I used an array of characters to represent my individual, in which the index at which a character is located, corresponds to its value.

I created a random instance by iterating through the list of the characters involved, and repeatedly selecting a random index until I found an empty index, and placing the next character there.

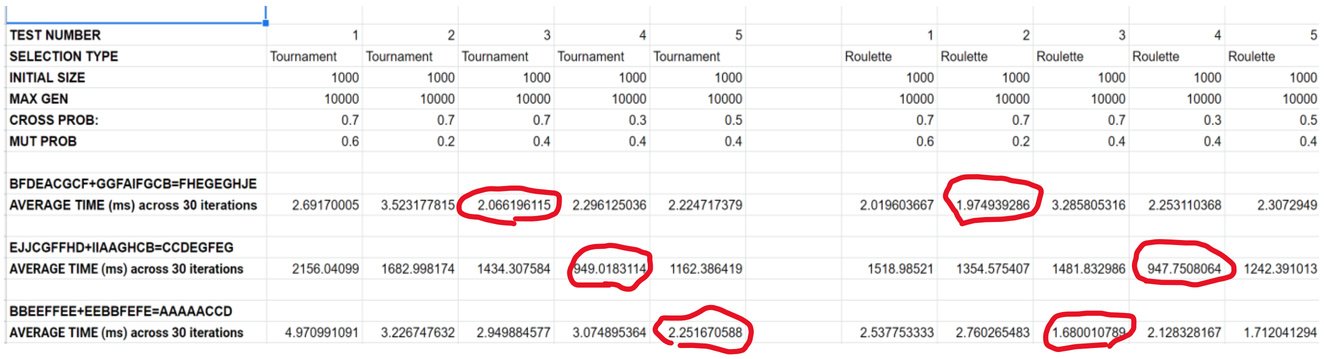
My fitness function retuned the absolute value of (sum-augend-addend) when the values for each character of that individual were plugged back into the original function. Thus, my threshold was reached when the fitness was zero, and thus, that individual provided a mapping that was a solution to the equation.

My algorithm performed mutation by selecting two pairs of two indices (ensuring that at least one index in each pair contained a character). For each pair, it swapped the values located at the two indices in that pair.

My algorithm performed crossover of individuals a and b by taking a list of the relevant characters and splitting it in half. For the first half of the characters, it placed them in newA at the same indices that they were at in a, and in newB at the same indices that they were at in b. Then, for the second half of the characters, it found the index where they were located in b, and attempted to place them there in newA, moving to the right until it found an open index. It did the same in newB for the indices at which those characters were located in a.

I ran three sets of experiments, the results of which can be found in the chart below, each with different values for augend, addend, and sum. Within each experiment, I ran five tests with tournament selection, and five tests with roulette, keeping the values of maxGen and initial size constant, and toggling the values of crossover and mutation probability for each of the five tests. In each of the tests, I ran it 30 times and calculated the average time in milliseconds that this test took, ignoring anything drastically different from the mean when calculating my average so it wouldn't be skewed by random outliers.

The results, depicted below are entirely inconclusive. The only thing I can say with certainty, is that most of the tests seem to have taken a shorter amount of time when roulette selection was used than the test with the corresponding input values in which tournament selection was used. In terms of the specific values of probability that were most effective, it is impossible to discern as with each of my 3 experiment suites, a different combination produced the quickest results (circled below).



TEST NUMBER	1	2	3	4	5		1	2	3	4	5
SELECTION TYPE	Tournament	Tournament	Tournament	Tournament	Tournament		Roulette	Roulette	Roulette	Roulette	Roulette
INITIAL SIZE	1000	1000	1000	1000	1000		1000	1000	1000	1000	1000
MAX GEN	10000	10000	10000	10000	10000		10000	10000	10000	10000	10000
CROSS PROB:	0.7	0.7	0.7	0.3	0.5		0.7	0.7	0.7	0.3	0.5
MUT PROB	0.6	0.2	0.4	0.4	0.4		0.6	0.2	0.4	0.4	0.4
BFDEACGCF+GGFAIFGCB=FHEGEGHJE											
AVERAGE TIME (ms) across 30 iterations	2.69170005	3.523177815	2.066196115	2.296125036	2.224717379		2.019603667	1.974939286	3.285805316	2.253110368	2.3072949
EJJCGFFHD+IIAAGHCB=CCDEGFEG											
AVERAGE TIME (ms) across 30 iterations	2156.04099	1682.998174	1434.307584	949.0183114	1162.386419		1518.98521	1354.575407	1481.832986	947.7508064	1242.391013
BBEEFFEE+EEBBFEFE=AAAAACCD											
AVERAGE TIME (ms) across 30 iterations	4.970991091	3.226747632	2.949884577	3.074895364	2.251670588		2.537753333	2.760265483	1.680010789	2.128328167	1.712041294