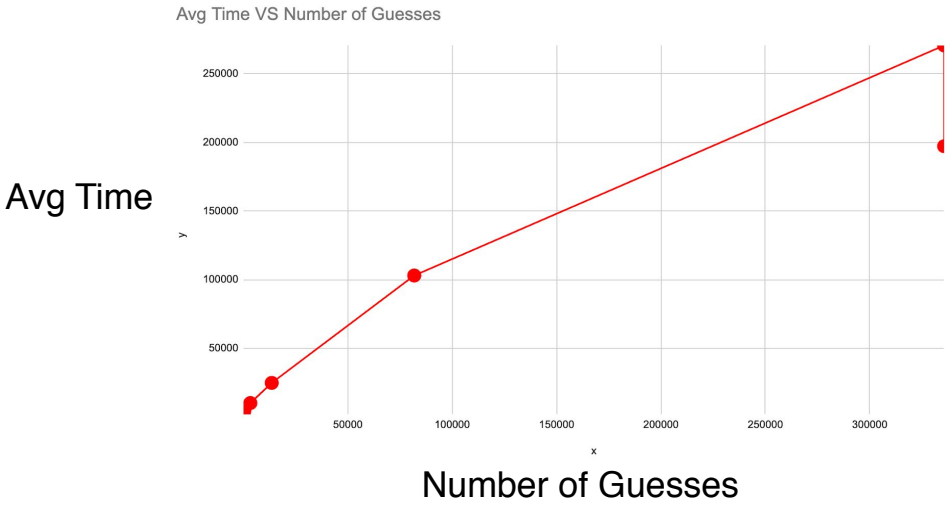
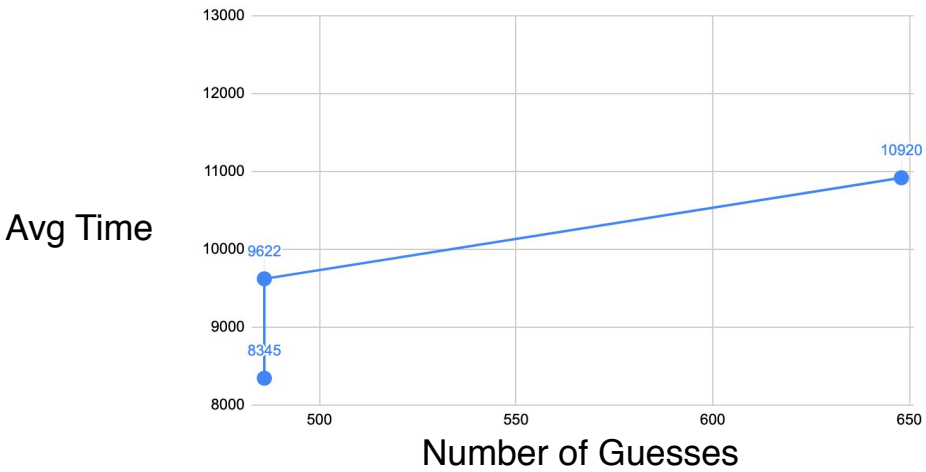
Alejandro Rubio
Alejandro Serrano

Analysis Document

The following graph represents the average time the recursive method took to solve the puzzle (y-axis) and the number of guesses it did to solve it (x-axis).
Graph:



Overall, the recursive algorithm seems to be working properly when solving, but the time increases by a lot depending on the difficulty of the puzzle it is trying to solve.



For the SudokuByElimination class, something similar is taking place. The only main difference is that instead of using an amount of guesses, our strategy was to go for the amount of eliminations the solve method does. In this case, the amount of eliminations is in the x - axis and the time it took for those eliminations is in the y-axis. Unlike the recursive method, there isn't really any backtracking during the elimination process, so for us the best way to measure the effectiveness of solve was to calculate eliminate.

For the design document we said that it would take us around 10 hours to finish the assignment. We thought that we might need 5 hours for one class and 5 for another. But it took us around 20 hours or more.

The part that was the hardest for us was definitely the recursive solve method. Mainly because we didn't have a good grasp on how the method worked conceptually, so we tried to solve it at times by guessing, which, of course, cost us a lot of time. Maybe a better plan will be to have conceptualized what is required to complete the method (on paper) before applying it to code. Not having this concept properly understood made the recursive solve method take a disproportionate amount of time in comparison with the other methods.

Definitely, the recursive method is not magic, but the entire concept it's still difficult to grasp. Nonetheless, by developing a more challenging method that used this we were able to understand it more.

The way the recursive works is as follows:

- The for loops help us traverse the 2d array indices.
- Then we check if the cell is available.
- We check if is valid for row, column or box. If it is, then we set there and we call the method again. This is where we use recursion.
- Otherwise, we need to return false so that it goes to the previous call. This will allow the call to check the values again and guess which one will fit better in the appropriate cell.

The way SudokuByElimination works is by using the 'eliminate' method, which checks rows, columns and the box and then eliminates the value from the set if is not valid.

The toString methods have a very similar structure. They both use two loops. The recursive class uses two for loops, and the elimination class uses one for loop and a while.hasNext that helps the iterator advance. The other aspects of the toString are doing basically the same thing, which is constructing the string that needs to be returned. Probably making the method a little bit more general in a way that allows different types of values to enter would help make it more accessible for other classes.