**Experiment No: 11**

**AIM:** Implementation N Queens problem using Backtracking

**THEORY:**

The N Queen is a classic combinatorial problem which deals with placing N chess queens on an N×N chessboard so that no two queens attack each other. Any Two queens will attack each other if they’re in the same row, column or diagonal. Hence, we must place the N queens in a manner in which they’re in distinct rows, columns and diagonals.

The problem is efficiently solved in programming using a concept called Backtracking.

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

**ALGORITHM:**

**Algorithm** Place(k,i)

//Returns true if a queen can be placed in kth row and ith column, else returns false

//x[] is a global array which holds the solution vector and whose k-1 vales have been initialised

//abs(r) returns the absolute value of r

{

for j:= 1 to k-1 **do**

{

If(x[j] = I or abs(x[j] – i) = abs(j-k)) **then**

return false;

}

return true;

}

**Algorithm** NQueens(k,n)

// Using backtracking, this procedure prints all

// possible placements of n queens on an n x n

// chessboard so that they are not attacking.

{

for i :=1to n do

{

if Place(k,i)then

{

x[k]:=i;

if (k = n) then write (x[1:n]);

else NQueens(k+1,n);

}

}

}

*Time Complexity*

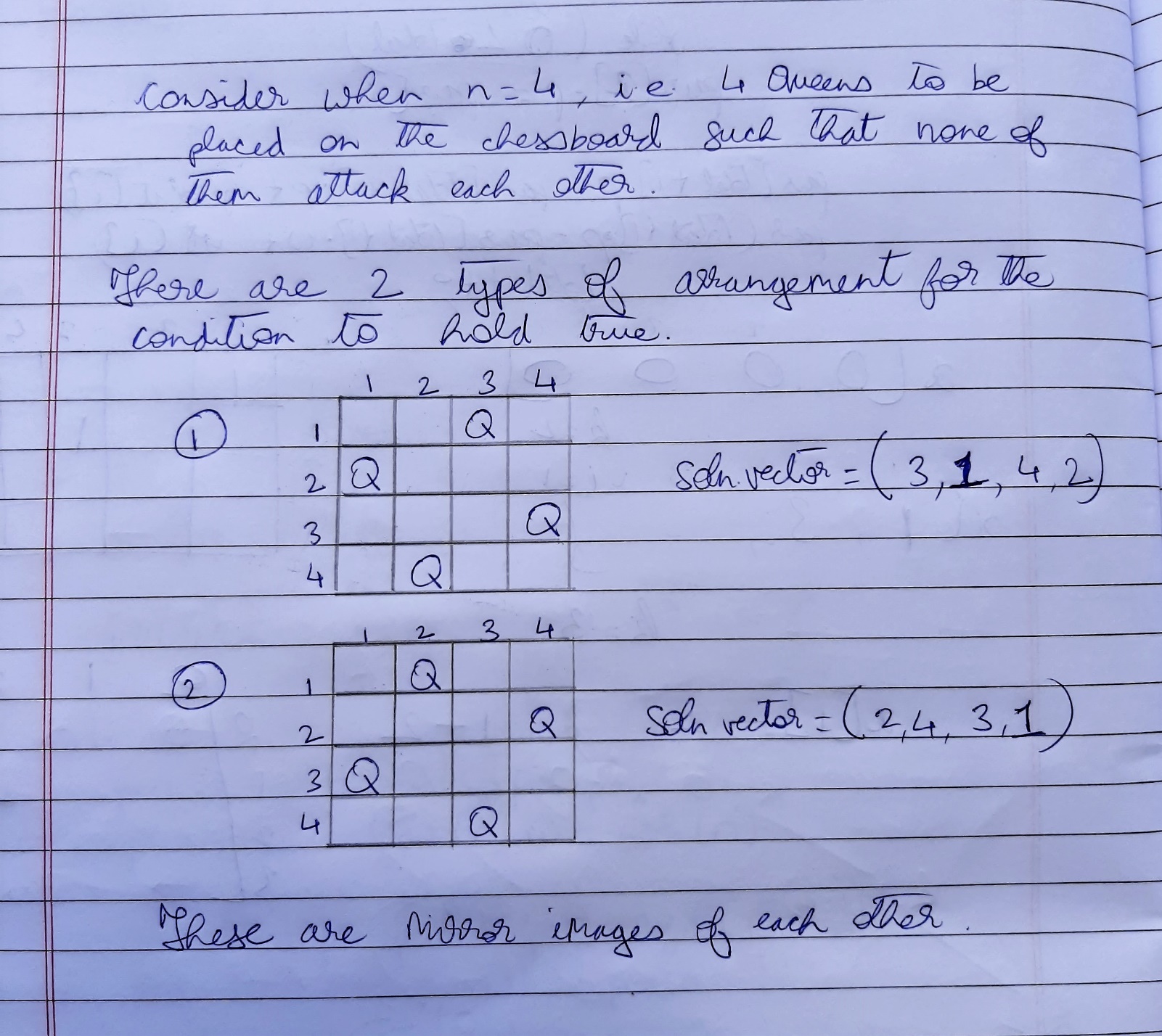
The time complexity is computed by:

T(n) = n\*T(n-1) + O(n^2), which translates to the order of O(n!)

Hence,

• The time complexity of this algorithm is O (n!).

*Problem Tracing*



PROGRAM IMPLEMENTATION:

#include<iostream>

using namespace std;

int \*x; //x is solution vector

void show\_soln(int n)

{

cout<<"( ";

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

cout<<x[i]<<" ";

cout<<")"<<endl;

}

int abs(int num)

{

if(num<0)

num\*=-1;

return num;

}

bool place(int k, int i)

{

for(int j=0;j<=k-1;j++)

if(x[j] == i || (abs(x[j]-i) == k-j) ) //if the queen to be placed

return false;

//is on the same column or diagonal

return true;

} //of prev queen

void Nqueens(int k, int n) //k is row number and i is column number;

{

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

if(place(k,i))

{

x[k] = i;

if(k == n-1)

show\_soln(n);

else

Nqueens(k+1,n);

}

}

int main()

{

int n;

cout<<"Enter number of queens:\n";

cin>>n;

x = new int[n];

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

x[i]=-1; //initialised solution vector to -1

int k=0;

Nqueens(k,n);

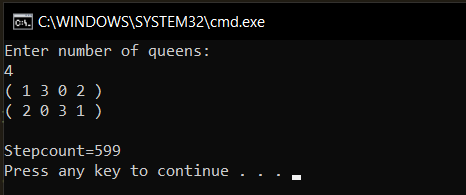
return 0;

}

OUTPUTS:

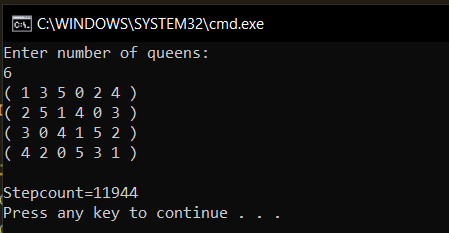
1. When n=4

**Count=599**



1. When n=6

**Count=11944**

****

**Conclusion**:

* **Since T(n) = n\*T(n-1) + O(n^2) approximately being O(n!), the time complexity of this algorithm is O(n!)**
* **Some solutions generated by the algorithm are reflections or rotations of others.**