Sudoku Algorithm (Back-tracking)

Giovanni Matthew DiSalvo

Quick brief on the game of Sudoku and rules

Sudoku is a puzzle game where you are given a board, in classic this is a 9 by 9 board, in which some of the board is already filled for you prior to you beginning (the amount of filled cells is varied upon the difficulty of each puzzle). **You then can insert numbers ranging from 1-9 into any of the empty cells on the board**. The board is split up into 9 subgrids each being of size 3 by 3. This is important because **when placing your numbers, your number must be unique within the subgrid, horizontally, and vertically.**

9	4	2	1	6	3	8	5	7
5	3	6	2	8	7	9	4	1
8	7	1	9	5	4	2	3	6
3	2	7	8	1	9	4	6	5
1	5	4	3	2	6	7	9	8
6	9	8	7	4	5	1	2	3
2	6	5	4	7	1	3	8	9
7	8	9	6	3	2	5	1	4
4	1	3	5	9	8	6	7	2

History on the game and algorithm

The game of Sudoku is a relatively new game in which the modernized version of the game, **first being introduced in 1979**, was designed anonymously by Howard Garns. Garns was a 74 year old retired architect and puzzle constructor from Connersville, Indiana. The puzzle was first introduced to Japan in which it received its name “Sudoku” (5). As for the backtracking Sudoku solving algorithm, it was the first of many algorithms to solve Sudoku which first came to light in 2004, **developed by Wayne Gould in 2004**. Wayne Gould is a retired judge from New Zealand who had acquired great interest in the game of Sudoku whilst on a visit to Japan (6). Gould had **spent six years developing this Sudoku solving algorithm** in which he had **first started with a basic brute-force approach** and then continually added countermeasures that would accomplish more difficult Sudoku puzzles. Through trials and tribulations, he eventually came up with this final technique and tweaked it accordingly to boost its performance.

This method vs others

There are many other methods that solve Sudoku puzzles including those that use stochastic search, optimization methods, constraint programming, exact cover, relations and residuals. Although backtracking not only **guarantees a solution every time**, but the solving time is also unrelated to the difficulty of each individual puzzle, **meaning the runtime of performing the hardest puzzle would run similarly to the same time of the easiest of puzzles**. The downsides to the backtracking algorithm to solve Sudoku puzzles would be that the run time is **often slow compared to those that use deductive methods.** Although, the runtime of this algorithm can solve most puzzles in under a second on modern computers, which is much better than the brute force algorithm which took one programmer in **2008, six hours to get a final solution** for a Sudoku puzzle.

High level intuition

Step 1: We start from the cell in the first row and first column, **first going from left to right through each row**, looking for the first empty cell.

Step 2: Once we find an empty cell, we try to place a number 1-9 in the cell.

Step 3: **If the number is valid then we continue to repeat steps 1-3**, if it is not valid we go onto step 4.

Step 4: **If the number violates the rules of Sudoku** of which we covered earlier, such as the number must not already be in the subgrid, the number must also be unique within their respective horizontal and vertical lines, then we must backtrack to a previous cell of which we have entered before and try other valid numbers. **If there are no other valid numbers for said backtracked cell, it will continue this process of backtracking to previous cells until we find a cell we can swap to another valid number.** After this, it will then proceed back forwards.

Step 5: The algorithm is complete when valid numbers fill all the empty cells. We have completed the Sudoku puzzle.

Pseudocode

Parameter board is a two dimensional array representing the 9 by 9 board with the puzzle numbers already entered, and empty spots being signified by the number 0.

MAIN RECURSIVE FUNCTION

```
Function sudokuSolver(board)
        If (isBoardFilled(board) == true)
                Return board

        Xcoord, Ycoord = nextEmptyCell(board)
        For int j, 1 to 9
                If (isValid(board, Xcoord, Ycoord, j) == true)
                        board[Xcoord][Ycoord] = j
                        If (sudokuSolver(board))
                                Return board
                        grid[Xcoord][Ycoord] = 0

        Return false
```

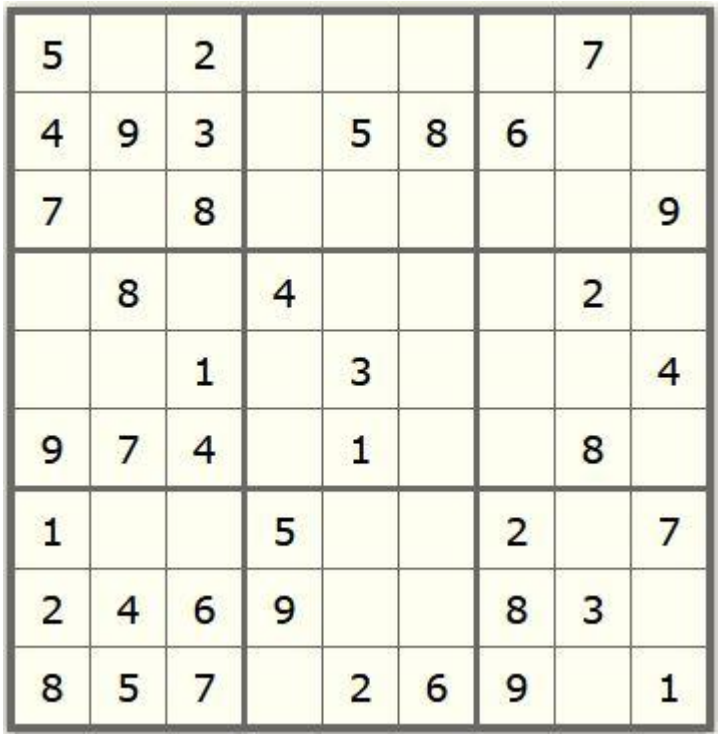
Runtime

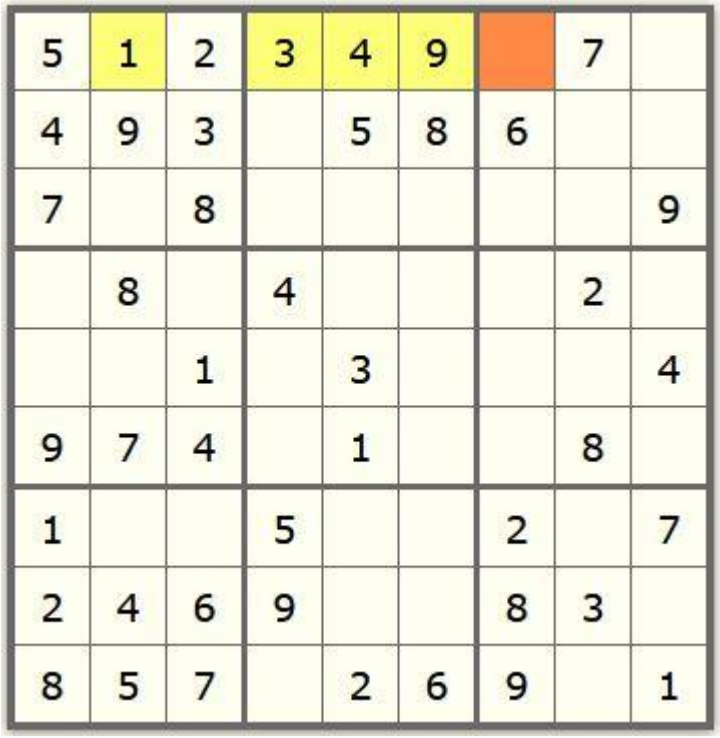
The runtime of the algorithm is very much dependent on the amount of possibilities that can be within each cell, this varies from Sudoku puzzles, but it is commonly 9 as most play Sudoku classic. Knowing this, here is the most important segment from our main recursive function contains the following:

```
For int j, 1 to 9
        If (isValid(board, Xcoord, Ycoord, j) == true)
                board[Xcoord][Ycoord] = j
                If (sudokuSolver(board))
```

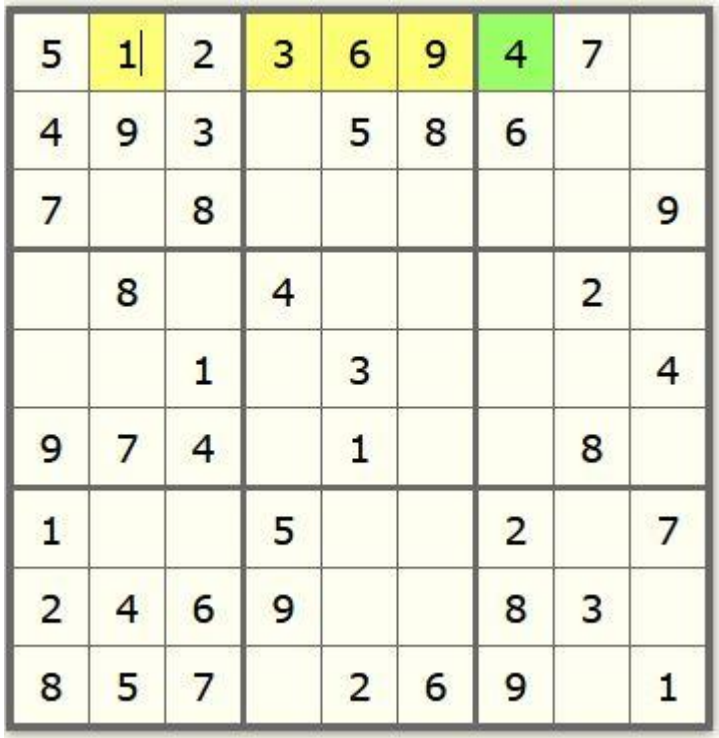
We are going through 9 possibilities for each cell and we know that this function is recursive which introduces the point that our runtime is going to be exponential. Writing this into a recurrence equation can then be written as $T(m) = 9*T(m - 1) + O(1)$. Solving this results in $O(9^m)$ From this we can make the case that our runtime is $O(9^m)$ for both our worst and average case. Although the best case would be ??? (Audience Help)

Visualization

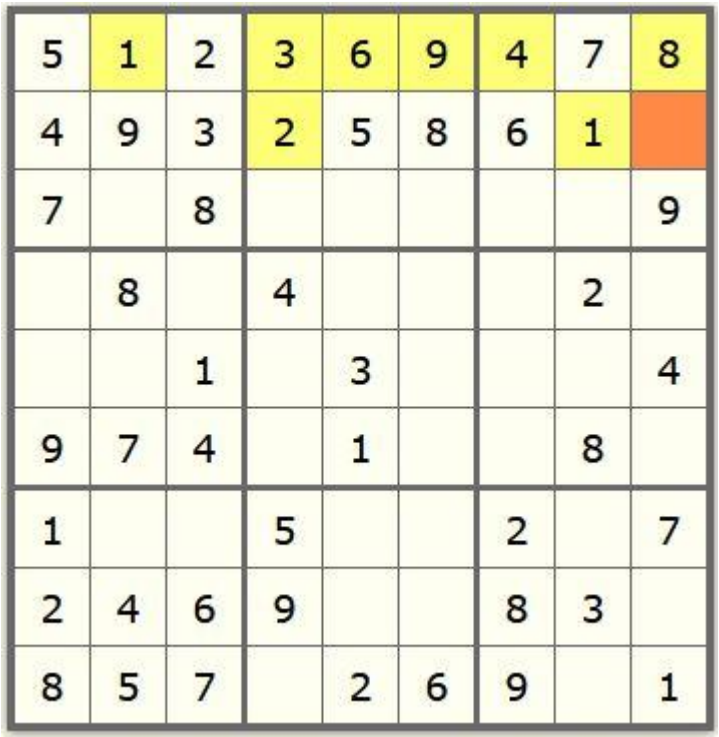




Visualization



5	1	2	3	6	9	4	7	
4	9	3		5	8	6		
7		8						9
	8		4				2	
		1		3				4
9	7	4		1			8	
1			5			2		7
2	4	6	9			8	3	
8	5	7		2	6	9		1



5	1	2	3	6	9	4	7	8
4	9	3	2	5	8	6	1	
7		8						9
	8		4				2	
		1		3				4
9	7	4		1			8	
1			5			2		7
2	4	6	9			8	3	
8	5	7		2	6	9		1

Visualisation Complete

5	1	2	3	6	9	4	7	8
4	9	3	7	5	8	6	1	2
7		8						9
	8		4				2	
		1		3				4
9	7	4		1			8	
1			5			2		7
2	4	6	9			8	3	
8	5	7		2	6	9		1

5	6	2	1	4	9	3	7	8
4	9	3	7	5	8	6	1	2
7	1	8	2	6	3	4	5	9
3	8	5	4	9	7	1	2	6
6	2	1	8	3	5	7	9	4
9	7	4	6	1	2	5	8	3
1	3	9	5	8	4	2	6	7
2	4	6	9	7	1	8	3	5
8	5	7	3	2	6	9	4	1

Planned

Implementation of 9 x 9, 16 x 16, maybe 25 x 25?

Actual real time comparison of easy vs medium vs hard puzzles

Run time comparison of 9 x 9 vs 16 x 16