Problems

Due 1 June:

1. Implement both recursive and iterative versions of power-mod, to compute b^k mod N given b, k, and N; and to compute b^k given only b and k (but not N). Your iterative version must use only a bounded amount of memory beyond what is needed for the input and output variables. Compare the running times of these implementations, and compare them to the naive algorithm (repeated multiplication).

2. Implement the Fermat test: specifically, given N and b it should apply the Fermat test to N with base b. For each number x less than 10000 determine whether x is definitely prime, definitely composite, or a Fermat pseudoprime (to every possible base). On average, how many tests are required to make this decision?

3. Given N and α, how expensive is it to use the Fermat test to locate a value p such that $p > N$ and p is prime with probability at least $1 - \alpha$? I am looking for an empirical answer here, but I will accept a theoretical one if it is thorough and well-written.

Due 8 June:

4. Compile a list P of all the primes of 5 or fewer digits. For 1000 randomly chosen 5-digit numbers n, test n for primality using the Fermat, Solovay-Strassen, and Miller-Rabin tests, using one randomly chosen base b for each test. Compare their average running times and rates of error. Do the same with 1000 randomly chosen 10-digit numbers. Note: You can use your list P to verify the accuracy of the tests. Make sure you do not include the time to verify the accuracy in your data on the running time for the tests themselves.

5. Suppose that $f\colon [a,b] \to [0,M]$ is a continuous function and `A(x,e)` is an algorithm which returns an approximation of $f(x)$ with error less than e. How expensive is it to use the Monte Carlo method to compute k decimal digits of $\int_a^b f$ with 95% confidence? I am looking for a theoretical answer backed by empirical evidence.

6. Use depth-first search to write python code which solves 2-CNF-SAT. Demonstrate empirically the asymptotic efficiency of your solution.

Due 15 June:

7. Write three python functions:

1. To turn an instance of 3-CNF-SAT into an instance of CLIQUE.
2. To turn an instance of CLIQUE into an instance of VERTEX-COVER.
3. To approximate a minimal vertex cover.

What results when you compose these? Is the result in some sense an approximate solution to 3-CNF-SAT?