Tutorial 14, Design and Analysis of Algorithms, 2019

1. Lucas numbers are defined as: $L_1 = 1, L_2 = 3$, and for $n > 2, L_n = L_{n-1} + L_{n-2}$. Prove both the parts using mathematical induction.

   (a) Find the GCD (L_n, L_{n-1}) using Euclid's GCD algorithm.

   (b) Find integers x_n and y_n so that $(L_n, L_{n-1}) = x_n L_n + y_n L_{n-1}$ using Euclid's Extended GCD algorithm.

2. Using extended Euclid's algorithm for GCD solve the following congruences, showing all the steps involved.

   (a) $49x \equiv 5000(\mod 999)$

   (b) $495x \equiv 5001(\mod 999)$

3. Show how to compute multiplicative inverses modulo a prime p via a single exponentiation. Either prove that this also works modulo composite n or give a counterexample.

4. To see if a number, say 562437487, is divisible by 3, you just add up the digits of its decimal representation, and see if the result is divisible by 3 ($5+6+2+4+3+7+4+8+7 = 46$, so it is not divisible by 3). To see if the same number is divisible by 11, you can do this: subdivide the number into pairs of digits, from the right hand end $(87, 74, 43, 62, 5)$, add these numbers, and see if the sum is divisible by 11 (if it is too big, repeat). How about 37? To see if the number is divisible by 37, subdivide it into triples from the end $(487, 437, 562)$, add these up, and see if the sum is divisible by 37 . This is true for any prime p other than 2 and 5. That is, for any prime $p \neq 2, 5$, there is an integer r such that in order to see if p divides a decimal number n, we break n into $r-$tuples of decimal digits (starting from the right-hand end), add up these $r-$tuples, and check if the sum is divisible by p.

   (a) What is the smallest such r for $p = 13$?

   (b) What is the smallest such r for $p = 17$?

   (c) Prove that r is a divisor of $p - 1$.

5. By making use of efficient algorithms, find the last three digits in the decimal expansion of $3^{20182018201820182018}$

6. Write an algorithm for computing a^b where a and b are positive integers using the repeated squaring method. Assuming that (1) $a = O(n)$, and $b = O(n)$; (2) you can multiply two m bit numbers in $O(m^2)$ time; and (3) you can add two m bit numbers in $O(m)$ time; find the best possible upper bound on worst case time complexity in $O()$ notation as a function of n.

7. Let `IROOT`(n, m) be a function that computes the integral part of m'th root of n (m and n are positive integers):
   `IROOT`$(n, m) = \lfloor n^{1/m} \rfloor$
   Give a polynomial-time algorithm for `IROOT` and find its time-complexity in terms of n (you can assume that $m \leq n$).