Prof. Friedrich Eisenbrand
Nicolai Hähnle

Discussions from: March 9, 2010

Computer Algebra

Fall 2010

Assignment Sheet 2

Exercises marked with a $\star$ can be handed in for bonus points. Due date is March 23.

Exercise 1
Use the extended Euclidean algorithm to compute the (multiplicative) inverse of $22 \in \mathbb{Z}_{71}^{\star}$.

Exercise 2
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 3
Consider the problem of deciding whether the equality $a^\alpha b^\beta = c^\gamma$ holds for given natural numbers $a, b, c, \alpha, \beta, \gamma \in \mathbb{N}$.

1. Is it efficient to decide this question by simply evaluating the sides of the equation? Why?
2. Is $45^{12407} \cdot 35^{9381} = 105^{17315}$? Why?

We say that a set of pairwise coprime numbers $p_1, \ldots, p_k$ is a factor basis for a, b, and c if we can write a, b, and c as products of powers of the p_j, e.g. $a = \prod_{j=1}^{k} p_j^{a_j}$ for some natural numbers $a_1, \ldots, a_k \in \mathbb{N}_0$.

3. Give an efficient algorithm to decide $a^\alpha b^\beta = c^\gamma$ given a factor basis for a, b, and c. Show the correctness of your algorithm and analyze its running time in O-notation. (You only need to show an upper bound!)

Exercise 4 (⋆)
Consider the following algorithm.

FACTORBASIS($S = \{s_1, \ldots, s_m\} \subset \mathbb{N}$)

```
1  S ← S \ {1}
2  while there exist a ≠ b ∈ S such that g = gcd(a, b) ≥ 2
3     then S ← (S \ {a, b}) ∪ {g, a/g, b/g}
4          S ← S \ {1}
5  return S
```

1. Show that FACTORBASIS(S) returns a factor basis of S.

2. Analyze the running time of FACTORBASIS in O-notation. Show that it is polynomial in the bit size of the input. (You only need to show an upper bound!)

   Hint: What is the time needed for step 2? To bound the number of iterations, consider the function $\phi(S) = \prod_{a \in S} a$.

Exercise 5 (⋆)
The Subversion repository (update using `svn update`) contains a new program `test_karatsuba` and a function `integer::karatsuba` that currently falls back to the basic multiplication.

1. Implement the Karatsuba algorithm for multiplication, using the tests in `test_karatsuba` to check whether your implementation is correct.

2. Use the timing functions in `test_karatsuba` to determine the approximate *cross-over* point where your implementation of Karatsuba multiplication becomes faster than simple multiplication.

Submit the results of your timing tests together with the patch containing your Karatsuba implementation.

Note: It may well be that the cross-over point between simple multiplication and Karatsuba multiplication is at a very large number of digits. If the cross-over point is not reached for multiplications taking more than a minute, you should check whether your timing results approximately fit the expected asymptotic growth of $O(n^2)$ vs. $O(n^{\log 3})$. If they do, an "educated guess" based on extrapolation is sufficient.