| | | | |
|---|---|---|---|
| | Q | | |
| | | | Q |
| Q | | | |
| | | Q | |

n = 4

| | | | |
|---|---|---|---|
| | 1 | | |
| | | | 1 |
| 1 | | | |
| | | 1 | |

n = 4

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |

BINARY ??!??!!!!!!

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |

binary

| |
|---|
| 4 |
| 1 |
| 8 |
| 2 |

decimal

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | **4** | $\mathbf{2}^2$ |
| 0 | 0 | 0 | 1 | **1** | $\mathbf{2}^0$ |
| 1 | 0 | 0 | 0 | **8** | $\mathbf{2}^3$ |
| 0 | 0 | 1 | 0 | **2** | $\mathbf{2}^1$ |

binary decimal base 2

Working solution

forget about the 1s and 0s

array indices

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |

rowPossibilities  = [4, 1, 8, 2];

| array index | decimal | base 2 |
|---|---|---|
| 0 | 4 | $2^2$ |
| 1 | 1 | $2^0$ |
| 2 | 8 | $2^3$ |
| 3 | 2 | $2^1$ |

Working solution

Failing solutions?

## Working solution

| binary | | | | decimal | base 2 |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 4 | $2^2$ |
| 0 | 0 | 0 | 1 | 1 | $2^0$ |
| 1 | 0 | 0 | 0 | 8 | $2^3$ |
| 0 | 0 | 1 | 0 | 2 | $2^1$ |

red = primary concept

yellow = secondary

*failing solution:* row conflict

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 12 | $2^{?}$ |
| 0 | 0 | 0 | 1 | 1 | $2^{0}$ |
| 1 | 0 | 0 | 0 | 8 | $2^{3}$ |
| 0 | 0 | 1 | 0 | 2 | $2^{1}$ |

binary decimal base 2

*failing solution:* row conflict

| | | | |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |

binary

| |
|---|
| 12 |
| 1 |
| 0 |
| 2 |

decimal

| |
|---|
| $2^?$ |
| $2^0$ |
| $2^?$ |
| $2^1$ |

base 2

*failing solution:* <u>column conflict</u>

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |

binary

| |
|---|
| 8 |
| 1 |
| 8 |
| 2 |

decimal

| |
|---|
| $2^3$ |
| $2^0$ |
| $2^3$ |
| $2^1$ |

base 2

*failing solution:* <u>diagonal conflict</u> v1.1

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 8 | $2^3$ |
| 0 | 0 | 0 | 1 | 1 | $2^0$ |
| 0 | 1 | 0 | 0 | 4 | $2^2$ |
| 0 | 0 | 1 | 0 | 2 | $2^1$ |
| binary | | | | decimal | base 2 |

## *failing solution:* diagonal conflict v1.1

$abs( 2 - 3 ) === abs( 2 - 1 )$

$abs( -1 ) === abs( 1 )$

$1 === 1$

| array index | decimal | base 2 |
|---|---|---|
| 0 | 8 | $2^3$ |
| 1 | 1 | $2^0$ |
| 2 | 4 | $2^2$ |
| 3 | 2 | $2^1$ |

*failing solution:* **diagonal conflict** v1.2

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |

binary

| |
|---|
| 8 |
| 4 |
| 1 |
| 2 |

decimal

| |
|---|
| $2^3$ |
| $2^2$ |
| $2^0$ |
| $2^1$ |

base 2

*failing solution:* **diagonal conflict** v2.1

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |

binary

| |
|---|
| 4 |
| 8 |
| 1 |
| 2 |

decimal

| |
|---|
| $2^2$ |
| $2^3$ |
| $2^0$ |
| $2^1$ |

base 2

## *failing solution:* diagonal conflict v1.1

$\text{abs}(1 - 3) === \text{abs}(3 - 1)$

$\text{abs}(-2) === \text{abs}(2)$

$2 === 2$

| array index | decimal | base 2 |
| --- | --- | --- |
| 0 | 4 | $2^2$ |
| 1 | 8 | $2^3$ |
| 2 | 1 | $2^2$ |
| 3 | 2 | $2^1$ |

*failing solution:* **diagonal conflict** v2.2

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | $2^0$ |
| 0 | 1 | 0 | 0 | 4 | $2^2$ |
| 0 | 0 | 1 | 0 | 2 | $2^1$ |
| 1 | 0 | 0 | 0 | 8 | $2^3$ |
| binary | | | | decimal | base 2 |

*failing solution:* diagonal conflict v1.1

$$abs(\ 0 - 3\ ) \ === \ abs(\ 0 - 3\ )$$

$$abs(\ -3\ ) \ === \ abs(\ 3\ )$$

$$3 === 3$$

| array index | decimal | base 2 |
|---|---|---|
| 0 | 1 | $2^0$ |
| 1 | 4 | $2^2$ |
| 2 | 2 | $2^1$ |
| 3 | 8 | $2^3$ |

conflicts recap:

## row numbers must be base 2, else ***row conflict***

can’t be 12 because in binary 12 is 1100 and contains more than one 1

## no identical numbers, else ***column conflict***

can’t have 8 and 8 because 1000 and 1000 stack 1s

## if the absolute value of the difference in array indices = the absolute value of the difference in base 2 exponents, there exists a ***diagonal conflict***

if n = 2 and board = [4, 2] = [$2^{2}$, $2^{1}$], abs(base 2 diff) === abs(index diff) → abs( 2 – 1 ) === abs( 0 – 1 ) “coding is not fast;

it’s 10% typing and 90% thinking.

└→ getting a solution is a process, not a **binary** light switch.”

-Marcus Phillips

# Bitwise Operators

let’s jump into the chrome console…

**input**: n, a number, representing an *n* by *n* board with *n* non-conflicting queens

0. declare a solution counter variable equal to 0.
1. get an array of potential rows for an *n* x *n* board.

    n = 5 → [1, 2, 4, 8, 16]

2. declare recursive function expression that searches and potentially appends new rows.
    - 2.0 base case – if board length = n, add to solutionCounter
    - 2.1 initialization - iterate over the list and begin a board search using an array containing each.
        - search([1]), search([2]), search([4]) . . .
    - 2.2 recursion -using our ***row***, ***column***, and ***diagonal*** conditions, check to see if we can append from our potential rows array. If all tests pass:
        - 2.2.0 append potential row to board and call recursive search on new board

3. invoke recursive search function on empty array
4. return solutionCounter

**output**: number of solutions for *n* queens

*let's not worry about time/space complexity on this one ;)*

just an outline!

```
var bitwiseNQueens = function(n) {

  var rowOptions = [1];
  for(var i = 1; i < n; i++) {
    rowOptions.push(rowOptions[i-1] << 1);
  }

  var solutionsCounter = 0;

  var findSolutions = function(board) {
  };

  findSolutions([]);

  return 'A board with ' + n + ' queens has ' + solutionsCounter + ' solutions.
  '

};
```

0) creating row of base 2 options

1) declaring counter variable

2) recursive function expression

3) recursive function invocation

4) return # of solutions

*let's jump into findSolutions() . . .*

```
var findSolutions = function(board) {
  // base case
  if(board.length === n) {
    return solutionsCounter++;
  }

  // initializer
  if(board.length === 0) {
    _.each(rowOptions, function(option){
      findSolutions([option])
    });
  }

  // recursion
  if(board.length > 0){
    _.each(rowOptions, function(potentialRow){

      var failures = 0;

      _.each(board.reverse(), function(rowToCompare, i){
        if(potentialRow === rowToCompare ||
           potentialRow === (rowToCompare >> (board.length - i)) ||
           potentialRow === (rowToCompare << (board.length - i))) {
            failures++;
        }
      });

      if(!failures){
        var confirmedRow = potentialRow;
        var newBoard = board.slice();
        newBoard.push(confirmedRow);
        findSolutions(newBoard);
      }
    });
  }
};
```

2.0) base case
updating solutionCounter if full board n x n board exists

2.1) dealing with initial '[]' argument `findSolutions([]);`
recurses on singly populated array

2.2) iterating over potential next row options

2.3) keeping track of failures

2.4) checking: iterating over current board, backwards

2.4.0) checking column conflicts

2.4.1) checking both diagonal conflicts while accounting for absolute changes in:

1) base 2 exponent
2) indices

2.5) if the potential row passed all the tests
set potentialRow to confirmedRow for readability
slice our current board to avoid modification conflict
add our newly tested row to the board
test it!

underscore!