COMP560 Fall 2019
Assignment 1: KenKen

Group Members: Tony Bird, Ellie Hadden-Ford, Thomas Goss

Simple Backtracking Search:
We iterate through the puzzle in row-major order. For each cell, we try values 1...n, checking if row, column, or box constraints are violated. If all constraints are satisfied, we record the cell value and continue to the next cell. Otherwise, we go back and increment previous cell(s) until we find valid values.

Best Backtracking Search:
The best backtracking search begins by sorting the cells from most constrained to least constrained. This is done by determining the number of valid values for a cell. For multiplication and addition, we recursively check all combinations of values, e.g, for 3 cells, we check the sum or product of [1,1,1] through [6,6,6]. For division and subtraction, since there are only 2 cells, we simply calculate distinct pairs that reach the target. After ordering our cells, we run the same backtracking algorithm as above, beginning with the cells with the fewest valid values.

Local Search:
This local search algorithm uses simulated annealing with random restarts. First, you evaluate the initial randomly generated state with the utility function, which is the number of cells in the puzzle that have invalid values, or the total number of cell constraints violated. Next, you randomly choose a cell, and change its value to generate a neighboring state. If this randomly generated value does produce a new state (the randomly generated value is not the current value of the cell) and the state has not been seen before, proceed. If not, try again. Evaluate the neighboring state, and accept if it has a lower energy (lower output of the utility function- a 0 indicates the solution has been found). If it has higher or equal energy, you accept based on a probability (1-energy of state/temperature). Once you have either accepted or rejected the state, the process of generating a new state begins again. Once you have not improved your state 40 times, you perform a random restart. This generates an entirely new random state, and the process of generating neighbor states begins again. 1000 random restarts are permitted before the algorithm stops if the solution has not been found.

In terms of node ordering- I tried randomly selecting a set of nodes, finding the value of the cell that would result in the smallest energy, and selecting my new state from one of these options, but it did not improve my result in terms of number of constraints violated, and made the algorithm run much slower. I implemented this selection of states with both simulated annealing and regular hill climbing, and got similar (worse) results.

Individual Contribution:
Tony- initial setup of Java classes and reading input, backtracking search in Java
Thomas- reimplementation in Python (reading input, setup), best backtracking search
Ellie- best local search