After running the code multiple times with different length input strings and checking each set up 10 times to make sure that my numbers were not just outliers I came to a few conclusions:

For target phrase, it naturally took more generations to iterate through and find the match. Three letter words averaged at 2 generations when it was only letters being checked. When numbers and all punctuation/symbols were added to the possibilities, that average went up to 4.1. Longer input phrases took longer numbers of generations, with just basic letters taking 2.8 generations to get a four-letter word, 3.9 generations for a five-letter word, 15.5 for a seven-character two word string and 219.5 for a 16 character four word sentence. Now when numbers and punctuation are added the averages all go up. Four letters take 17.7 generations (which was slightly off because of a major outlier increasing the average. Otherwise it would be about 4.4), five take 10.1 generations, six and a punctuation take 25.9, and then seven and two punctuations take 191.3.

From there I went to test whether the Half breed or the Random breed worked better. This was a resounding result that random breed was more efficient and took anywhere between 1.4 and 3.3 times as many generations to calculate the target string.

Mutation rate could go down to make the system more efficient but after it got too high, the program could no longer calculate the string within any reasonable number of generations. Thus, the lower the mutation rate the fewer generations.

Then as mentioned at the end of the assignment, the more population members the quicker it will be to calculate the string.

The mating factor was all over the place, with a lower factor being better when using all the characters, but not for just letters. I tired to figure out what was happening but there were multiple very large values which came up 2/10 times in my run which threw off my calculation, but because they then continued to arise had to be left as legitimate values, not just statistical flukes.

Finally, as far a run time, there was no clear winner on which took the longest because I wrote far more functions than were needed to simplify my life and help with debugging. Naturally main took the longest, but then buildPopulation was also up there in length. The remainder were too fast for the code to pick up and be able to output.

The exponential factor was maxed at 6. Lower than that and higher than that the words took more generations but when it was at six the 16 character long string took 55% fewer generations to calculate.