Prof. Friedrich Eisenbrand
Alfonso Cevallos

Discussions from: March 17, 2015

Computer Algebra

Spring 2015

Assignment Sheet 3

Exercises marked with a $\star$ can be handed in for bonus points. Due date is March 31.

Exercise 1

Recall that a composite number N is called Carmichael if $a^{N-1} \equiv 1 \mod N$ for each a with $gcd(a,N) = 1$. Prove that an odd, composite number N is Carmichael if and only if, for every prime factor $p|N$, we have $p^2 \nmid N$ and $p-1|N-1$. (Hint: use the fact that if $M = p^k$, $k \geq 1$ is the power of a prime, then $\mathbb{Z}_M^*$ is cyclic; i.e. there is a number $c \in \mathbb{Z}_M^*$ whose order is $|\mathbb{Z}_M^*|$).

Exercise 2

Let N be a Carmichael number, with prime decomposition $N = \prod_{i=1}^{k} p_i$. What is the probability that the Fermat Test will answer "composite" for input N? Assume that the Fermat test picks a random number between 1 and $N-1$.

Exercise 3

Assuming $\pi(x) := \{p \leq x \mid p \text{ is prime}\} \sim x/\log x$, show the following: for each $\epsilon > 0$, there exists $c > 0, N \in \mathbb{N}$ such that, for each $n \geq N$, one has $\pi((1+\epsilon)n) - \pi(n) \geq cn/\log n$. Conclude from this that, if $f(n) = \frac{|\{\text{primes of bit size } n\}|}{|\{\text{numbers of bit size } n\}|}$, then $f(n) = \Omega(1/n)$.

Exercise 4 ($\star$)

In this exercise we analyse how to use the Miller-Rabin algorithm (MR) to generate large primes, and we implement the RSA cryptosystem. Parts 4 and 6 require implementations in Python.

1. Suppose you buy a lottery ticket, and you know that the winning probability of any ticket is one in a million. Then you show it to your friend John, and he predicts that it is the winning ticket. You know that John is a great foreteller, and his predictions are correct with probability p. How high must p be, for you to have a 90 % confidence that you actually have the winning ticket? (Hint: p must be a lot higher than 0.9.)

2. For a fixed n, you generate an n-bit odd number N uniformly at random. Now, suppose you run k iterations of MR, and the output is "probably prime" every time. How large must k be for you to convince yourself that N is prime with probability at least $1-\epsilon$? (Hint: estimate the original probability of N to be prime.)

3. Suppose you run MR over a composite number N. What is the expected number of iterations you will need to run, to detect that N is actually composite?

4. Implement MR. Then implement an algorithm that takes as input a number n, and uses MR to output a random number N of n bits, which is prime with probability at least $1-\epsilon$.

5. Consider the previous algorithm. In expectation, how many numbers N' will this algorithm generate before finding a prime? And what is the expected number of iterations of MR needed, in total? Taking into account the complexity of MR, what is the expected complexity of your algorithm, in terms of n and ϵ?

6. Implement an algorithm that uses 2 primes (obtained via 4), to generate private an public keys for the RSA cryptosystem, and an algorithm that decodes a message coded through this scheme. Test your algorithm on the following input. Public key: 43, 9379; private key: 2563, 9379; encrypted message: 2982. What is the original message?

7. Suppose that the public key is a number $M = pq$, and the bit size of M is m. We commented in class that the use of Miller-Rabin for primality testing means that there is a small probability that the scheme will fail (if p or q is not prime). But we argued that an adversary can simply guess any sent message, with probability 2^{-m}, so if the failure probability is of the same order of magnitude, we can consider it negligible. What value would you pick for ϵ to achieve this?

Exercise 5

A good riddle should be hard to answer, but once you have the answer, it should be easy to check that it is correct. We analyse some problems that have this property, from an algorithmic point of view. (This is an introduction to the complexity class NP.)

1. An algorithm is said *polynomial time* if its running time is polynomial in the input size. A proper factor of an integer N is a number distinct from N that divides N. Show that a polynomial time algorithm for the problem (P) implies the existence of a polynomial time algorithm for the problem (Q).

   (P) INPUT: A pair of positive integers $l \leq N$.
   OUTPUT: 'YES' if N has a proper factor greater than l; 'NO' otherwise.

   (Q) NPUT: A positive integer N. OUTPUT: A factorization of N in prime numbers.

2. If you *know* the answer to a problem for a given input, how do you convince someone that you are right? A YES-instance (resp. NO-instance) for problem (P) is a pair l, N such that the answer to (P) for input l, N is YES (resp. NO). We say that $S : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is a 'YES'-certificate (resp. 'NO'-certificate) for (P) if:
   a) $S(l, N)$ is of size polynomial in $\log(N)$, and
   b) There exists a polynomial time algorithm that, given l, N and $S(l, N)$ as input, answers 'YES' (resp. 'NO') if and only if l, N is a YES- (resp. NO-) instance.

   Give YES- and NO-certificates for the problem (P) from part 1. (Hint: you can assume that we can test if a number is prime in polynomial time).