Department of Computer Engineering
Faculty of Engineering
University of Sri Jayewardenepura

Course	Advanced Algorithms
Course Code	CO4352
Title	N-Queens Problem Solver
Project Number	1
Outcomes	• Using algorithm design techniques to solve a given problem • Understanding and implementing optimization techniques
Submission Deadline	-

General Instructions:

- This is an **individual** project.
- Each student must complete and submit their own work.
- Please archive all files into a zip file, upload the zip file into LMS.
- Use the following format when you are naming the zip file: yy_ENG_xxx_Project.zip, yy_ENG_xxx is your index number.

N-Queens Solving Algorithm

1. Introduction

The N-Queens problem is a classic constraint satisfaction and backtracking problem in computer science. The standard version, often called the 8-Queens puzzle, asks you to place 8 queens on a standard 8×8 chessboard so that no two queens attack each other. More generally, in the N-Queens problem we are given an N×N chessboard and must place N queens on the board so that none of the queens share the same row, column, or diagonal.

The rules of the standard N-Queens problem are as follows:

- Each queen must be placed on a distinct row of the board.
- Each queen must be placed on a distinct column of the board.
- No two queens may be placed on the same diagonal (both main and anti-diagonals).

A configuration that respects all of the above constraints is called a valid solution to the N-Queens problem. For a given N, there may be no solution, exactly one solution, or multiple distinct solutions.

Your task in this mini project is to design and implement an efficient algorithm to find valid queen placements for a given value of N, respecting all constraints.

2. Implementing an algorithm to solve the N-Queens Problem

Objective

You are required to implement an algorithm capable of solving the general N-Queens problem for a given N (for example, 8 for the standard 8-Queens puzzle). Your implementation must focus on efficiency. You must analyse the time complexity in your report, and we will benchmark your program on several N values.

At minimum, your program should be able to:

- Determine whether a valid arrangement of N queens exists on an N×N board.
- Produce at least one valid arrangement when solutions exist.
- Enumerate and output all distinct solutions for a given N.

Implementation and Requirements

- Your algorithm should be implemented in either Python or C++.
- The algorithm must be able to solve the N-Queens problem for any N in a reasonable range, for example $4 \leq N \leq 20$.
- Your program must accept a command-line argument specifying the name of the input file.
- Example (Linux):

  ./nqueens_solver input1.txt

  where nqueens_solver is the name of your program and input1.txt is the input text file.

Input format

- The input must be provided as a text file in the root folder of the program. The text file must have the following format:
- First line: a single integer N representing the size of the board and the number of queens to place.
- The text file must not contain any unnecessary characters, additional text, or random spaces beyond what is required for the input format.

Output format

- The solution must be written to a text file in the root folder of the program.
- The output file name must be the same as the input file name with the suffix "_output" added at the end.

  Example:

  If input1.txt is the input file, the program must write the output to input1_output.txt.

- If at least one solution exists, the output file must contain:
  1. First line: the value of N.
  2. Second line: an integer indicating whether you output a single solution or multiple solutions. For example, this can be the number of solutions printed in the file.
  3. From the third line onwards: the board configuration(s). You may use one of the following formats:

  Option A – Q / . representation

  - ➔ Use Q to represent a queen and . to represent an empty square.
  - ➔ For each solution, write N lines, each with N characters separated by a single space.

➔ If you print multiple solutions, separate them by an empty line or a clear separator line.

Option B – column index representation

- Represent a solution by listing the column position of the queen in each row.
- For example, a line 2 4 1 3 for N = 4 means:
  - Row 1 → column 2
  - Row 2 → column 4
  - Row 3 → column 1
  - Row 4 → column 3

Note: You must use exactly one of the above representations (Option A or Option B) and use it consistently throughout the output file.

- If no solution exists for the given N, the output file must contain the exact string: “No Solution”.

3. Generalized N-Queens

In the standard puzzle, N = 8. However, the N-Queens problem is defined for any positive integer N. Some values of N have no solution (for example, N = 2 and N = 3), while most larger values have at least one solution.

You must extend your algorithm to handle arbitrary N values in the allowed range (e.g., 4 to 20). Your program must:

- Read N from the input file.
- Automatically attempt to solve the N-Queens problem for this N without requiring code changes.
- Correctly detect when no solution exists and handle such cases gracefully (by writing "No Solution" to the output file).

In your report, you should discuss:

- How the behaviour of your algorithm changes as N increases.
- For which values of N your program is able to find solutions within a reasonable time limit.
- Any patterns or observations you noticed about the existence of solutions.

4. Efficiency and Time Complexity

The efficiency of your algorithm will be tested. Although the N-Queens problem has an exponential search space, an efficient backtracking algorithm with pruning can solve reasonably large N quickly.

You should describe any optimization techniques you use, such as:

– Early pruning of invalid partial solutions.

– Efficient checks for column and diagonal conflicts.

– Symmetry reductions (e.g., avoiding mirror-image duplicate solutions).

You should aim for your algorithm to handle at least the following within a reasonable time:

- N = 9 almost instantly.
- Larger values such as N = 15 and N = 20 within a few seconds on average (you may define specific performance targets in your report depending on your environment).

In the report, include experimental results for several N values showing:

- Execution time.
- Number of solutions found.

5. Submission Guidelines

Along with all the source files, submit a report (maximum 5 pages including the cover page) which includes the following information:

- Introduction.
- Background and constraints (explain the rules, representation of the board, and constraints).
- Implementation of the algorithm (data structures, backtracking strategy, how you check safety of placements).
- Optimization techniques (any pruning, heuristics, symmetry handling, etc.).
- Challenges faced during implementation and how you addressed them.
- Limitations of your current approach.
- Possible future improvements.

All source files should be clearly structured and well commented. Provide clear instructions on how to compile and run your program, including example commands.

6. Evaluation Criteria

- 70% of the marks will be allocated to the project, including correctness of the N-Queens solver, code quality, use of appropriate algorithms/data structures, and the written report.
- 30% of the marks will be allocated according to the ranking of your project based on the average time it takes to solve a set of N-Queens test instances.
  If your rank is 1st, you will receive 30 marks; if your rank is 30th, you will receive 0 marks, with intermediate ranks scaled proportionally.