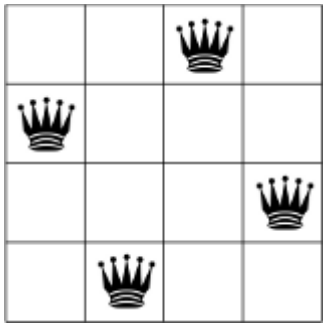
Problem Name: The N-Queens Problem

Problem Statement:

The N-queens problem[1] asks:

How can N queens be placed on an NxN chessboard so that no two of them attack each other? So, In N-Queen problem we need to find a solution (or a set of solutions) to set N-Queens in a N x N chess board so that no two Queens threaten each other.

Below, you can see a solution to the 4 queens problem.



No two queens are on the same row, column, or diagonal.

N.B: Chessboard queens can attack horizontally, vertically, and diagonally.

Algorithm[2]:

One such better way is through the use of a *backtracking algorithm*. In a backtracking algorithm, one incrementally builds candidates to the solution(s) and abandons partial candidates (backtracks) as soon as it is determined it cannot possibly be a valid solution.

For the N-Queens problem, one way we can do this is given by the following:

1. *For each row, place a queen in the first valid position (column), and then move to the next row*
2. *If there is no valid position, then one backtracks to the previous row and try the next position*
3. *If one can successfully place a queen in the last row, then a solution is found. Now backtrack to find the next solution.*

Pseudo Code:

Here is the Pseudo Code[2] for N-Queens problem:

```
Create empty stack and set current position to 0

Repeat {

  loop from current position to the last position until valid position found //current row

  if there is a valid position {
    push the position to stack, set current position to 0 // move to next row
  }

  if there is no valid position {
    if stack is empty, break // stop search
    else pop stack, set current position to next position // backtrack to previous row
  }

  if stack has size N { // a solution is found
    pop stack, set current position to next position // backtrack to find next solution
  }
}
```

Implementation:

Here is the source[3] code in prolog:

```
queens(N, Rs) :-
   bagof(X, between(1,N,X), Qs),
   path(Qs,[],Rs).

path(Qs,Xs,Rs) :-
        Qs = [] -> Rs = Xs ;
        select(Q,Qs,Qe),
        %write(Q),
        %write(Xs),nl,nl,
        safe_queens(Xs, Q, 1),
        %write(Q),
        %write(Xs),nl,
        path(Qe,[Q|Xs],Rs).

safe_queens([], _, _).
safe_queens([Q|Qs], Q0, D0) :-
   abs(Q0 - Q) =\= D0,
   D1 = D0 + 1,
   %write(D0),nl,
   %write(Q0),nl,
   safe_queens(Qs, Q0, D1).
```

To run this program (Sample):

```
?- queens(4,Ps).
Ps = [3, 1, 4, 2] ;
Ps = [2, 4, 1, 3] .

?-
```

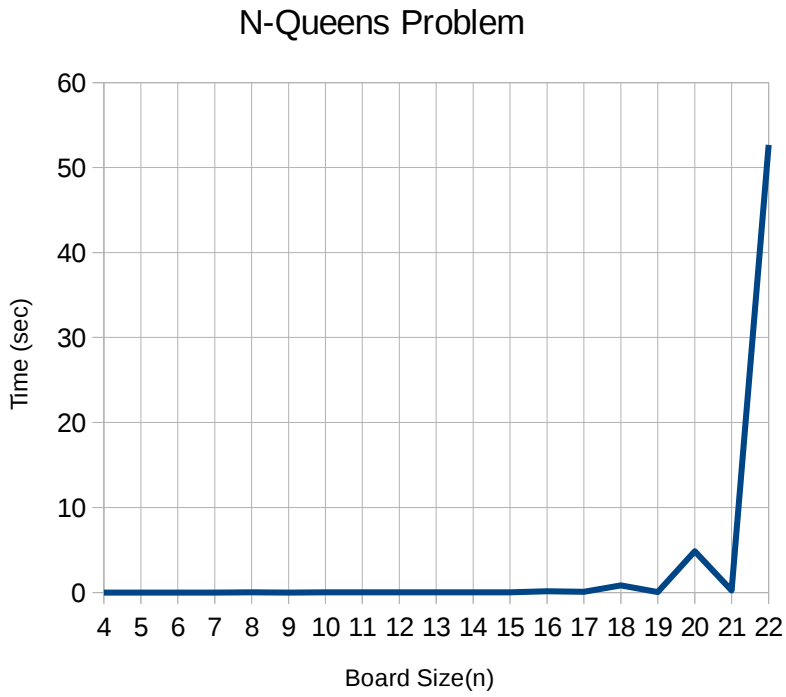
Computer Specification:

OS : Ubuntu 16.04 LTS
Memory : 5.7 GiB
Processor : Intel® Core™ i3-2370M CPU @ 2.40GHz × 4
Graphics : Intel® Sandybridge Mobile
OS Type : 64-bit

Discussion:

Here, We calculate the complexity for one solution. The table and chart for this problem for one solution.

Board Size(n)	Time(sec)
4	0.000
5	0.000
6	0.000
7	0.000
8	0.001
9	0.000
10	0.001
11	0.001
12	0.003
13	0.001
14	0.025
15	0.020
16	0.156
17	0.094
18	0.819
19	0.059
20	4.851
21	0.257
22	52.681



The algorithm complexity is O(n!) for finding all solutions. But if I calculate complexity for one solution then it decreases rapidly.

Here some information about complexity and number of solution which is taken from Google [1].

The number of solutions goes up roughly exponentially with the size of the board:

Board size	Solutions	Time to find all solutions (ms)
1	1	0
2	0	0
3	0	0
4	2	0
5	10	0
6	4	0
7	40	3
8	92	9
9	352	35
10	724	95
11	2680	378
12	14200	2198
13	73712	11628
14	365596	62427
15	2279184	410701

Many solutions are just rotations of others, and a technique called *symmetry breaking* can be used to reduce the amount of computation needed. We don't use that here; our solution above isn't intended to be fast, just simple. Of course, we could make it much faster if we wanted to only find one solution instead of all of them: no more than a few milliseconds for board sizes up to 50.

Limitation: Here, this code is not efficient for board sizes more than 50. If this code is run for board sizes more than 50, it can take few hours to find a solution or PC will be crashed.

Reference:

[1] https://developers.google.com/optimization/puzzles/queens
[2] http://www.oxfordmathcenter.com/drupal7/node/627
[3] https://rosettacode.org/wiki/N-queens_problem#Prolog