2) PRINTING ALL THE SOLUTIONS OF THE N QUEEN GAME.

ALGORITHM AND TRICK:

(LINK OF EXPLANATION :
Backtracking | N-Queen Problem & Sudoku Solver | DSA-One Course #11
6.1 N Queens Problem using Backtracking

1) **In Order That the queens do not directly attack each other, any pair of queen must not be in same column and same row.**
2) **So we make a decision of placing every queen in different rows and for columns we check whether it is sufficient or not.**
3) **So we go on placing the first queen in each position of the first row and recursively call the same function for placing the next queen.**
4) **Eg :**
   For placing the first queen at each and every position of first row we use for loop:

```
Void function( int  i , ……)
{
     for( int j=0 ; j<n ; j++){
          if( isSafe( i , j )){

          }
     }
}
```

5) **The isSafe function has the moto to check whether the queen is not placed in the column in which some other queen is already present and also is there some queen present diagonally to the position.**

```
Bool isSafe( int i ,int j){
    for( int column=0 ; column<i ; column++){
        if( mat[ column ][ j ]==1){
            Return false ;
        }
    }
}
```

6) **The isSafe function is Very easy to understand. Firstly we have to check the column vertically above in which the queen is placed .If there is already some queen present in that position then we return false.**
   Secondly we have to check the elements diagonally for which we use (i - - , j - -) , (i + +, j + +) , (i - - , j + +) , (i + + , j - -) technique.

```
int row=i,col=j;
  while(row>=0 && col>=0){
    if(mat[row][col]==1){
      return false;
    }
    row--;
    col--;
  }
  row=i,col=j;
  while(row<n && col<n){
```

```
            if(mat[row][col]==1){
                return false;
            }
            row++;
            col++;
        }
        row=i,col=j;
        while(row>=0 && col<n){
            if(mat[row][col]==1){
                return false;
            }
            row--;
            col++;
        }
        row=i,col=j;
        while(row<n && col>=0){
            if(mat[row][col]==1){
                return false;
            }
            row++;
            col--;
        }
```

8) The most important thing is that we have to push the column value in the answer vector if the isSafe function is true and call the function for the next queen but after the function has recursively solved the problem, so we have to pop that value and also make the mat[i][j]=-1 again

CODE FOR THE APPROACH:

```
#include<iostream>
#include<vector>
using namespace std;

vector<vector<int>> answer;

bool isSafe(vector<vector<int>> &mat,int i,int j,int n){
    for(int column=0;column<i;column++){
        if(mat[column][j]==1){
            return false;
        }
    }
    int row=i,col=j;
    while(row>=0 && col>=0){
        if(mat[row][col]==1){
            return false;
        }
        row--;
```

```
        col--;
    }
    row=i,col=j;
    while(row<n && col<n){
        if(mat[row][col]==1){
            return false;
        }
        row++;
        col++;
    }
    row=i,col=j;
    while(row>=0 && col<n){
        if(mat[row][col]==1){
            return false;
        }
        row--;
        col++;
    }
    row=i,col=j;
    while(row<n && col>=0){
        if(mat[row][col]==1){
            return false;
        }
        row++;
        col--;
    }
    return true;
}

void fun(vector<vector<int>> &mat,int i,int n,vector<int> &table){

    if(i==n){
        answer.push_back(table);
        return;
    }
    else{
        for(int j=0;j<n;j++){
            bool status=isSafe(mat,i,j,n);
            if(status==true){
                mat[i][j]=1;
                table.push_back(j);
                fun(mat,i+1,n,table);
                table.pop_back();
                mat[i][j]=0;
            }
        }
    }
}

int main(){
```

```
    int n;
    cout<<"\n Enter the number of queens in the game:";
    cin>>n;
    vector<int> table(0);
    vector<vector<int>> mat(n,vector<int>(n));
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            mat[i][j]=0;
        }
    }
    fun(mat,0,n,table);
    cout<<"\n The solutions are : \n";
    for(int i=0;i<answer.size();i++){
        for(int j=0;j<answer[i].size();j++){
            cout<<answer[i][j]<<" ";
        }
        cout<<endl;
    }
    return 0;
}
```