Name: Sagar H. Narsingani.

Roll No.: 19BCP114.

Branch: Computer Engineering.

Division: II.

Semester: IV.

Subject: Design and Analysis of Algorithm – Lab (20CP209P).

Lab 6 Assignment

Aim: To Solve Optimization Problems through Backtracking technique and draw state-space diagram.

#problem1: Write a C/C++ program to solve given sudoku puzzle using Backtracking Approach. Analyze its time complexity.

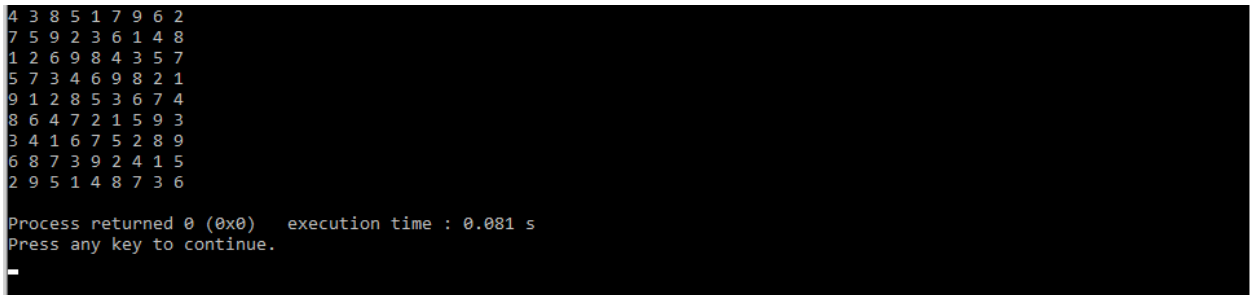
Code:

```
 #include <iostream>

using namespace std;

#define UNASSIGNED 0

#define N 9

bool FindUnassignedLocation(int grid[N][N], int& row, int& col);

bool isSafe(int grid[N][N], int row, int col, int num);

bool SolveSudoku(int grid[N][N])

{

        int row, col;

        if (!FindUnassignedLocation(grid, row, col))

                return true;

        for (int num = 1; num <= 9; num++)

        {

                if (isSafe(grid, row, col, num))

                {

                        grid[row][col] = num;

                        if (SolveSudoku(grid))

                                return true;

                        grid[row][col] = UNASSIGNED;

                }

        }

        return false;
```

```
}
bool FindUnassignedLocation(int grid[N][N], int& row, int& col)
{
        for (row = 0; row < N; row++)
                for (col = 0; col < N; col++)
                        if (grid[row][col] == UNASSIGNED)
                                return true;
        return false;
}
bool UsedInRow(int grid[N][N], int row, int num)
{
        for (int col = 0; col < N; col++)
                if (grid[row][col] == num)
                        return true;
        return false;
}
bool UsedInCol(int grid[N][N], int col, int num)
{
        for (int row = 0; row < N; row++)
                if (grid[row][col] == num)
                        return true;
        return false;
}
bool UsedInBox(int grid[N][N], int boxStartRow,
                        int boxStartCol, int num)
{
        for (int row = 0; row < 3; row++)
                for (int col = 0; col < 3; col++)
                        if (grid[row + boxStartRow][col + boxStartCol] == num)
```

```
                                    return true;
          return false;
}
bool isSafe(int grid[N][N], int row,int col, int num)
{
          return !UsedInRow(grid, row, num)
                    && !UsedInCol(grid, col, num)
                    && !UsedInBox(grid, row - row % 3,
                                                        col - col % 3, num)
                    && grid[row][col] == UNASSIGNED;
}
void printGrid(int grid[N][N])
{
          for (int row = 0; row < N; row++)
          {
                    for (int col = 0; col < N; col++)
                              cout << grid[row][col] << " ";
                    cout << endl;
          }
}
int main()
{
          int grid[N][N] = {
                                    { 0, 3, 0, 0, 1, 0, 0, 6, 0 },
                                    { 7, 5, 0, 0, 3, 0, 0, 4, 8 },
                                    { 0, 0, 6, 9, 8, 4, 3, 0, 0 },
                                    { 0, 0, 3, 0, 0, 0, 8, 0, 0 },
                                    { 9, 1, 2, 0, 0, 0, 6, 7, 4 },
                                    { 0, 0, 4, 0, 0, 0, 5, 0, 0 },
                                    { 0, 0, 1, 6, 7, 5, 2, 0, 0 },
                                        { 6, 8, 0, 0, 9, 0, 0, 1, 5 },

                                        { 0, 9, 0, 0, 4, 0, 0, 3, 0 }

                                };

if (SolveSudoku(grid) == true)

                printGrid(grid);

        else

                cout << "No solution exists";

        return 0;

}
```

Output:



```
4 3 8 5 1 7 9 6 2
7 5 9 2 3 6 1 4 8
1 2 6 9 8 4 3 5 7
5 7 3 4 6 9 8 2 1
9 1 2 8 5 3 6 7 4
8 6 4 7 2 1 5 9 3
3 4 1 6 7 5 2 8 9
6 8 7 3 9 2 4 1 5
2 9 5 1 4 8 7 3 6

Process returned 0 (0x0)   execution time : 0.081 s
Press any key to continue.
```

Time Analysis:

For every unassigned index, there are 9 possible options so the time complexity is **O(9^(n*n))**. The time complexity remains the same as normal algorithm (without backtracking) but there will be some early pruning so the time taken will be much less but the upper bound time complexity remains the same.

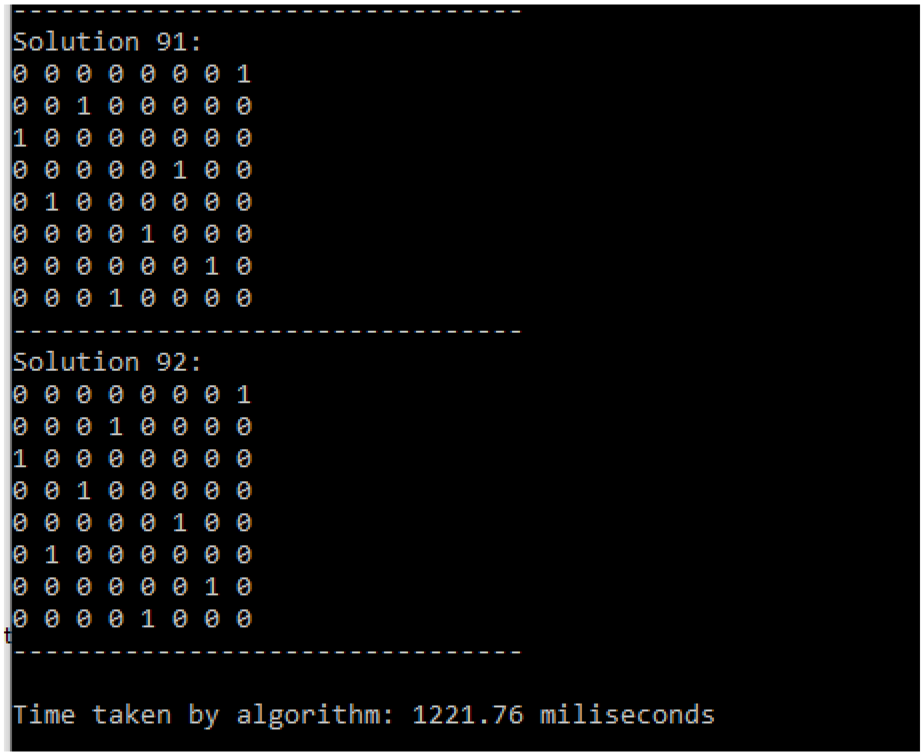
#problem2: Write a C/C++ program to solve 8-Queen's problem. Analyze the time required by algorithm using Backtracking to solve the problem.

Code:

```
#include<iostream>

#include<algorithm>

using namespace std;

bool CanPlace(int x,int y,int b[8][8])

{

   for(int i=0;i<8;i++)

   {

      for(int j=0;j<8;j++)

      {

         if(b[i][j]==1)

         {

            if( x == i || y == j || x-y == i-j || x+y == i+j )

               return false;

         }

      }

   }

   return true;

}

int solution=0;

void solve(int b[8][8],int i)

{

  if(i==8)

  {

    solution++;
```

```
      cout<<"Solution "<<solution<<": "<<endl;
      for(int i=0;i<8;i++)
      {
         for(int j=0;j<8;j++)
          {
            cout<<b[i][j]<<" ";
          }
         cout<<endl;
      }
      cout<<"--------------------------------"<<endl;
      return;
   }
   for(int j=0;j<8;j++)
   {
      if(CanPlace(i,j,b))
      {
         b[i][j]=1;
         solve(b,i+1);
         b[i][j]=0;
      }
   }
}
int main()
{
    int b[8][8];
    for(int k=0;k<8;k++)
     {
        for(int l=0;l<8;l++)
        {
            b[k][l]=0;
        }

     }

    solve(b,0);

    return 0;

}
```

Output:



```
--------------------------------
Solution 91:
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 1 0 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
--------------------------------
Solution 92:
0 0 0 0 0 0 0 1
0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
--------------------------------

Time taken by algorithm: 1221.76 miliseconds
```

Time Analysis:

When we assign a location of the queen in the first column, we have n options, after that, we only have n-1 options as we can't place the queen in the same row as the first queen, then n-2 and so on. Thus, the worst-case complexity is still upper bounded by **O(n!)**.