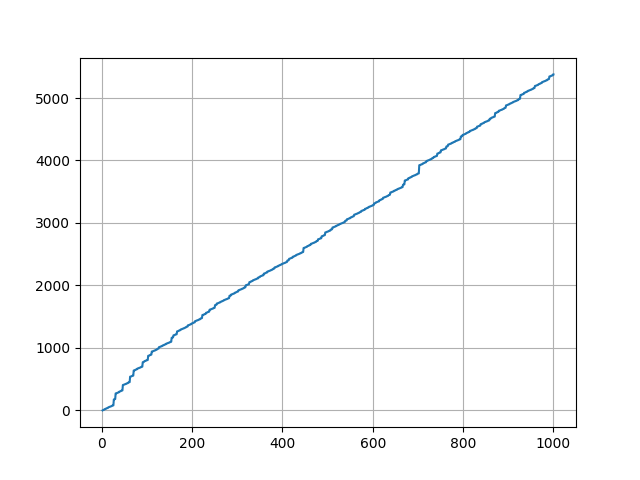
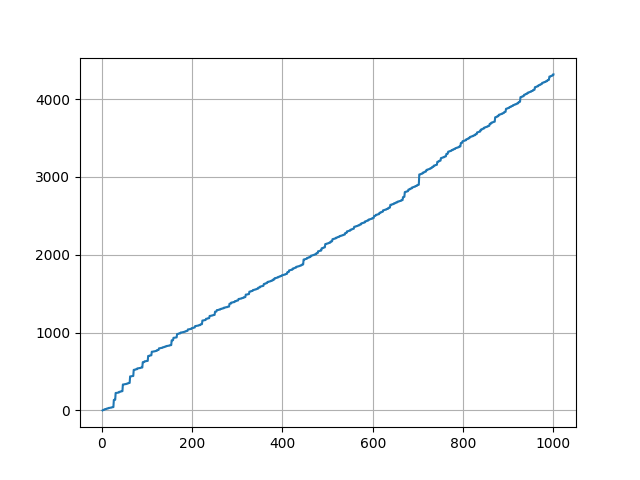
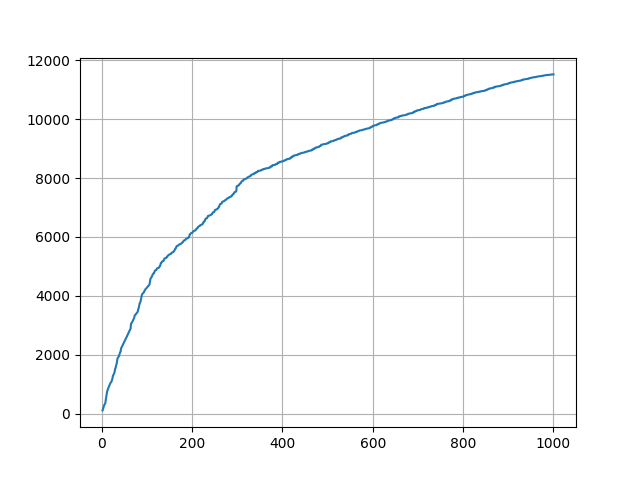
1. The curve is fairly linear, which shows that as increases the number of times the loop is iterated increases roughly in proportion to the increase in . From the plot, it looks like each time is increased by 1, the program goes through about 5 more loops.



1. I tried three different orders. The first one was increasing by 1 each time, and the performance was similar to the first part. The only changes came because I changed how I initialized the dictionary for the check boxes and how I checked the boxes after I checked that a number converged. When I initialized the dictionary, I prematurely checked off all powers of 2, since clearly those will converge. When I checked a number in the loop, I checked off not only its box in the dictionary, but also itself multiplied by any power of 2, so long as the result was less than . For example, if I checked 3 for , I also checked off 6, 12, 24, 48, and 96. Effectively, then, I chose the order 2, 4, 8,…, 3, 6, 12,…, 5, 10, 20,… and so on.



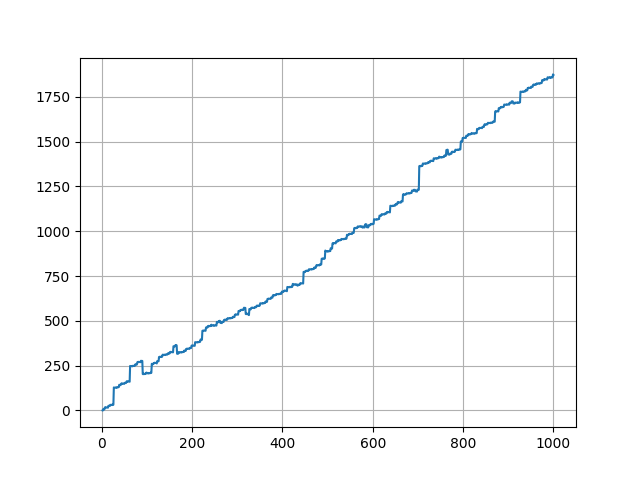
For the second order, I went from down to 2, decreasing by 1 for each loop. This was considerably slower, but interestingly seems to follow a logarithmic trend as increases.



The final order I tried was a little cleverer. Instead of predefining an order, I went through and checked all the values up to and including before actually doing the check, and paid attention to the order in which a number converged to 1. This order was then reversed so that all the dependencies would be checked before the numbers that depended on them. For example, 7 converges in the following sequence:

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

This sequence would then be reversed and appended to the list with the best order, so that all the numbers which follow 7 in the sequence will be checked first. In particular, 22 will be checked before 7, so it only takes one iteration through the loop to ensure that 7 converges. This should be the optimal ordering, but I’m not sure how to prove it. The reason it takes longer than 1000 iterations is that we have to check numbers greater than 1000 in some of the sequences, and numbers greater than 1000 are not included in the list of check boxes.



1. It seems from the tests that extending the list by multiplying by ensures the fewest possible iterations of the Collatz loop. In other words, the list of check boxes should have a length of . This is verified empirically for a couple values of , but has not been proven. In the figures, is the size of the check list divided by .

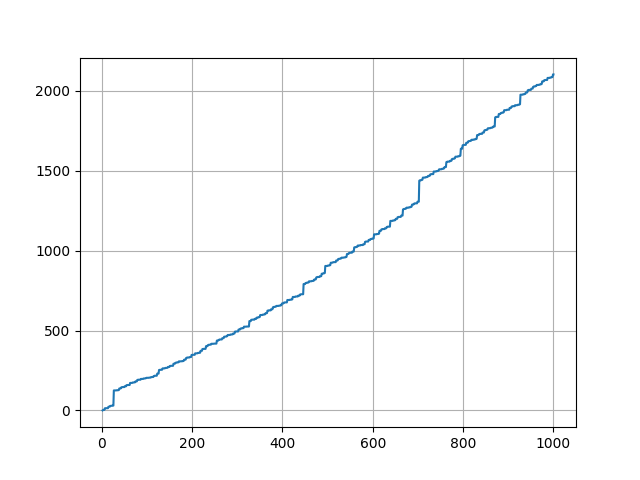


Figure 1: m = 1

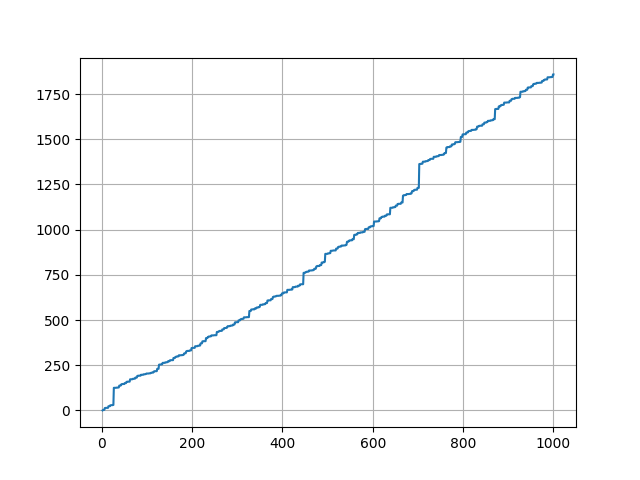


Figure 2: m = 3

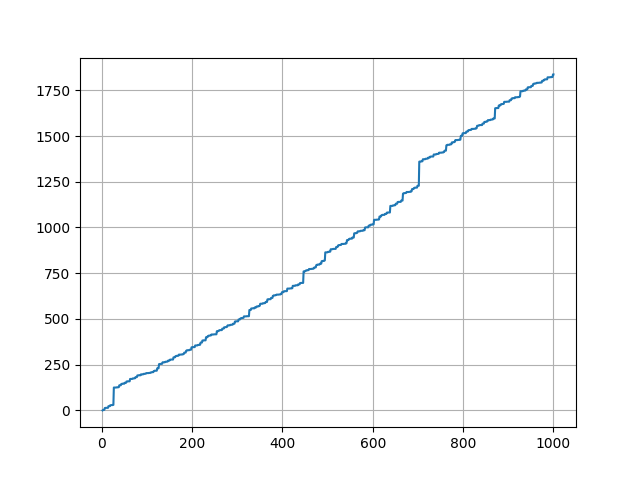


Figure 3: m = 5

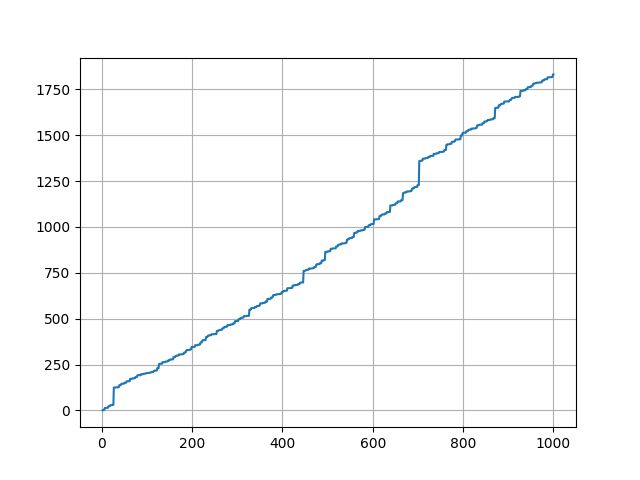


Figure 4: m = 10

The next set of images correspond to .

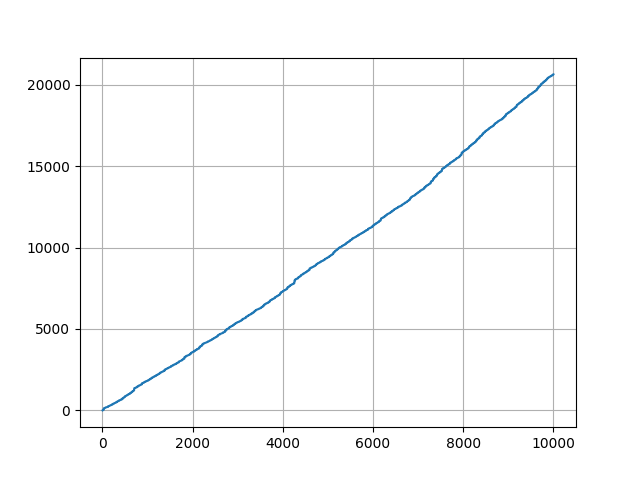


Figure 5: m = 1

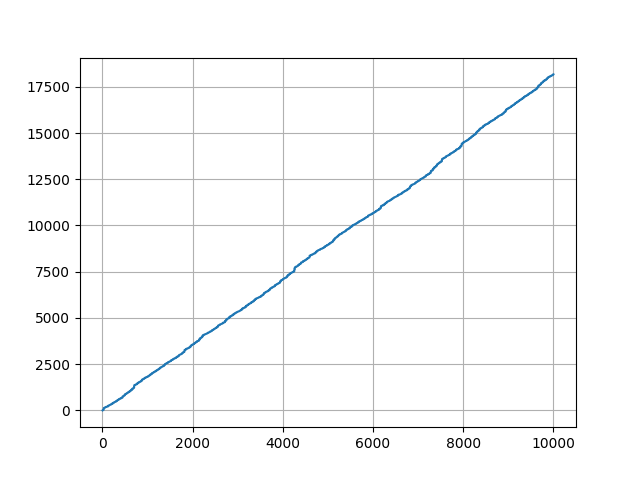


Figure 6: m = 3

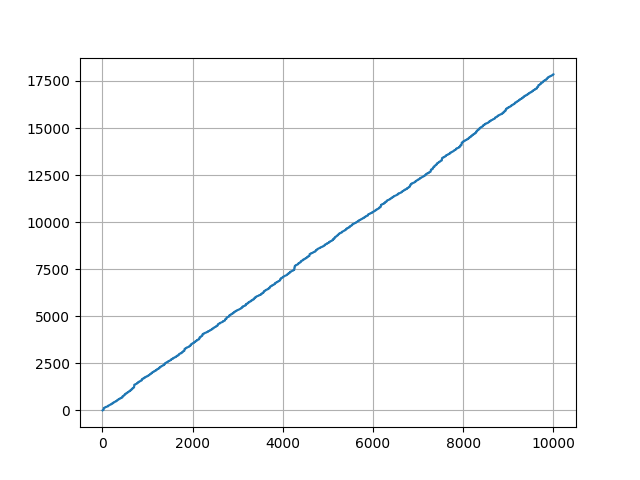


Figure 7: m = 5

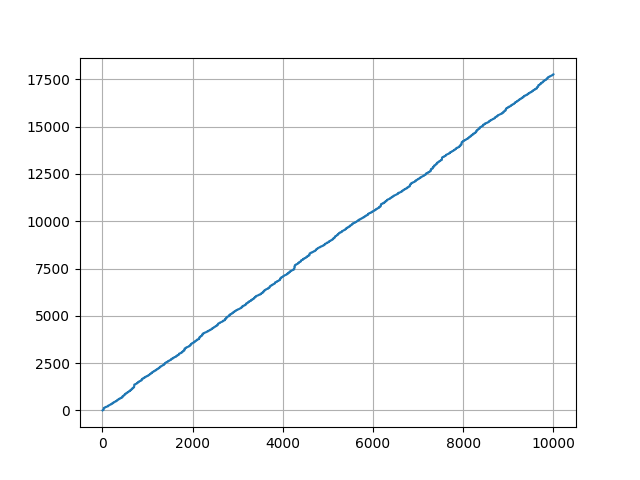


Figure 8: m = 10

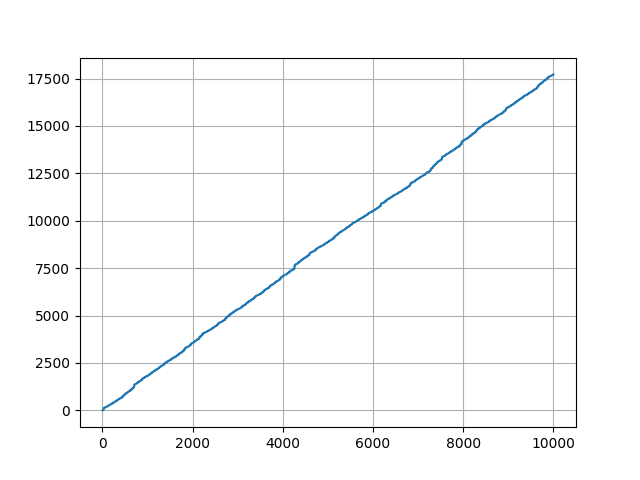


Figure 9: m = 25

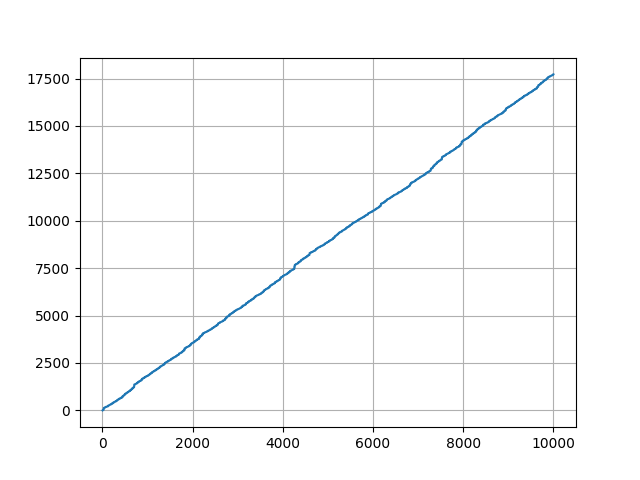


Figure 10: m = 50

1. Every number that converges must pass through . This is because must be reached through one of two steps – either for an odd or for an even . Since the solution to the first equation is , which is not positive or odd, the only possibility is the solution to the second equation, which is . Moreover, every number that converges must also pass through 4, 8, and 16. To reach we must have either for an odd or for an even , and since the first equation has no integer solutions we must pass through , the solution to the second equation. Similarly, to reach 4, we must have

for an odd or for an even , and since the solution to the first equation is , which is impossible, we must pass through . Finally, to reach 8 we must have

for an odd or for an even , and since the first equation has no integer solutions we must pass through .

Another thing we note is that when a number converges, it must finish converging with a descent of powers of 2. This descent of powers of 2 must start at an even power of 2. To see why, consider how we reach odd powers of 2 in the sequence of convergents. Again, either or , where is odd. But if is odd, is congruent to 2 modulo 3, so the first equation has no integer solutions, and certainly no odd integer solutions. Thus, if we pass through for odd, we must also pass through first.

As a result of this, we must also pass through some solution of .