Prof. Friedrich Eisenbrand
Alfonso Cevallos

Discussions from: February 17, 2015

Computer Algebra

Spring 2015

Assignment Sheet 1

Exercises marked with a $\star$ can be handed in for bonus points. Due date is March 3.

Exercise 1
Sort the following functions according to their asymptotic growth. Indicate which pairs of functions satisfy $f = O(g)$, $f = \Omega(g)$, and $f = \Theta(g)$, motivating your answer.

$2^{3+\log n}$, $\sqrt{n}$, $\log n^n$, 4^n, 13, $\log n^{1337}$, $2^{\log^2 n}$, $\log n$, $e^{\log n}$, $3n$, $n^6 - 5n^2$, $-n^6 + 5n^2$, $2^{4 \log n}$, 2^n, $\log^2 n$

Note: $\log n$ without an indicated base is always base 2.

Exercise 2
Let $f, g : \mathbb{N} \to \mathbb{R}_+$. Show that $f = O(g)$ if and only if $\limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty$.

Exercise 3
Let $f, g : \mathbb{N} \to \mathbb{R}_+$. We say $f \sim g$ (f is asymptotically equal to g) when $f(x)/g(x) \to 1$ as $x \to \infty$.

a) [$\star$] Show that $f \sim g$ implies $f = \Theta(g)$. Is the converse also true?

b) Show that $f \sim g$ implies $f = (1 + o(1))g$. Is the converse also true?

c) [$\star$] Let $F(n) = \sum_{i=1}^{n} f(i)$ and $G(n) = \sum_{i=1}^{n} g(i)$. Show that $f \sim g$ and $G(n) \to +\infty$ when $n \to \infty$ implies $F \sim G$.

Exercise 4 ($\star$)
Let $f(n) = n \log n$ and $g(n) = \log(n!)$. Show which of the following relations are true: $f = O(g)$; $f = \Omega(g)$; $f = \Theta(g)$.

Exercise 5
Let A_1 and A_2 be algorithms for the same problem which run for $T_1(n) = 5n^2$ and $T_2(n) = 1000 n \log n$ machine operations on an input of size n, respectively. Let M_1 be a machine that can execute 10^{10} machine operations per second, and M_2 a machine that can execute 10^6 machine operations per second. For which values of n is A_1 on M_1 faster than A_2 on M_2?

Exercise 6 ($\star$)
Download the accompanying file **assignment01.py**. It provides an implementation of simple multiplication on Python, as well as other tools.

1. Implement in Python a function that takes as input n and returns a random number of bit length n (that is, it returns a random sequence of n bits with the most significant bit always 1).

2. Implement in Python the Karatsuba algorithm for multiplication.

3. Add a function that tests the correctness of your implementation of Karatsuba. It should call it repeteadly on random numbers of varying lengths n and m, and compare the result with simple multiplication.

4. Benchmark the running time of your algorithm for varying input lengths, compared to the running time of simple multiplication. Determine the bit length at which Karatsuba becomes faster than simple multiplication.