K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: A1	**Roll No.: 16010123012**
Experiment No.: 08	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Study, Implementation and Analysis of 8-Queens problem.

Objective: To learn the Backtracking strategy of problem solving for 8-Queens problem

CO to be achieved:

Sr. No	Objective
CO 1	Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations.
CO 2	Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.math.utah.edu/~alfeld/queens/queens.html**
4. **http://wwwisl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf**
5. **http://www.slideshare.net/Tech_MX/8-queens-problem-using-back-tracking**
6. **http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html**
7. **http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/**
8. **http://www.hbmeyer.de/backtrack/achtdamen/eight.htm**

Pre Lab/ Prior Concepts:
Data structures, Concepts of algorithm analysis

Historical Profile:
The **N-Queens puzzle** is the problem of placing N queens on an N×N chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

New Concepts to be learned:
Application of algorithmic design strategy to any problem, Backtracking method of problem-solving Vs other methods of problem solving, 8- Queens problem and its applications.

Algorithm N Queens Problem: -

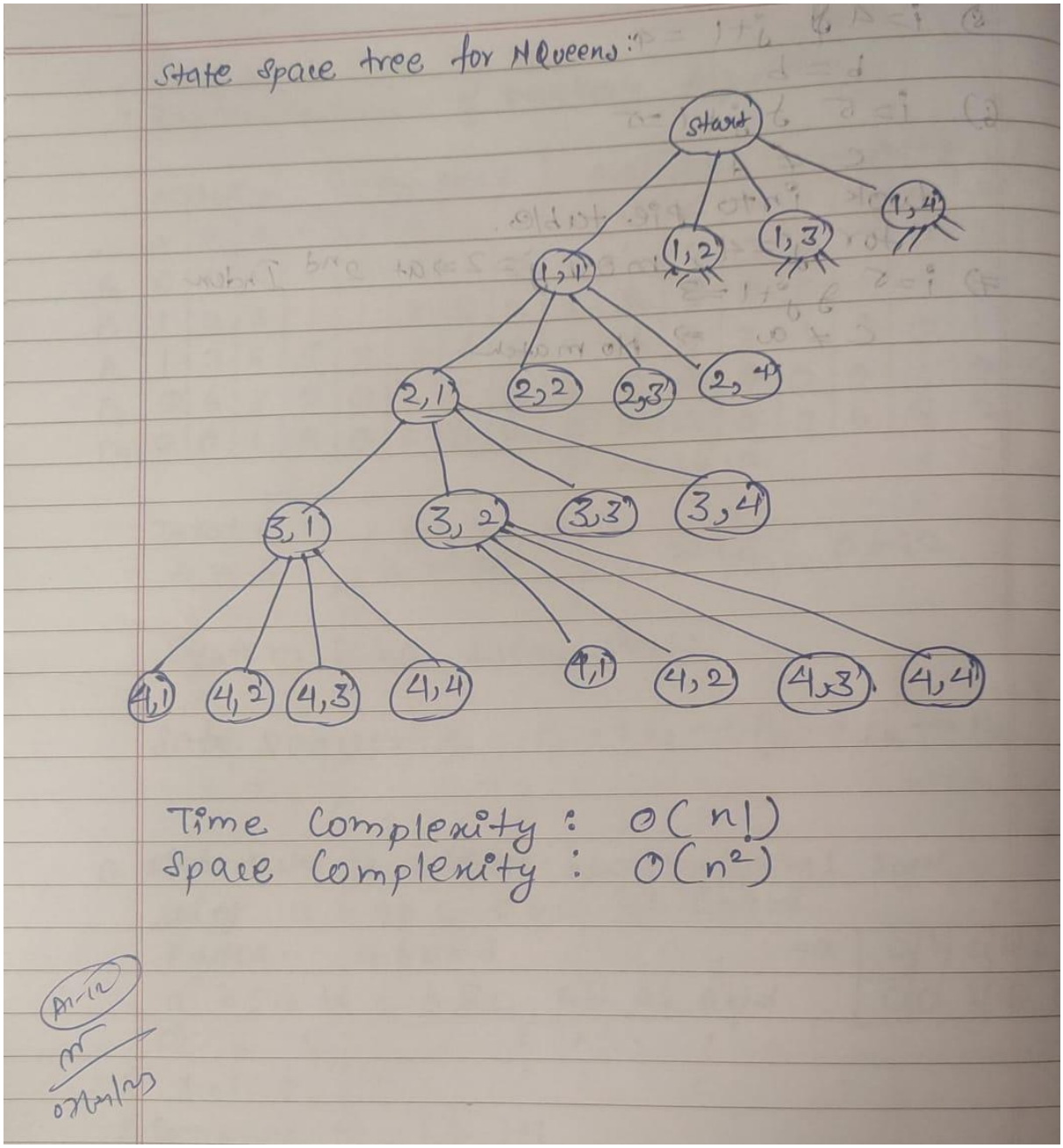
```
void NQueens(int k, int n)
// Using backtracking, this procedure prints all possible placements of n queens on an n X n
chessboard so that they are nonattacking.
{       for (int i=1; i<=n; i++)
      {
            if (Place(k, i))
             {
               x[k] = i;
               if (k==n)
                        for (int j=1;j<=n;j++)          Print  x[j] ;
               else NQueens(k+1, n);
             }
        }
}
```

```
Boolean Place(int k, int i)
// Returns true if a queen can be placed in k^th row and i^th column.  Otherwise it returns false.
// x[] is a global array whose first (k-1) values have been set. abs(r) returns absolute value of
r.
{
for (int j=1; j < k; j++)
        if ((x[j] == i)  // Two in the same column
      || (abs(x[j]-i) == abs(j-k)))                  // or in the same diagonal
          return(false);
return(true);
 }
```

Example 8-Queens Problem:
The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other i.e. no two queens share the same row, column, or diagonal.

Solution Using Backtracking Approach:
The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

State Space tree for N-Queens (Solution):



Implementation (Code):

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int x[8];
bool Place(int k, int i)
{
```

```
    for (int j = 1; j < k; j++)
    {
        if ((x[j] == i) || (abs(x[j] - i) == abs(j - k)))
        {
            return false;
        }
    }
    return true;
}

void NQueens(int k, int n)
{
    for (int i = 1; i <= n; i++)
    {
        if (Place(k, i))
        {
            x[k] = i;
            if (k == n)
            {
                for (int j = 1; j <= n; j++)
                {
                    cout << x[j] << " ";
                }
                cout << endl;
            }
            else
            {
                NQueens(k + 1, n);
            }
        }
    }
}

int main()
{
    int n;
    cout << "Enter the number of queens: ";
    cin >> n;
    NQueens(1, n);
    return 0;
}
```

OUTPUT:

```
Enter the number of queens: 4
2 4 1 3
3 1 4 2


...Program finished with exit code 0
Press ENTER to exit console.
```

```
Enter the number of queens: 5
1 3 5 2 4
1 4 2 5 3
2 4 1 3 5
2 5 3 1 4
3 1 4 2 5
3 5 2 4 1
4 1 3 5 2
4 2 5 3 1
5 2 4 1 3
5 3 1 4 2


...Program finished with exit code 0
Press ENTER to exit console.
```

```
Enter the number of queens: 6
2 4 6 1 3 5
3 6 2 5 1 4
4 1 5 2 6 3
5 3 1 6 4 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Analysis of Backtracking solution:
Time Complexity: O(N!)
Auxiliary Space: $O(N^2)$

CONCLUSION:
I have successfully implemented the N-Queens problem using the backtracking strategy. Through this experiment, I gained a deeper understanding of recursive problem-solving techniques and how backtracking can be effectively applied to constraint satisfaction problems.