# Design Analysis and Algorithem, Lab Assignment 11

Bhuvana Kanakam - SE21UCSE035

November 23, 2023

## Question

**Assignment** You are tasked with implementing a solution to the N-Queens problem using the backtracking algorithm. The N-Queens problem is a classical problem in which N queens are to be placed on an N×N chessboard in such a way that no two queens threaten each other. (Take N=4 or 8).

### Python Code

```
def is_safe(board, row, col, n):
    for i in range(row):
        if board[i][col] == 1:
            return False

    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    for i, j in zip(range(row, -1, -1), range(col, n)):
        if board[i][j] == 1:
            return False

    return True

def solve_nqueens_util(board, row, n, solutions):
    if row == n:
        solution = []
        for i in range(n):
            solution.append("".join(["Q" if x == 1 else "." for x in board[i]]))
        solutions.append(solution)
        return

    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 1
            solve_nqueens_util(board, row + 1, n, solutions)

            board[row][col] = 0

def solve_nqueens(n):
    board = [[0 for _ in range(n)] for _ in range(n)]
    solutions = []
    solve_nqueens_util(board, 0, n, solutions)
    return solutions

n = 4
solutions = solve_nqueens(n)
```

```
for i, solution in enumerate(solutions, 1):
    print(f"Solution {i}:\n")
    for row in solution:
        print(row)
    print("\n")
```

## Ouput of the Code

This code defines a function solveNQueens that takes an integer n as input and returns a list of all solutions for the N-Queens problem on an N×N chessboard. The solutions are represented as lists of strings, where each string represents a row on the chessboard, with 'Q' indicating the position of a queen.

```
Solution 1:

.Q..
...Q
Q...
..Q.


Solution 2:

..Q.
Q...
...Q
.Q..
```

## Analysis of the Algorithm:

### Time Complexity:

The time complexity of the backtracking algorithm for the N-Queens problem is not straightforward to express precisely. In the worst case, the algorithm explores all possible configurations of queens on the board. However, the algorithm often prunes large portions of the search space by backtracking when it determines that the current partial solution cannot be extended to a valid solution.

The time complexity is often approximated as O(N!), where N is the number of queens (and the size of the chessboard). This is because, in the worst case, each queen can be placed in N positions in its row, the next queen in N-1 positions (as it cannot attack the first queen), the third queen in N-2 positions, and so on. Therefore, the total number of configurations to be checked is N * (N-1) * (N-2) * ... * 1, which is N!.

### Space Complexity:

The space complexity is primarily determined by the space used for the chessboard. In the provided implementation, the chessboard is represented as a 2D list, which requires O(N square) space. Additionally, the recursion depth of the backtracking function contributes to the space complexity.

The recursion depth is at most N since each recursive call corresponds to placing a queen in a new row. Therefore, the overall space complexity is O(N square + N) = O(N square).

### Backtracking Approach:

- **Decision Space:** At each step, the algorithm makes decisions on where to place the queens. It explores all possible positions for the current queen in the current row.
- **Constraints:** The algorithm enforces constraints to ensure that no two queens threaten each other.
- **Backtracking:** If the current configuration violates the constraints, the algorithm backtracks, undoing the last decision and exploring a different branch of the decision space.
- **Base Case:** The algorithm terminates when a valid placement for all queens is found (base case), and the solution is recorded.