The graph above compares different scenarios of both the best and worst case. As you can see the best case(Already solved grid) runs much faster than the worst case( 64 empty cells in a grid).

The graph above compared the number of empty cells with the number of backtracks needed to solve a Sudoku puzzle. It is clear in the graph that as the number of empty cells increases the the number of backtracks needed increases exponentially.

The graph above compared the number of empty cells with the time taken(s) to solve a Sudoku puzzle. It is clear in the graph that as the number of empty cells increases , the time taken to solve the puzzle increases exponentially. This concludes that the theoretical analysis corresponds with the empirical analysis.

Theoretical analysis

Examining our specific backtracking algorithm:

The scenarios for the worst is a Sudoku puzzle to be uniquely solved contains 64 empty cells(17 clues) .

Complexity for worst case:

**Line Z:** This if statement checks if there any empty cells available, this O(1)

**Line A** : This for loop runs N times, where N is the number of rows/columns.

**Line B**: The isSafe function with in the for loop in **Line A** calls 3 other functions all with the complexity of N , thus adding up to N+N+N = 3N. Including the outter if statement: N+N+N+ 1= 3N+1

**Line C:** This If statement returns true if the Sudoku puzzle is solved , Thus a complexity of O(1)

**Line D:** This returns false, if the Sudoku puzzle is not completely solved , thus O(1).

Since this is a recursive algorithm , when the code reaches **Line D** ,is where the recursion occurs. The worst is for the algorithm to recur O(N^2) times.

Therefore the complexity for the worst case is:

Complexity =

Complexity for best case:

The scenario for the best case is an already solves Sudoku puzzle i.e. Number of empty cells are zero.

Best Case

**Line Z:** This if statement checks if there any empty cells available, since the best case has no empty cells available , it would go to  **Line X** and return true, thus the complexity for the best case is O(1)

Problem statement

A Sudoku puzzle has many cases with regards to its positions of numbers as well as the number of clues provided. This experiment will be testing the performance (Time taken to solve a puzzle and the number of Backtracks needed) of the Recursive Backtracking Algorithm by solving different cases of 9x9 Sudoku puzzles.

Design

This experiment will consist of the following:

* A Recursive BackTracking Algorithm implemented in Java 7(Eclipse).
* A TextFile names sudoku.txt , which will contain our test samples of partially completed Sudoku puzzles. A ‘0’ in any grid marks an empty cell

Hypothesis

* The performance of the BackTracking Algorithm will decrease as the number of empty cells increases.
* The number of Backtracking needed will increase as the number of empty cells increases.

Conclusion

From our analysis it is clear that our Hypotheses are correct. However, there were many isolated incidences that did not produce expected results. Upon further research it is concluded that the complexity of a Sudoku puzzle is not always determined strictly by the number of clues/empty cells provided.