Alex Moening
CS-203
Professor Turini
11/05/2023

# Theoretical Analysis

The N-queens problem is solved with an iterative repair algorithm using N as the board size input. The algorithms' computational efficiency is also proportionate to this input value, and thus, it will be used for the theoretical analysis.

Due to the various possible conditions, analysis of the algorithm requires a best and worst-case analysis. However, due to the random generation of the board, it is possible to generate a solution initially or to generate unsolvable boards infinitely. However, both occurrences are statistically unlikely and will be treated as a general best and worst case, while a probable best and worst case will be more practically evaluated, in which the probable cases do not generate more than one board (no resets). Additionally, all time-constant operations will be treated as a basic operation for the sake of simplicity.

## General Best Case

Summary of initialization and main collision check

1. getInput()
    a. Will loop infinitely until the user enters a valid input
        i. For valid input cases, enters try block, gets user input from the scanner, and returns it
        ii. Basic operation, return, on line 63
        iii. 1
2. initializeBoard()
    a. Create an array of size n
        i. Loop through the array and assign i to each index
        ii. Basic operation assignment on line 83
        iii. $\sum_{i=0}^{n-1} 1$
    b. Randomize the array
        i. Generate a random index, swap (n - i) index with the random index.
        ii. Basic operation swap on lines 91-94
        iii. $\sum_{i=2}^{n} 1$
    c. Generation of Diagonals
        i. Calculate both diagonal positions of each queen and place it in the diagonal array
            1. Calls calcRdiags and calcLdiags, which perform basic math operations to calculate the diagonal positions
        ii. Basic operation additions on lines 104-105

        iii. $\sum_{i=0}^{n-1} 1$

3. collisionReport()
    a. (Occurs within solveIterativeRepair(), but is a general check)
    b. Loop through both diagonal arrays at once, identify any collisions, and add them to a counter (the general best case will return false and not enter)
    c. Assuming the basic operators in the for loop are the comparisons, which are time-constant. Lines 137-141
    d. $\sum_{i=0}^{2n-2} 1$

$$C_{best}(n) = 1 + \sum_{i=0}^{n-1} 1 + \sum_{i=2}^{n} 1 + \sum_{i=0}^{n-1} 1 + \sum_{i=0}^{2n-2} 1$$

$$C_{best}(n) = 1 + (n) + (n - 1) + (n) + (2n - 1)$$

$$\boxed{\begin{gathered} C_{best}(n) = 5n - 1 \\ C_{best}(n) \in \Omega(n) \end{gathered}}$$

*Equation for general best case*

## Probable Best Case

The probable best case is when there is one collision, which is solved on the first instance of the while loop.

To reduce complexity, it can be assumed that the first time diagonal collisions are found with the if statement on line 210 is when the swap is performed. Additionally, the probable best case can be broken down into an initialization and a solver, with the initialization being taken from the general best case.

$$P_{prob-best}(n) = C_{best}(n) + P_{best-iter}(n)$$

with

$$C_{best}(n) = 5n - 1$$

For $P_{best-iter}(n)$, since the if conditions are both assumed to run once, all of the functions besides collisionReport() within the nested for loops are time-constant and will not be affected by the size of the input N. As such, they can be considered all as one basic operation (lines 210-228, except 218-219).

$$P_{best-iter}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} (1) + (1 + 2 \times collisionReport())$$

$$P_{best-iter}(n) = \sum_{i=0}^{n-2} (n - i - 1) + (1 + 2(2n - 1))$$

$$P_{best-iter}(n) = \sum_{i=0}^{n-2} (n - i - 1) + (1 + 4n - 2)$$

$$P_{best-iter}(n) = \sum_{i=0}^{n-2}(n) - \sum_{i=0}^{n-2}(i) - \sum_{i=0}^{n-2}(1) + (4n - 1)$$

$$P_{best-iter}(n) = (n - 1)n - \frac{(n-2)(n-1)}{2} - (n - 1) + (4n - 1)$$

$$P_{best-iter}(n) = 2n^2 - n - \frac{1}{2}\left(n^2 - 3n + 2\right) - n + 1 + 4n - 1$$

$$P_{best-iter}(n) = \frac{3}{2}n^2 + \frac{7}{2}n - 1$$

And so,

$$P_{prob-best}(n) = (5n - 1) + \left(\frac{3}{2}n^2 + \frac{7}{2}n - 1\right)$$

$$P_{prob-best}(n) = \frac{3}{2}n^2 + \frac{17}{2}n - 2$$

$$P_{prob-best}(n) \in \theta(n^2)$$

*Equation for probable best case*

## Probable Worst Case

The probable worst case is when there are n-1 collisions, in which one collision is reduced for each operation of the nested for loops. Because of this, the while loop will occur once for each collision, (n-1) times.

To reduce complexity, it can be assumed that the first time diagonal collisions are found with the if statement on line 210 is when the swap is performed. Additionally, the probable worst case can be broken down into an initialization and a solver, with the initialization being taken from the general best case.

$$P_{prob-worst}(n) = C_{best}(n) + P_{worst-iter}(n)$$

with

$$C_{best}(n) = 5n - 1$$

For $P_{worst-iter}(n)$, since the if conditions are both assumed to run together, they will both occur (n-1) times. As such, all of the functions besides collisionReport() within the nested for loops are time-constant and will not be affected by the size of the input N. As such, they can be considered all as one basic operation (lines 210-228, except 218-219) and will occur (n-1) times.

$$P_{worst-iter}(n) = \sum_{k=0}^{n-1}\left(\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(1) + (1 + 2 \times collisionReport())\right)$$

Since the interior of the k summation is the same as $P_{best-iter}(n)$,

$$P_{worst-iter}(n) = \sum_{k=1}^{n-1}\left(\frac{3}{2}n^2 + \frac{7}{2}n - 1\right)$$

$$P_{worst-iter}(n) = \sum_{k=1}^{n-1}\frac{3}{2}n^2 + \sum_{k=1}^{n-1}\frac{7}{2}n - \sum_{k=1}^{n-1}1$$

$$P_{worst-iter}(n) = \frac{3}{2}\sum_{k=1}^{n-1} n^2 + \frac{7}{2}\sum_{k=1}^{n-1} n - \sum_{k=1}^{n-1} 1$$

$$P_{worst-iter}(n) = \frac{3}{2}(n - 1)n^2 + \frac{7}{2}(n - 1)n - (n - 1)$$

$$P_{worst-iter}(n) = \frac{3}{2}n^3 - \frac{3}{2}n^2 + \frac{7}{2}n^2 - \frac{7}{2}n - n + 1$$

$$P_{worst-iter}(n) = \frac{3}{2}n^3 + 2n^2 - \frac{9}{2}n + 1$$

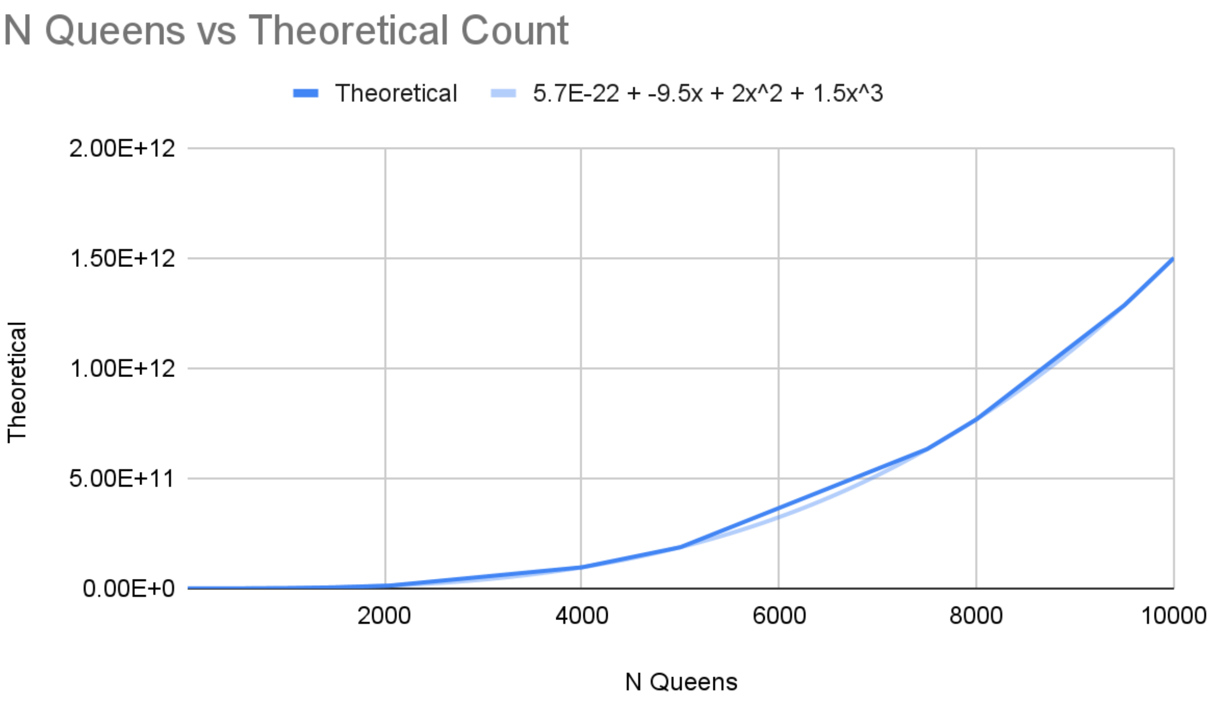
And so,

$$P_{prob-worst}(n) = 5n - 1 + \frac{3}{2}n^3 + 2n^2 - \frac{9}{2}n + 1$$

$$P_{prob-worst}(n) = \frac{3}{2}n^3 + 2n^2 - \frac{19}{2}n$$

$$P_{prob-worst}(n) \in \theta(n^3)$$

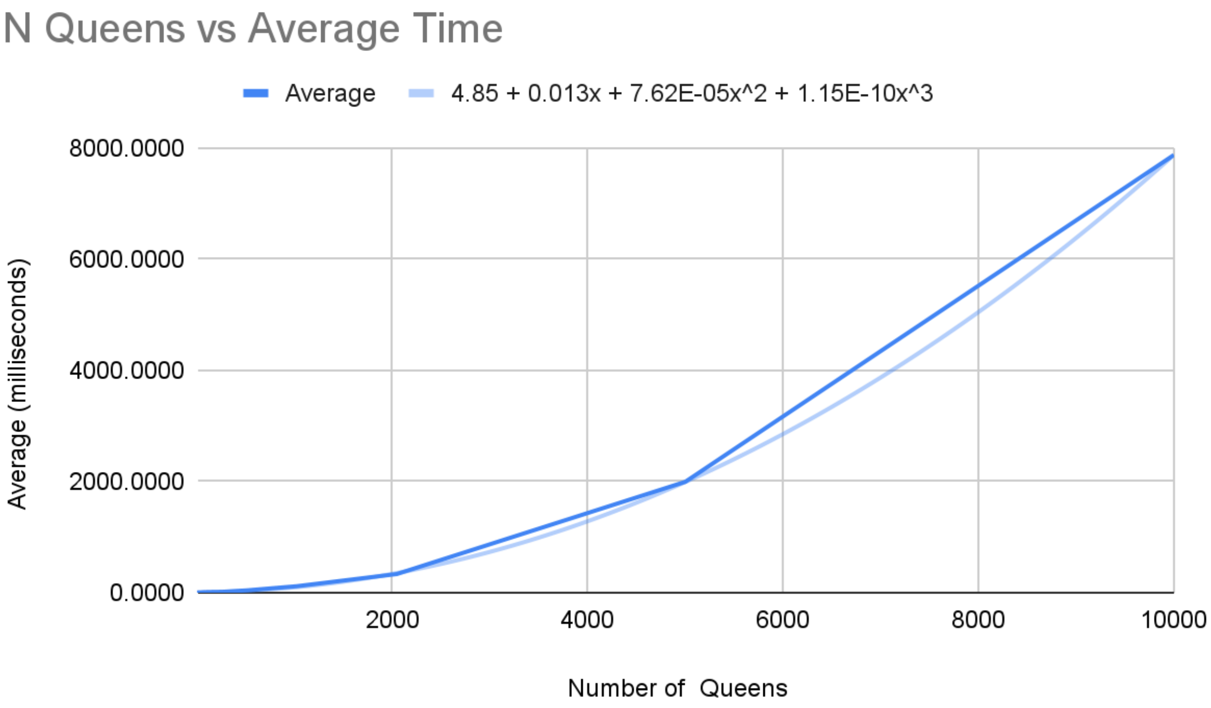
*Equation for probable best case*



## Empirical Analysis

| N Queens | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|
| 4 | 0.0596 | 0.1931 | 0.0952 | 0.0947 | 0.2922 | 0.1470 |
| 8 | 0.1347 | 0.1370 | 0.1882 | 0.1847 | 0.6509 | 0.2591 |
| 16 | 1.0932 | 0.4689 | 3.9382 | 3.9199 | 0.3378 | 1.9516 |
| 32 | 2.5616 | 3.0258 | 2.6915 | 0.5137 | 2.3720 | 2.2329 |
| 64 | 10.8420 | 2.3661 | 8.7608 | 9.2489 | 1.8812 | 6.6198 |
| 128 | 16.0420 | 12.6065 | 10.4307 | 4.8536 | 7.3715 | 10.2609 |
| 256 | 7.8808 | 22.7691 | 8.1937 | 16.0348 | 15.8362 | 14.1429 |
| 512 | 30.3354 | 45.4631 | 38.5013 | 52.7424 | 34.8817 | 40.3848 |
| 1024 | 130.5142 | 111.1659 | 107.3093 | 106.2043 | 122.5724 | 115.5532 |
| 2048 | 329.0919 | 343.0867 | 322.9010 | 322.6092 | 355.0440 | 334.5466 |
| 5000 | 1738.1149 | 1933.5642 | 1878.6384 | 2011.3471 | 2403.4133 | 1993.0156 |
| 10000 | 8873.7409 | 7044.852 | 7832.7469 | 7915.3943 | 7692.5132 | 7871.8495 |

Note: Empirical analysis used the function analyze() which iterated through an array of input sizes, printing time recorded for each. This was done utilizing the java command System.nanoTime() and scaling by 10e-6 to retrieve milliseconds



# Conclusion

Through the theoretical analysis, it was determined that $C_{best}(n) \in \Omega(n)$, $P_{prob-best}(n) \in \theta(n^2)$, $P_{prob-worst}(n) \in \theta(n^3)$, and theoretically $C_{worst}(n) \in O(\infty)$ though $C_{worst}(n)$ is not technically proper notation.

The analysis of the algorithm using a theoretical probable worst case approach results in a cubic growth factor for the probable worst case in terms of board size. When measuring runtime, the empirical analysis results in a similarly shaped cubic growth factor when comparing board size vs time. Despite this, there are inherent flaws in using runtime as a measurement for empirical analysis that make it less valuable when compared to the theoretical analysis. Since the runtime is affected by processing capabilities, it is directly affected by the computer's hardware and will vary from one to another. Similarly, if other programs are running at the same time as the algorithm, it can cause variability in performance run-to-run under otherwise exactly identical conditions.

In the probable worst case, the growth factor is $P_{prob-worst}(n) \in \theta(n^3)$. For a similar problem utilizing a brute force search, it's on the order of N! since there are decreasing choices for each iteration by one (ex. the first search has N, the second has N-1, etc. until a solution is found or the algorithm reaches 0). However, since it is an N-by-N board, that growth factor is squared. As such, the resulting growth factor of $C_{bfs}(n) \in \theta(n^2!)$. This is significantly larger than $P_{prob-worst}(n) \in \theta(n^3)$, and as such results in a much less efficient algorithm.