Niels Turley

Fermat Report

1. Graphical user interface, application, email, website

   Description automatically generatedWorking screenshot:
2. Code
3. Graphical user interface, text, application, email

   Description automatically generatedDiscussion of input disagreement:

Carmichael numbers are composite numbers that can pass Fermat’s test for all *a* relatively prime to *N*. An imperfect solution to this is to use the Fermat test *k* times, choosing a different *a* value each time in hopes that one of them will not be relatively prime to *N*. A better solution was devised by Miller-Rabin, which repeatedly squares the *an-1* value, checking if it deviates from 1 (mod *N*). If done *k* times it can catch more Carmichael numbers.

1. Time and space complexity:
   1. mod\_exp(x, y, N)
      1. **Time complexity: O(n3).** This function uses recursion, which halts after n iterations since we are dividing y by 2 every time. Within each recursion, we have several checks and a choice between [integer exponentiation, modulo] and [multiplication, integer exponentiation, modulo]. The worst case scenario would be the multiplication of the two n-bit integers, which takes n2 time. So total time complexity is O(n iterations \* n2).
      2. **Space complexity: O(n).** There are no big space occupying variables (like lists or matrices), so that remains constant. However, the call stack will need n elements for the recursion.
   2. fermat(N, k)
      1. **Time complexity: O(k \* n3).** This function uses a for loop which runs k times. Each run of the for loop has a check and calls mod\_exp, which has the dominating O(n3) time complexity. The total time complexity is O(k iterations \* n3).
      2. **Space complexity: O(n).** There are no big space occupying variables (like lists or matrices), so that remains constant. The for loop does not save any variables, so the only spot memory comes in is during the mod\_exp recursion, which has space complexity of O(n).
   3. miller\_rabin(N, K)
      1. **Time complexity: O(k \* n3 \* log2(n)).** This function uses a for loop which runs k times. Each run of the for loop has a couple checks, runs mod\_exp once O(n3), and moves on to the recursive miller\_rabin\_helper function if it passes the first check, which has O(n3 \* log2(n)) time complexity. The total is O(k iterations \* n3 \* log2(n)).
      2. **Space complexity: O(log2(n)).** There are no big space occupying variables (like lists or matrices), so that remains constant. The for loop does not save any variables, so the only spot memory comes in is during the miller\_rabin\_helper recursion, which has space complexity of O(log2n).
   4. miller\_rabin\_helper(a, exp, N)
      1. **Time complexity: O(n3 \* log2(n)).** This function uses recursion, which halts after the returned value from mod\_exp equals something different than 1, or when exp can no longer be divided by 2 evenly. The worst case scenario would be the latter for certain exp = 2x (which would mean the recursion halves exp all the way down to 1). This gives the function log2(n) recursions, each of which calls mod\_exp O(n3). The total time complexity is O(n3 \* log2(n) iterations).
      2. **Space complexity: O(log2(n)).** There are no big space occupying variables (like lists or matrices), so that remains constant. As explained in the time complexity, though, the call stack will need log2(n) elements for the worst case recursion.
2. Probability equations:

Fermat’s algorithm will incorrectly return ‘prime’ for at most half of the values of *a* < *N*. This error can be reduced by picking and testing *k* values of *a*. In other words, each test reduces our chance of error by 50%. So, to calculate the probability that Fermat’s algorithm is correct, the formula is:

Miller Rabin’s algorithm will incorrectly return ‘prime’ for at most ¼ of the values of *a* < *N,* due to its square root testing. This error can also be reduced by picking and testing *k* values of *a*. In other words, each test reduces our chance of error by 75%. The probability that Miller Rabin’s algorithm is correct is: