COURSE: CP468 – Artificial Intelligence
ASSIGNMENT: TERM PROJECT
DATE: 2024-12-06

GROUP 10: N-QUEENS

Group Members:
- Lucas Duncan (169024504)
- Jack Jarrett (210576140)
- Nick Simbhunauth (210675780)
- Carter Chan (210688220)
- Talal Ahmad (169036802)
- Andre Moodley (169022560)
- Aaron Kim (169053077)
- Ryan Chan (169030535)
- Lucas Huang - (absent from presentation –dropped?)
- Brayden Van De Wyckel -
  (absent from presentation – dropped?)

2. DESIGN CHOICES:

- **Language:** We chose to write this project in C (initially in GO), as these languages were much faster at computing low-level operations compared languages such as Python, which we originally considered. Additionally, it allowed us to make use of specialized compilation flags which could improve our runtime.

- **Algorithm Choice:** We decided to approach this project using a randomized setup implementing a parallelized minimal conflicts algorithm. Notably, seen within the attached zip file, this differs from the original presented in class, by spreading the task of finding the column with min conflicts for each row between multiple concurrent threads.

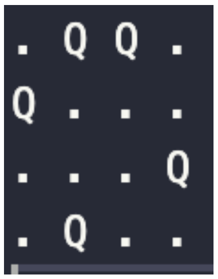
- **Board Design:** We chose to represent the board as a C struct defined as "Board". Board contains the following variables.
  - ☐ The integer "N", representing the board's X and Y dimensions.
  - ☐ A dynamically allocated array of N integers called "Queens", where each index corresponds to a row, and the value at that index represents the column position of the queen in that row.
  - ☐ An array of N atomic integers represented as "rowConflicts" whose (i)th index stores the number of queens within the (i)th row.
  - ☐ Two arrays of 2*N - 1 atomic integers: diag1Conflicts and diag2Conflicts. The (i)th index of diag1Conflicts stores the number of queens on the (i)th diagonal, while the (i)th index of diag2Conflicts stores the number of queens on the (i)th antidiagonal.

- **Input and Output:**
  - **Parameters:**
    - **Board Sizes (boardSizes[]):** An array that specifies the sizes of the N-Queens boards to solve. In the provided code, the boardSizes array contains a single size, which is 1000000. This indicates that the algorithm will attempt to solve the N-Queens problem for a 1,000,000 x 1,000,000 board.
    - **Test Quantity (testQuantity):** The number of tests to perform for averaging the execution time. In the code, this is set to 1, meaning the program will run the N-Queens solution once for each board size in boardSizes. For inputs greater than 1 the program will run multiple iterations of the N-Queens solution based on the input and present the average run time across all iterations.
    - **Number of CPU Cores (numCPU)**: The number of CPU cores to be used for parallel processing. The code automatically detects the number of available cores on the machine using sysconf(_SC_NPROCESSORS_ONLN) and adjusts the number of threads accordingly. If fewer than 1 core is detected, it defaults to 1 core.
    - **Determine Whether Assessing Validity Of Stored State (CheckInput):** Our program has 2 states, checking the validity of a state stored within input.txt and solving the above specified tests. When CheckInput = 1, it does the input check, and when CheckInput = 0, it solves the random states as outlined by the other

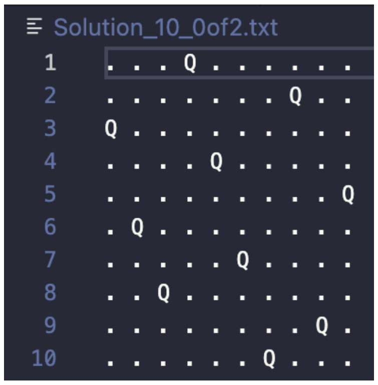
parameters. In the case where CheckInput = 1, we setup the file into the given format and run our validate function to either confirm of deny its a valid solution

-
    - **Whether To Print Solution(printSolution):** When using the case where CheckInput = 0, we solve a given number of NQueens Puzzles, and when printSolution = 0, we simply print time information and steps in the terminal, however when it is set to 1, we instead save each solution as a unique .txt file. Note that we store these solutions in the same format as our input, thus these files get quite large, we recommend turning off this feature for anything about n=10000.

- **Input/ Output Format**
    - The Format we use for input files (for when CheckInput = 1 and input.txt exists) as well as our output format (for when CheckInput = 0 and printSolution = 1), as a text file where any empty squares are represented by periods, and queens are represented by upper case q's, with each square having a space between it. Please see examples below



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



Solution_10_0of2.txt

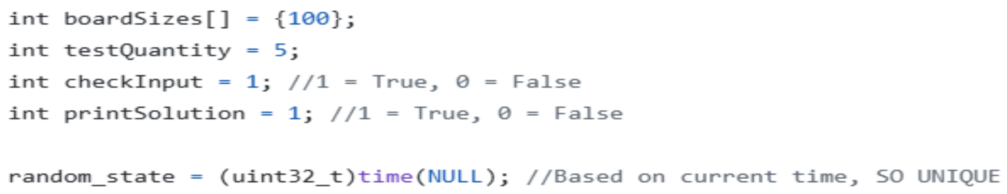
```
. . . Q . . . . . .
. . . . . . . Q . .
Q . . . . . . . . .
. . . . Q . . . . .
. . . . . . . . . Q
. Q . . . . . . . .
. . . . . Q . . . .
. . Q . . . . . . .
. . . . . . . . Q .
. . . . . . Q . . .
```

- **Data Structures Used:**
    - **Atomic Integers:** Given our algorithm runs multiple threads simultaneously, which all require read access to row and diagonal counts, we suffered the potential risk of race conditions and simultaneous read/write conflicts. To address this conflict, we defined globally required variables of the board using Atomic Integers, a variable that ensures only one thread can modify it at once.
    - **Random Generation:** C's Random number generating function rand() is notoriously costly, due to the substantial number of random numbers our program required, we found it more beneficial to implement a more cost-efficient solution XORSHIFT() that uses bit shifted exclusive or to calculate the next random state (as compared to the original combination of shifting and multiplication done by default).
    - **Multi-threading:** We adopted a parallel processing approach to leverage the capabilities of multi-core processors. Splitting conflict columns into smaller groups and distributing

these groups across multiple threads managed by a pool, we efficiently utilize the available CPU cores to speed up computation, particularly for larger board sizes. To ensure concurrent access to shared resources, such as row and diagonal conflict arrays, we utilized atomic operations. This minimizes synchronization overhead while maintaining thread safety. Additionally, by introducing a randomization step to shuffle the column processing order, we reduce deterministic patterns and improve solution effectiveness. This design effectively balances efficiency and scalability, making it a powerful method for solving the N-Queens problem in a timely manner.

3. Installation and Execution

1. Start by downloading a C compiler like Clang or GCC (on mac: brew install gcc)
2. Install git (on mac: brew install git)
3. Clone our repository at by doing "git clone https://github.com/chanr335/468TP" or download the file "n-queens-final.c" available on the same link.
4. Change directory to get into the folder using the command "cd 468TP"
5. From here you can execute one of two modes at a time by modifying the file



```
int boardSizes[] = {100};
int testQuantity = 5;
int checkInput = 1; //1 = True, 0 = False
int printSolution = 1; //1 = True, 0 = False

random_state = (uint32_t)time(NULL); //Based on current time, SO UNIQUE
```

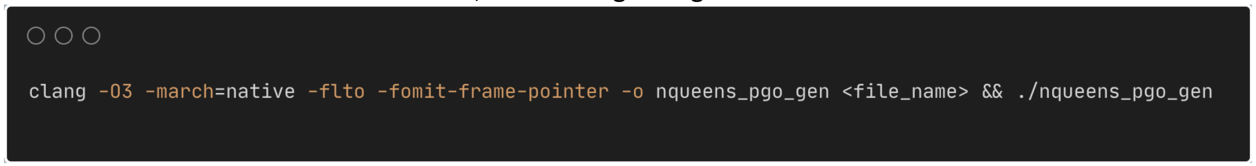
Solve Random States:

- Set point_sizes to the list of n's you want to solve (IE (10, 100, 1000)
- Set test_quantity to the amount of tests you want to run on each (IE 5)
- Set checkInput = 0
- Set printSolution (1 if you want solutions are .txt files, 0 otherwise)

Checking State Stored At "input.txt"

- Set checkInput = 1
- Ignore all other arguments

6. Now from within the command line run, after saving changes to file



```
clang -O3 -march=native -flto -fomit-frame-pointer -o nqueens_pgo_gen <file_name> && ./nqueens_pgo_gen
```

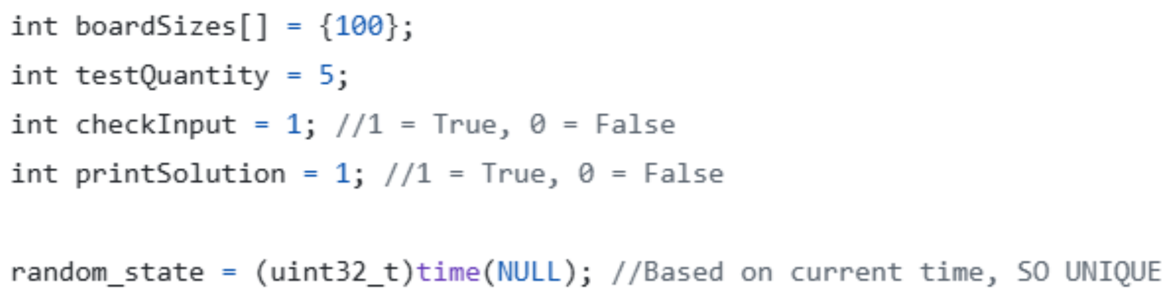
Compiler Optimization Flags Explanation

- -03 enables the highest optimization for the compiler.
- -march=native generates code optimized for the architecture of the machine where the compilation is taking place.
- -flto enables Link-time optimization to perform optimizations across the entire program at link time versus just within individual source files.
- -fomit-frame-pointer tells the compiler not to maintain the frame pointer in registers if not needed.

7. This will run the settings you have chosen

4. Results: (Random + Checking Input)

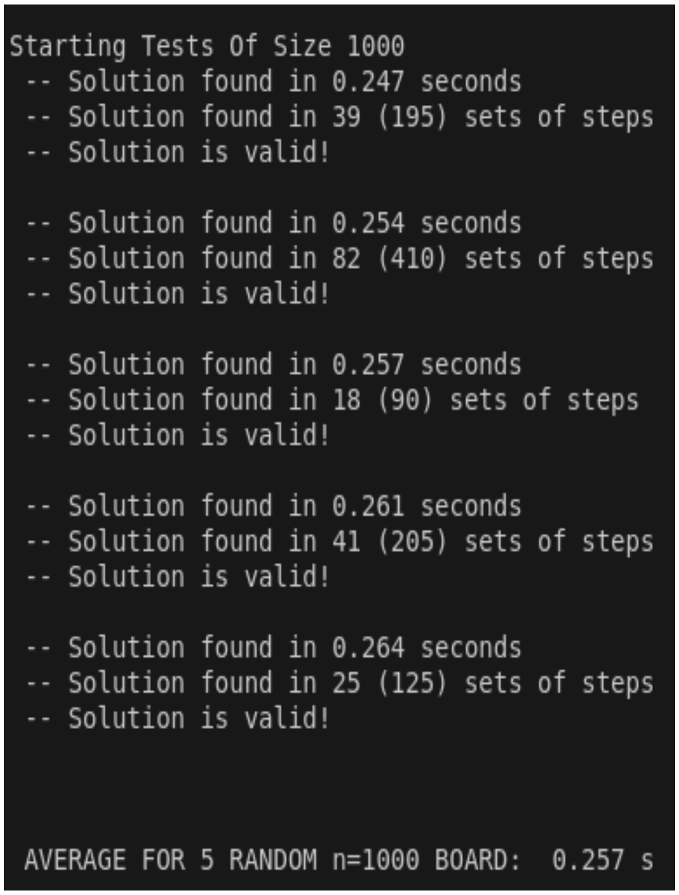
Below we run each size (100, 1,000, 10,000, 100,000, 1,000,000) 5 times on a randomly initialized placed board to get an average runtime. To run this simply set the following attributes within the main method.



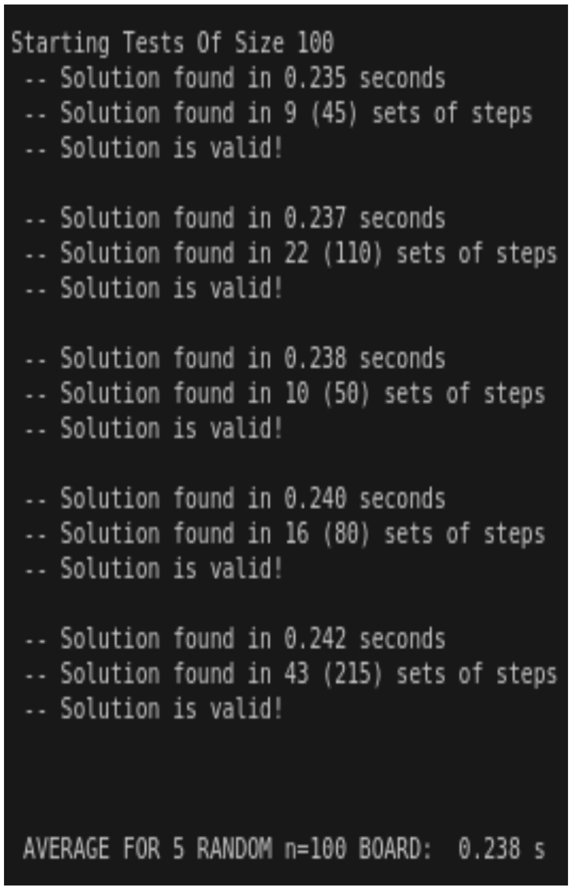
```
int boardSizes[] = {100};
int testQuantity = 5;
int checkInput = 1; //1 = True, 0 = False
int printSolution = 1; //1 = True, 0 = False

random_state = (uint32_t)time(NULL); //Based on current time, SO UNIQUE
```

Due to the high cost of Parallelization (unnecessary for low N) the time to compute required to complete low values of N is slower relative to their size than higher values of N.

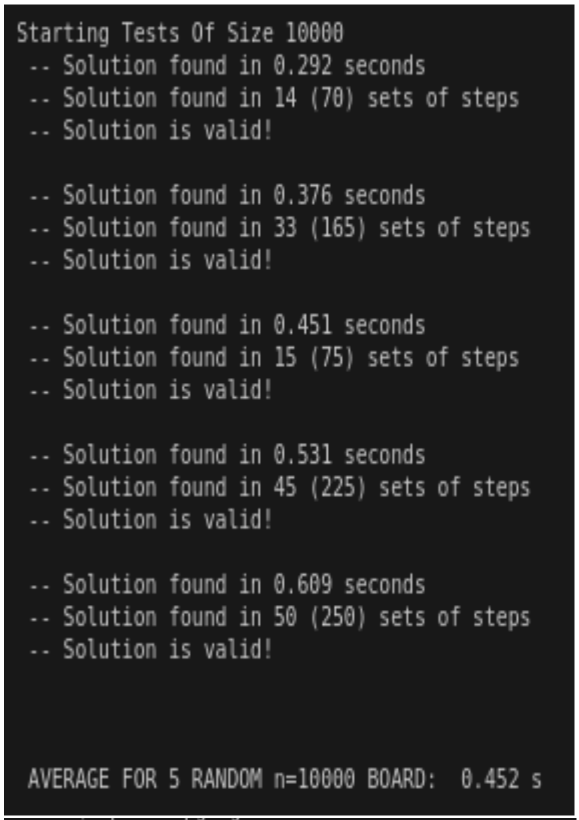
Within the terminal output below, the time required to solve, the number of step, and the confirmation that the solution is valid by running it within the Validate function (not included in solve time).



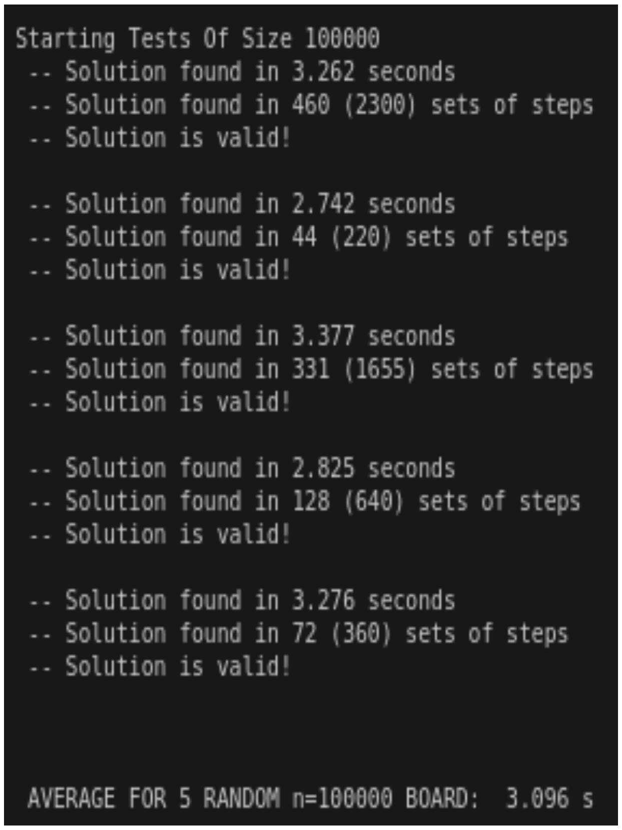
```
Starting Tests Of Size 1000
 -- Solution found in 0.247 seconds
 -- Solution found in 39 (195) sets of steps
 -- Solution is valid!

 -- Solution found in 0.254 seconds
 -- Solution found in 82 (410) sets of steps
 -- Solution is valid!

 -- Solution found in 0.257 seconds
 -- Solution found in 18 (90) sets of steps
 -- Solution is valid!

 -- Solution found in 0.261 seconds
 -- Solution found in 41 (205) sets of steps
 -- Solution is valid!

 -- Solution found in 0.264 seconds
 -- Solution found in 25 (125) sets of steps
 -- Solution is valid!



 AVERAGE FOR 5 RANDOM n=1000 BOARD:  0.257 s
```



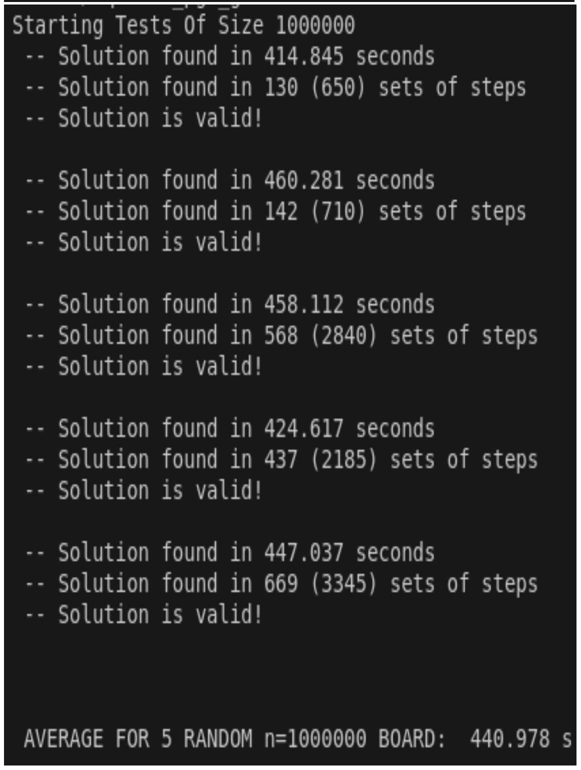
```
Starting Tests Of Size 100
 -- Solution found in 0.235 seconds
 -- Solution found in 9 (45) sets of steps
 -- Solution is valid!

 -- Solution found in 0.237 seconds
 -- Solution found in 22 (110) sets of steps
 -- Solution is valid!

 -- Solution found in 0.238 seconds
 -- Solution found in 10 (50) sets of steps
 -- Solution is valid!

 -- Solution found in 0.240 seconds
 -- Solution found in 16 (80) sets of steps
 -- Solution is valid!

 -- Solution found in 0.242 seconds
 -- Solution found in 43 (215) sets of steps
 -- Solution is valid!



 AVERAGE FOR 5 RANDOM n=100 BOARD:  0.238 s
```



```
Starting Tests Of Size 10000
 -- Solution found in 0.292 seconds
 -- Solution found in 14 (70) sets of steps
 -- Solution is valid!

 -- Solution found in 0.376 seconds
 -- Solution found in 33 (165) sets of steps
 -- Solution is valid!

 -- Solution found in 0.451 seconds
 -- Solution found in 15 (75) sets of steps
 -- Solution is valid!

 -- Solution found in 0.531 seconds
 -- Solution found in 45 (225) sets of steps
 -- Solution is valid!

 -- Solution found in 0.609 seconds
 -- Solution found in 50 (250) sets of steps
 -- Solution is valid!



 AVERAGE FOR 5 RANDOM n=10000 BOARD:  0.452 s
```

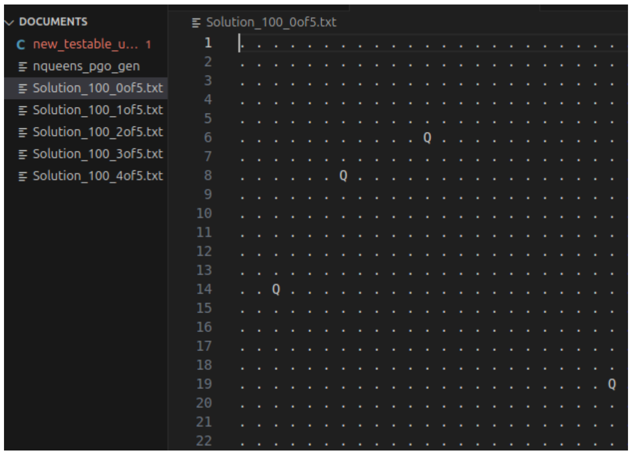


```
Starting Tests Of Size 100000
 -- Solution found in 3.262 seconds
 -- Solution found in 460 (2300) sets of steps
 -- Solution is valid!

 -- Solution found in 2.742 seconds
 -- Solution found in 44 (220) sets of steps
 -- Solution is valid!

 -- Solution found in 3.377 seconds
 -- Solution found in 331 (1655) sets of steps
 -- Solution is valid!

 -- Solution found in 2.825 seconds
 -- Solution found in 128 (640) sets of steps
 -- Solution is valid!

 -- Solution found in 3.276 seconds
 -- Solution found in 72 (360) sets of steps
 -- Solution is valid!



 AVERAGE FOR 5 RANDOM n=100000 BOARD:  3.096 s
```



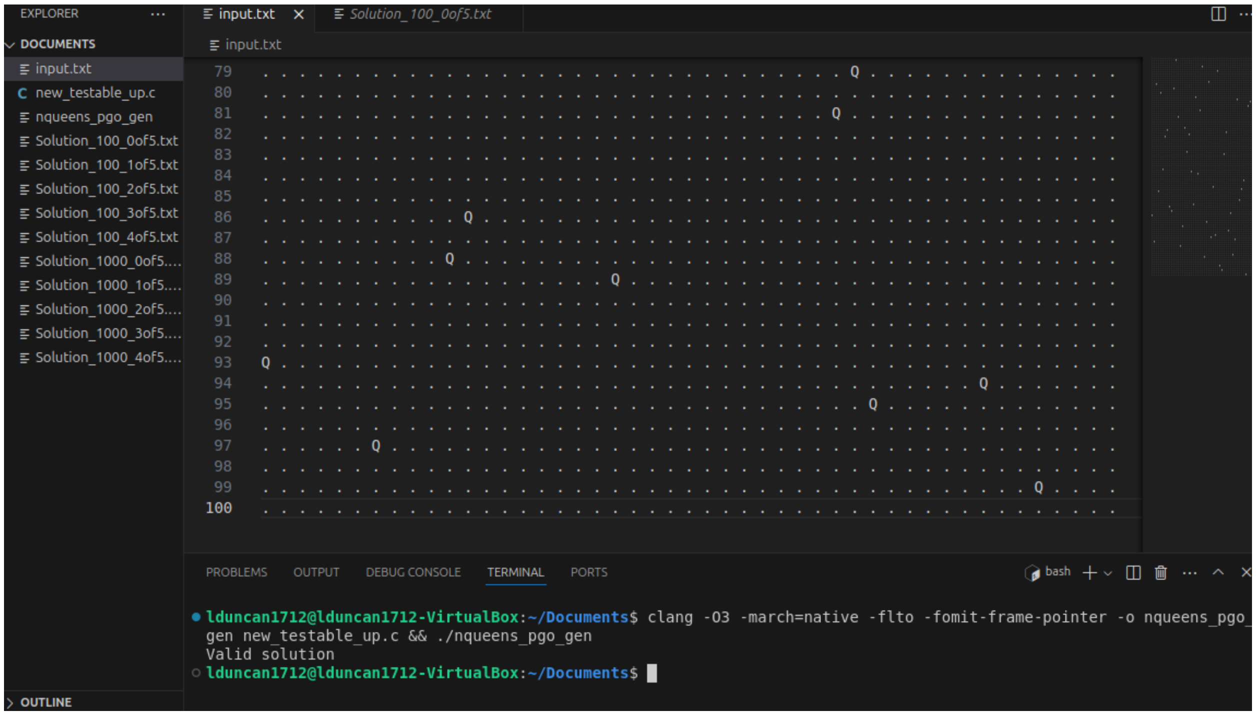
```
Starting Tests Of Size 1000000
 -- Solution found in 414.845 seconds
 -- Solution found in 130 (650) sets of steps
 -- Solution is valid!

 -- Solution found in 460.281 seconds
 -- Solution found in 142 (710) sets of steps
 -- Solution is valid!

 -- Solution found in 458.112 seconds
 -- Solution found in 568 (2840) sets of steps
 -- Solution is valid!

 -- Solution found in 424.617 seconds
 -- Solution found in 437 (2185) sets of steps
 -- Solution is valid!

 -- Solution found in 447.037 seconds
 -- Solution found in 669 (3345) sets of steps
 -- Solution is valid!



 AVERAGE FOR 5 RANDOM n=1000000 BOARD:  440.978 s
```

Below we also print the example output files created by these runs. Please note that these files are VERY LARGE for high N, so be careful when printing with n=1,000,000 (for demonstration purposes we have only printed the n = 100 to give an example, (setting printSolution=1)



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

Finally, we demonstrate testing an imported state, by setting Check Input = 1 within the file, and ensuring we have a file of the following format within input.txt to check



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

```
lduncan1712@lduncan1712-VirtualBox:~/Documents$ clang -O3 -march=native -flto -fomit-frame-pointer -o nqueens_pgo_
gen new_testable_up.c && ./nqueens_pgo_gen
Valid solution
lduncan1712@lduncan1712-VirtualBox:~/Documents$ 
```

5. Poster

Link to poster