]:       70
Weight[2]:      20
Value[2]:       50
Weight[3]:      30
Value[3]:       80
Added object 4 (80 Rs., 30Kg) completely in the bag. Space left: 70.
Added object 3 (50 Rs., 20Kg) completely in the bag. Space left: 50.
Added object 2 (70 Rs., 30Kg) completely in the bag. Space left: 20.
Added 50% (80 Rs., 40Kg) of object 1 in the bag.
Filled the bag with objects worth 240.00 Rs.
Time complexity: 79
PS C:\c_prg\daa_prg\day_4>
```

```
//Program  -4 - Travelling salesman

#include <stdio.h>
int matrix[25][25], visited_cities[10], limit, cost = 0;
int tc=0;
int tsp(int c)
{
 int count, nearest_city = 999;
 int minimum = 999, temp;
 for(count = 0; count < limit; count++)
 {
    tc++;
 if((matrix[c][count] != 0) && (visited_cities[count] == 0))
 {
    tc++;
    if(matrix[c][count] < minimum)
 {
    tc++;
    minimum = matrix[count][0] + matrix[c][count];
```

```
    tc++;
 }
 temp = matrix[c][count];
 tc++;
 nearest_city = count;
 tc++;
 }
 }tc++;

 if(minimum != 999)
 {
    tc++;
    cost = cost + temp;
    tc++;
 }
 return nearest_city;
}

void minimum_cost(int city)
{
 int nearest_city;
 visited_cities[city] = 1;
 tc++;
 printf("%d ", city + 1);
 nearest_city = tsp(city);
 tc++;
 if(nearest_city == 999)
 {
    tc++;
    nearest_city = 0;
    tc++;
    printf("%d", nearest_city + 1);
    cost = cost + matrix[city][nearest_city];
    tc++;
    return;
 }
 tc++;
 minimum_cost(nearest_city);
}

int main()
{
 int i, j;
 printf("Enter Total Number of Cities:\t");
 scanf("%d", &limit);
 printf("\nEnter Cost Matrix\n");
 for(i = 0; i < limit; i++)
 {
```

```
    tc++;
 printf("\nEnter %d Elements in Row[%d]\n", limit, i + 1);
 for(j = 0; j < limit; j++)
 {
    tc++;
 scanf("%d", &matrix[i][j]);
 }tc++;
 tc++;
 visited_cities[i] = 0;
 }tc++;
 printf("\nEntered Cost Matrix\n");
 for(i = 0; i < limit; i++)
 {tc++;
 printf("\n");
 for(j = 0; j < limit; j++)
 {tc++;
 printf("%d ", matrix[i][j]);
 }tc++;
 }tc++;
 printf("\n\nPath:\t");
 minimum_cost(0);
 printf("\n\nMinimum Cost: \t");
 printf("%d\n", cost);
 printf("Time complexity: %d",tc);
 return 0;
}
```

```
Enter Total Number of Cities:   3

Enter Cost Matrix

Enter 3 Elements in Row[1]
2 6 9

Enter 3 Elements in Row[2]
3 7 11

Enter 3 Elements in Row[3]
1 5 8

Entered Cost Matrix

2 6 9
3 7 11
1 5 8

Path:   1 3 2 1

Minimum Cost:   17
Time complexity: 75
```

```c
//Program -5 – Minimum spanning tree
#include <stdio.h>
#include <limits.h>

#define V 6
int count=0;

int minKey(int key[], int mstSet[]) {
    int min = INT_MAX, min_index;
    int v;
    for (v = 0; v < V; v++){
        count++;
        if (mstSet[v] == 0 && key[v] < min){
            count++;
            min = key[v], min_index = v;
            count++;
        }
    }count++;

    return min_index;
}

int printMST(int parent[], int n, int graph[V][V]) {
    int i;
    printf("Edge   Weight\n");
    for (i = 1; i < V; i++){
        count++;
        printf("%d - %d    %d \n", parent[i], i, graph[i][parent[i]]);
    }count++;
}

void primMST(int graph[V][V]) {
    int parent[V];
    int key[V], i, v, count;
    int mstSet[V];


    for (i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = 0;


    key[0] = 0;
    parent[0] = -1;


    for (count = 0; count < V - 1; count++) {
        count++;
```

```
        int u = minKey(key, mstSet);
        count++;
        mstSet[u] = 1;
        count++;

        for (v = 0; v < V; v++){
            count++;
            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]){
                count++;
                parent[v] = u, key[v] = graph[u][v];
                count++;
            }
        }count++;

    }count++;
    printf("Time complexity: %d\n",count);
    count++;
    printMST(parent, V, graph);

}

int main() {

    int graph[V][V] = { { 0, 2, 0, 1, 4, 0 }, { 1, 0, 3, 3, 0, 7 },
            { 0, 3, 0, 5, 0, 7}, { 1, 3, 5, 0, 9, 0 }, { 4, 0, 0, 9, 0, 0 },
            { 0, 7, 8, 0, 0, 0} };

    primMST(graph);
    printf("Time complexity: %d",count);
    return 0;
}
```

```
PS C:\c_prg\daa_prg\day_4> gcc minspn_5.c
PS C:\c_prg\daa_prg\day_4> ./a.exe
Time complexity: 18
Edge   Weight
0 - 1    1
PS C:\c_prg\daa_prg\day_4>
```

```
//Program -6 - Hamiltonian circuit.

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

```
#define V 5
int count=0;

void printSolution(int path[]);


bool isSafe(int v, bool graph[V][V], int path[], int pos)
{

    if (graph [ path[pos-1] ][ v ] == 0){
        count++;
        return false;
    }



    for (int i = 0; i < pos; i++){
        count++;
        if (path[i] == v){
            count++;
            return false;
        }

    }count++;


    return true;
}


bool hamCycleUtil(bool graph[V][V], int path[], int pos)
{

    if (pos == V)
    {
        count++;

        if ( graph[ path[pos-1] ][ path[0] ] == 1 ){
            count++;
            return true;
        }

        else{
            count++;
            return false;
        }

    }
```

```
    for (int v = 1; v < V; v++)
    {
        count++;

        if (isSafe(v, graph, path, pos))
        {
            path[pos] = v;
            count++;


            if (hamCycleUtil (graph, path, pos+1) == true){
                count++;
                return true;
            }

            count++;
            path[pos] = -1;
        }
    }


    return false;
}


bool hamCycle(bool graph[V][V])
{

    int path[V];
    for (int i = 0; i < V; i++){
        count++;
         path[i] = -1;
         count++;
    }count++;


    path[0] = 0;
    count++;

    if ( hamCycleUtil(graph, path, 1) == false )
    {
        count++;
        printf("\nSolution does not exist");
        return false;
    }

    count++;
    printSolution(path);
    return true;
```

```
}


void printSolution(int path[])
{
    printf ("Solution Exists:\n"
            " Following is one Hamiltonian Cycle \n");
    for (int i = 0; i < V; i++){
        count++;
        printf(" %d ", path[i]);
    }count++;



    printf(" %d ", path[0]);
    printf("\n");
}


int main()
{

   bool graph1[V][V] = {{0, 1, 0, 1, 0},
                        {1, 0, 1, 1, 1},
                        {0, 1, 0, 0, 1},
                        {1, 1, 0, 0, 1},
                        {0, 1, 1, 1, 0},
                       };

    // Print the solution
    hamCycle(graph1);


    bool graph2[V][V] = {{0, 1, 0, 1, 0},
                        {1, 0, 1, 1, 1},
                        {0, 1, 0, 0, 1},
                        {1, 1, 0, 0, 0},
                        {0, 1, 1, 0, 0},
                       };


    hamCycle(graph2);
    printf("\nTime complexity: %d",count);
    return 0;
}
```

```
PS C:\c_prg> cd daa_prg
PS C:\c_prg\daa_prg> cd day_4
PS C:\c_prg\daa_prg\day_4> gcc hamcir_6.c
PS C:\c_prg\daa_prg\day_4> ./a.exe
Solution Exists:
 Following is one Hamiltonian Cycle
 0  1  2  4  3  0

Time complexity: 249
PS C:\c_prg\daa_prg\day_4>
```

```c
//Program -7 – Container Loading.
#include<stdio.h>
#include<stdlib.h>

int count=0;

int compare(const void *a, const void *b) {
    int *item1 = (int *)a;
    count++;
    int *item2 = (int *)b;
    count++;
    return (*item2) - (*item1);
    count++;
}

void container_loading(int items[], int n, int container_capacity) {
    int containers = 0,remaining_capacity = container_capacity;
    count++;

    qsort(items, n, sizeof(int), compare);
    count++;

    for (int i = 0; i < n; i++) {
        count++;
        if (remaining_capacity >= items[i]) {
            count++;
            remaining_capacity -= items[i];
            count++;
        }
        else {
            count++;
            containers++;
            count++;
            remaining_capacity = container_capacity - items[i];
            count++;
        }
```

```
    }count++;

    printf("Number of containers used: %d\n", containers + 1);
}

int main() {
    int items[] = {50, 100, 30, 80, 90, 200, 150, 20 };
    int n = sizeof(items) / sizeof(items[0]);
    int container_capacity = 100;
    container_loading(items, n, container_capacity);
    printf("Time complextiy: %d",count);
    return 0;
}
```

```
PS C:\c_prg\daa_prg\day_4> gcc contload_7.c
PS C:\c_prg\daa_prg\day_4> ./a.exe
Number of containers used: 7
Time complextiy: 89
PS C:\c_prg\daa_prg\day_4>
```

```
//Program -8 - Min-Max Sequency..

#include<stdio.h>
int seq(int arr[],int s);

int sort(int arr[],int s){

    int count=0;
    count++;
    for (int i = 0; i<=s; i++)
    {
        count++;
        for (int j = i+1; j<=s; j++)
        {
            count++;
            if(arr[i]>arr[j]){
                count++;
                int temp=arr[i];
                count++;
                arr[i]=arr[j];
                count++;
                arr[j]=temp;
                count++;
            }
```

```
        }count++;
    }count++;

    int res=seq(arr,s);

    return count+res;

}

int seq(int arr[],int s){
    int count=0,mid=(s-0)/2;
    count++;

    printf("pair: \n");
    for(int i=0,j=s;i<mid,j>=mid+1;i++,j--){
        count++;
        printf("%d,%d,",arr[i],arr[j]);
    }count++;

    return count;
}

void main(){
    int count=0,size,res=0;
    count++;

    printf("Enter tot element: ");
    scanf("%d",&size);

    int arr[size];

    printf("Enter the elements: ");
    for (int i = 0; i < size; i++)
    {
        count++;
        scanf("%d",&arr[i]);
    }count++;

    res=sort(arr,size-1)+count;
    printf("\nTime complexity: %d",res);
}
```

```
Enter tot element: 8
Enter the elements: 3
5
-4
1
8
2
0
4
pair:
-4,8,0,5,1,4,2,3,
Time complexity: 118
PS C:\c_prg\daa_prg\day_4>
```