Prof. Friedrich Eisenbrand
Yuri Faenza

Discussions from: March 05, 2013

Computer Algebra

Spring 2013

Assignment Sheet 2

Exercises marked with a $\star$ can be handed in for bonus points. Due date is March 19.

Exercise 1
Let $f : \mathbb{N} \to \mathbb{R}_+$ be a function with $f(a) + f(b) \leq f(a+b)$. Show that $f(1) + f(2) + f(4) + f(8) + \ldots + f(n) = O(f(n))$.

Exercise 2
Let $x \in \mathbb{R}$ and $n \in \mathbb{N}_{\geq 1}$. Show that $\lfloor \lfloor x \rfloor / n \rfloor = \lfloor x/n \rfloor$; in particular, $\lfloor \lfloor a/b \rfloor / c \rfloor = \lfloor a/bc \rfloor$ for all positive integers a, b, c.

Exercise 3
Let $a, b \in \mathbb{N}$ be odd numbers with $a - b = 2^k$ for some $k \in \mathbb{N}$. Show that a and b are coprime.

Exercise 4
Let $N = pq$, where $p \neq q$ are primes. Assuming that the roots of a polynomial can be computed efficiently, show that given only N and $\varphi(N)$, one can compute the prime factors p and q efficiently.

Exercise 5
Implement the fast modular exponentiation function.

Exercise 6 ($\star$)
Recall the Fibonacci numbers: $F(0) = 0$, $F(1) = 1$, and $F(n) = F(n-1) + F(n-2)$ for $n \geq 2$. Consider the following two algorithms for computing the $n-$th Fibonacci number.

```
Fib_1(n):  input= n ∈ Z_+

  if n == 0 or n == 1
      return n

  return Fib_1(n-1) + Fib_1(n-2)
```

`Fib`$_2(n)$`:  input=` $n \in \mathbb{Z}_+$

`Let` $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1}$

`Return` a

Note: The computation on A is left intentionally vague. How can this be done efficiently?

a) Prove they are correct.

b) Give a (smallest possible) estimate of their running time.

c) Implement them and compare their running time for different values of n.

Exercise 7 ($\star$)

a) Implement the Karatsuba algorithm for multiplication.

b) Implement a function `randbits(n)` that returns a random number of bit length exactly n (that is, it returns a random sequence of n bits with the most significant bit always 1).

c) Add a function that tests your implementation of Karatsuba by calling it repeatedly on random numbers of varying lengths n and m, and comparing the result with simple multiplication.

d) Benchmark the running time of your algorithm for varying lengths of inputs compared to the running time of simple multiplication (implemented in the accompanying file `assignment02.py`). Determine the bit length at which Karatsuba becomes faster than simple multiplication.

Note: You can use the functions in the accompanying Python source code `assignment02.py` and/or those implemented in Sage.