Sudoku Solver final report

Group members:

Shaheer Kamal

Badar Azeemi

Ammar Khawaja

Goal:

The main focus of our project is to implement python techniques to make an algorithm that solves the famous Sudoku board game. Sudoku is a board game comprising of a 9 by 9 grid (usually). Some boxes are filled with numbers from 1 to 9 while others are empty. The aim is to fill the board with combination of numbers such that no number appears more than once in a row, a column or a small 3 by 3 mini grid within the main grid.

Approach:

Two different algorithms have been implemented to analyze the best possible approach to solve the puzzle:

1. Naïve approach / Brute force approach:

It is the most basic approach to solve the puzzle. It works basically by trying all possible combinations in the boxes of the grid to end up with a combination that is the unique solution of the puzzle. After every run the algorithm checks whether the board is safe (valid solution) or not. If it is then the output solution is returned otherwise a different combination is tried. This method to solve the puzzle is not efficient and sometimes with more complicated boards fails to solve the puzzle effectively. The complexity of this approach is big O (n^2) since there are n (9 for a common board) possibilities in a particular cell.

1. Backtracking:

Instead of the naïve approach backtracking is a more efficient and commonly implemented algorithm. The approach uses recursion. In this particular approach if a particular number doesn’t fit in a cell so according to the algorithm we move backwards recursively and remove the last guessed number and replace it with another number instead that fits in that box . For a traditional Sudoku there are 9 different possibilities at particular index so the time complexity of this approach is big 0(n^2). This may seem high but in reality this is just an upper bound for the runtime and the code in many cases executes more quickly to solve the puzzle than the naïve approach.

Challenges faced:

Thinking of the recursive approach to solve the above problem was challenging and took some time to be executed efficiently.

DSA techniques used:

The nested list data structure has been used to simulate the Sudoku board. The code relies heavily on recursion to execute properly. The time library of python is used to record the runtime of the different approaches and matplotlib is used to graph them.