Sam Hopkins
CS 470 – Sudoku

Part 1:

Minimum-remaining-value not used: Takes too long to find a solution
Degree heuristic not used: 3.59 seconds on average
Least-constraint not used: 1.13 seconds on average
Without forward checking: Takes too long to find a solution
With all heuristics: 1.09 seconds on average
Without any heuristics: takes too long to find a solution

The heuristics that solve my sudoku the fastest are MRV, the degree heuristic, least-constraint heuristic, and forward checking.

Part 2:

I learned a lot about how these different heuristics attempt to optimize a backtracking algorithm. I thought it was interesting how a lot of the heuristics I understand are incredibly important, while others are simple tweaks that can make the program run even faster.

An example of this is shown in the difference of time between minimum remaining value and that of least-constraint. We see that if we choose not to implement the minimum remaining value, then our program can run on and on for hours. That being said, if we just choose to not use the least-constraint heuristic, we see that we only sacrifice a few milliseconds. While this would surely lead to a lot more of a difference on bigger puzzles, in the case of the sudoku puzzles I ran, it was very minimal. And that makes sense if we analyze deeper what the least-constraint heuristic is doing. It simply is reordering the different values that we have based on how constrained they are, so we do the ones with the least-constraints first. This extra ordering, while it may make our code faster, isn't necessary, at least on the puzzles that my program ran. That extra ordering only saves us a few milliseconds because our backtracking algorithm will find issues fast anyway.

Sam Hopkins
CS 470 – Sudoku

I also learned that a good backtracking optimization (and just a good optimization algorithm in general) really comes down to a bunch of little pieces that manipulate the data so that we only check things that we need to. It boils down to the return fast methodology, where if we find an issue, we want to find it as soon as we can so that we don't waste time going down a path that will just fail. This is what optimization really is, is mixing and matching the data so that we only follow paths of most promise first, and don't waste time on other paths that exist.