• A UNIFIED VIEW OF THE COMPLEXITY OF
EVALUATION AND INTERPOLATION*

Ellis Horowitz

TR 73-162

April 1973

• Department of Computer Science
Cornell University
Ithaca, New York 14850

*This research was supported by the National Science Foundation
under grant GJ-33169.

A Unified View of the Complexity of Evaluation and Interpolation*

Ellis Horowitz

Abstract

Four problems are considered: 1) from an n -precision integer compute its residues modulo n single precision primes; 2) from an n -degree polynomial compute its values at n points; 3) from n residues compute the unique n -precision integer congruent to the residues; 4) from n points compute the unique interpolating polynomial through those points. If $M(n)$ is the time for n -precision integer multiplication, then the time for problems 1 and 2 is shown to be $M(n)\log n$ and for problems 3 and 4 to be $M(n)(\log n)^2$. Moreover it is shown that each of the four algorithms are really all instances of the same general algorithm. Finally it is shown how preconditioning or a change of domain will reduce the time for problems 3 and 4 to $M(n)(\log n)$.

*This research was supported by the National Science Foundation under grant GJ-33169.

A Unified View of the Complexity of Evaluation and Interpolation

Ellis Horowitz

Introduction

The study of the theory of algorithms is an attempt to discover the intrinsic complexity of certain basic computational processes. This study has been subjected to increasing attention over the past few years with some notable successes. For example, one of the most important developments towards the understanding of the complexity of integer multiplication has been an algorithm by A. Schönhage and V. Strassen as presented in [7] pp. 269-275. Their method shows how to multiply two n -precision integers in not more than $O(n(\log n)(\log \log n))$ operations and in [7], pp. 275-277, it is shown how an integer division algorithm with the same computing time bound can be obtained. Using fast integer multiplication and division there has been intensive work on other important arithmetic and algebraic algorithms. Specifically, the problems that we will be dealing with are the following:

- 1) from an n -precision integer compute its residues modulo n single precision primes;
 - 2) from an n -degree polynomial compute its values at n points;
- and going in the reverse direction,

- 3) from n residues compute the unique n -precision integer which is congruent to the residues;
- 4) from n points compute the unique interpolating polynomial through those points.

These four algorithms have a very wide applicability. My own interest in them stems from the fact that their general use has become recognized for speeding up many algebraic algorithms. Such efficient algorithms are crucial to effective mathematical symbol manipulation systems. Regarding these processes as transformational techniques, orders of magnitude decreases in computing times for such polynomial tasks as computing powers and greatest common divisor calculation ([6] and [2]) have been effected.

The classical methods for all of the above problems (1) - (4) take $O(n^2)$ operations. Recently a host of papers [1,3,4,5,8,9] have been published which present a variety of algorithms for reducing one or more of these problems below $O(n^2)$. All of these algorithms have similarities such as their use of fast integer multiplication and division or the use of recursion. Unfortunately the presentations of each method have been varied and at the moment there are at least two competing algorithms for problem 4 which have the same time bound. In this paper it will be shown that the best algorithms for all of these problems can be viewed in a single framework and that each particular problem is merely a specific instance of what is essentially the same algorithm.

2. The Algorithms

Before we rigorously define the problems to be tackled, let us review some of the basic complexity results that we will need. As stated before, in [7] pp. 269-277 it is shown that the time for n -precision integer multiplication and division is $O(n(\log n)(\log \log n))$. Also in [7] pp. 363 and 441 we find that the time needed to multiply or divide two polynomials of degree at most n is $O(n \log n)$. Hopcroft in [12] clearly shows the relationship between a fast integer (polynomial) multiplication algorithm and a fast integer (polynomial) division algorithm which has the same time complexity. Sieveking in [11] also shows how to divide truncated power series in $O(n \log n)$ steps. We assume that the coefficients are single precision fixed or floating point numbers and we are not concerned here with analyzing the error propagation as the algorithm proceeds. Both the integer and polynomial division algorithms depend essentially upon the computation of the reciprocal of the divisor and then applying the fast multiplication algorithm. The final result we need is a relatively new gcd algorithm due to A. Schönhage, [10], which takes at most $O(n(\log n)^2(\log \log n))$ steps. His algorithm immediately generalizes to what is commonly called an extended gcd algorithm; namely given P and Q we can find R and S such that

$$PR + QS = \gcd(P, Q)$$

where P, Q, R and S are either n -precision integers or polynomials of degree at most n .

Let $M(n)$ represent the time to multiply or divide two n -precision integers or n -degree polynomials. We will take

$M(n) = n \log n$ throughout the rest of the paper, noting here that an additional $\log \log n$ factor must be appended to all of the resulting computing times. Thus the time for an extended gcd calculation is $M(n)(\log n)$.

Now consider the binary tree as given below.

Figure 1: Binary Tree

As we "go up" the tree the level numbers increase, while the root of the tree is at the bottom at level 1. The i -th level has 2^{i-1} nodes and a tree with k levels has a total of 2^k nodes. The nodes at higher levels are said to be ancestors and at lower levels descendants of a node at level i . In this paper we will be interested in computing different functions at every node of a binary tree. So for example an algorithm for moving down the tree is

Algorithm: Move Down a Tree

Input: $n = 2^{k-1}$ values $t_{k,1}, \dots, t_{k,n}$;

Output: a single value, $t_{1,1}$;

```

for i ← k-1, ..., 1 do
    l ← 1;
    for j ← 1, ..., 2i-1 do
         $t_{i,j} \leftarrow t_{i+1,l} \odot t_{i+1,l+1}$ 
        l ← l+2;
    end
end
end

```

Subsequently, we will be most concerned about the cost of the operation $\odot$, which will be denoted by $C(i)$. From $C(i)$ and the above algorithm the expression for the total time needed to compute every node on this tree is

$$\sum_{1 \leq i \leq k-1} 2^{i-1} C(i)$$

Similarly an algorithm which computes elements as we go up the tree would be

```

for i ← 2, ..., k do
    l ← 1;
    for j ← 1, ..., 2i-1 by 2 do
         $t_{i,j} \leftarrow t_{i-1,l} \odot t_{i-1,l+1}$ 
         $t_{i,j+1} \leftarrow t_{i-1,l+1} \odot t_{i-1,l+2}$ 
        l ← l+2;
    end
end
end

```

We now proceed to the specific problems.

Problem 1: Let u be an n -precision integer and $p_1, \dots, p_n$ single precision primes. Then we wish to compute the n residues $u_i = u \bmod p_i$ which gives us a mixed radix representation for u . We consider the binary tree in Figure 2.

Figure 2: Binary Tree with Moduli

Now if $n=2^{k-1}$ then products on the i -th level have precision 2^{k-i} , $1 \leq i \leq k$. Using our fast integer multiplication we compute the elements going down the tree. Therefore,

$$C(i) = M(2^{k-(i+1)}) = 2^{k-(i+1)}(k-i-1)$$

and the total time to complete the tree is

$$\sum_{1 \leq i \leq k-1} 2^{i-1} M(2^{k-(i+1)}) = 2^{k-2} \sum_{1 \leq i \leq k-1} (k-i-1)$$

$$= 2^{k-2} [k^2 - \frac{k(k+1)}{2} + 1] = O(n(\log n)^2)$$

Now to compute the n residues $u_i = u \bmod p_i$ we reverse direction and proceed to compute functions up the tree. Since u is n -precision and the primes are all near the maximum size of a single precision number, we first compute $u \bmod p_1 \dots p_n = \bar{u}$. Then we continue up the tree.

$$u_{12} = \bar{u} \bmod p_1 \dots p_{\frac{n}{2}} \quad \text{and} \quad u_{22} = \bar{u} \bmod p_{\frac{n}{2}+1} \dots p_n$$

Then we compute

$$u_{13} = u_{12} \bmod p_1 \dots p_{\frac{n}{4}} \quad u_{23} = u_{12} \bmod p_{\frac{n}{4}+1} \dots p_{\frac{n}{2}}$$

$$u_{33} = u_{22} \bmod p_{\frac{n}{2}+1} \dots p_{\frac{3n}{4}} \quad u_{34} = u_{22} \bmod p_{\frac{3n}{4}+1} \dots p_n$$

and so on up the tree until we have

$$u_{1k} = u_1, u_{2,k} = u_2, \dots, u_{2^{k-1},k} = u_{2^{k-1}} = u_n$$

A node on level i is computed using the previously made product of primes at that node plus the element $u_{j,i-1}$ at the descendent node. The computation requires a remainder operation, so

$$C(i) = M(2^{k-i+1}) = 2^{k-i+1} (k-i+1)$$

and the total time for problem 1 is

$$\begin{aligned} \sum_{1 \leq i \leq k} 2^{i-1} \cdot M(2^{k-i+1}) &= 2^k \sum_{1 \leq i \leq k} k-i+1 \\ &= 2^k \left[k^2 - \frac{k(k-1)}{2} \right] = O(n(\log n)^2) \end{aligned}$$

In [9] p. 91, Moenck and Borodin use this construction and show that the time for this operation is only a $\log n$ factor greater than the time for $M(n)$.

Problem 2: Let $P(x)$ be an n -degree polynomial and $x_1, \dots, x_n$ n single precision points. Then we wish to compute the n values $P(x_i)$, $1 \leq i \leq n$. We consider the binary tree

Figure 3: Binary Tree with Moduli

Now, if $n=2^{k-1}$ the products on the i -th level have degree 2^{k-i} , $1 \leq i \leq k$. Using our fast polynomial multiplication we compute the elements going down the tree. Therefore

$$C(i) = M(2^{k-i-1}) = 2^{k-i-1}(k-i-1)$$

and the total time to complete the tree is

$$\begin{aligned} \sum_{1 \leq i \leq k-1} 2^{i-1} M(2^{k-i-1}) &= 2^{k-2} \sum_{1 \leq i \leq k-1} (k-i-1) \\ &= 2^{k-2} \left[k^2 - \frac{k(k+1)}{2} + 1 \right] = O(n(\log n)^2). \end{aligned}$$

We note that this procedure shows us how to compute the values of all of the elementary symmetric functions of $x_1, \dots, x_n$ in $O(n(\log n)^2)$ time. This procedure was first given by Horowitz in [5], p. 162.

Now to compute the n values $P(x_i)$, $1 \leq i \leq n$, we reverse direction and proceed to compute functions up the tree. If $D(x) = (x-x_1) \dots (x-x_n)$ then we can divide $P(x)$ by $D(x)$ obtaining a quotient and remainder as follows

$$P(x) = D(x)Q(x) + R_{11}(x),$$

where $\deg(R_{11}(x)) < \deg(D(x))$. By substitution it follows that

$$P(x_i) = R_{11}(x_i), \quad 1 \leq i \leq n$$

since $D(x_i) = 0$. We can continue by dividing $R_{11}(x)$ by $D_1(x) = (x-x_1) \dots (x-x_{\frac{n}{2}})$ and $D_2(x) = (x-x_{\frac{n}{2}+1}) \dots (x-x_n)$ obtaining

quotients and remainders as follows:

$$R_{11}(x) = D_1(x)Q_1(x) + R_{12}(x)$$

$$R_{11}(x) = D_2(x)Q_2(x) + R_{22}(x)$$

By the same argument we see that

$$P(x_i) = \begin{cases} R_{12}(x_i), & 1 \leq i \leq \frac{n}{2} \\ R_{22}(x_i), & \frac{n}{2}+1 \leq i \leq n \end{cases}$$

Eventually we will arrive at constants $R_{1k}, \dots, R_{2^{k-1},k}$

where $P(x_i) = R_{i,k}$, $1 \leq i \leq n$. Since the time for multiplication and division of polynomials is the same,

$$C(i) = M(2^{k-i}) = 2^{k-i}(k-i)$$

and the total time for problem 2 is

$$\begin{aligned} \sum_{1 \leq i \leq k} 2^{i-1} M(2^{k-i}) &= 2^{k-1} \sum_{1 \leq i \leq k} (k-i) \\ &= 2^{k-1} [k^2 - \frac{k(k+1)}{2}] = O(n(\log n)^2). \end{aligned}$$

The idea of using the binary tree and successively dividing $P(x)$ was given by Fiduccia in [3]. Moenck and Borodin combined the idea of Fiduccia with a fast division algorithm to produce a time less than $O(n^2)$. H. T. Kung in [8] gives a different method for n -point evaluation which has the same computing time bound but uses a different division algorithm. Borodin and Munro were the first to give a better than $O(n^2)$ method for n -point evaluation in [1].

Problem 3: Given n residues u_i of n single precision primes p_i to find the unique n -precision integer u such that

$$u \equiv u_i \pmod{p_i}, \quad 1 \leq i \leq n.$$

It follows from the Chinese Remainder Algorithm, see [7] p. 249, that this integer exists and is unique. For this problem as for problem 1 we will assume the binary tree in figure 2 has already been computed. Now what we need to do is go down the tree at each stage computing a new integer which is congruent to the previous two. For example, at the first level let $u_i = u_{i,k}$, $1 \leq i \leq n = 2^{k-1}$. Then for i odd we compute from $u_{i,k} \pmod{p_i}$ and $u_{i+1,k} \pmod{p_{i+1}}$ the unique integer $u_{i,k-1} \equiv u_{i,k} \pmod{p_i}$ and $u_{i,k-1} \equiv u_{i+1,k} \pmod{p_{i+1}}$. Thus $u_{i,k-1}$ lies in the range $[0, p_i p_{i+1})$. Repeating this process down the tree we will eventually produce u in the interval $[0, p_1 \dots p_n)$. So we only need an algorithm to proceed from level i to $i-1$ in the tree.

One-Step Chinese Remainder Algorithm

Input: u_1, u_2, q_1, q_2 , $\gcd(q_1, q_2) = 1$ $q_1 > q_2 > 0$.

Output: u_3 : $u_3 \equiv u_1 \pmod{q_1}$, $u_3 \equiv u_2 \pmod{q_2}$

- 1) $\bar{u}_1 \leftarrow u_1 \pmod{q_2}$
- 2) $\bar{c}_{12} \leftarrow q_1 \pmod{q_2}$
- 3) $c_{12} \leftarrow \bar{c}_{12}^{-1} \pmod{q_2}$
- 4) $u_3 \leftarrow u_1 + (u_2 - \bar{u}_1)c_{12}q_1$

Now Steps (1) and (2) take $M(\log q_1)$ and since $c_{12} < q_2$ Step (4) also takes no more than $M(\log q_2)$ operations.

In order to find c_{12} , which is the multiplicative reciprocal of $\bar{c}_{12}$ in the finite field with the q_2 elements $\{0, 1, \dots, q_2-1\}$, we need an extended gcd algorithm. That is, given q_2 and $\bar{c}_{12}$ it produces a, b and $\gcd(q_2, \bar{c}_{12})$ such that

$$aq_2 + b\bar{c}_{12} = \gcd(q_2, \bar{c}_{12}) = 1$$

Thus $b\bar{c}_{12} = 1 \pmod{q_2}$ and $b = \bar{c}_{12}^{-1}$. Now A. Schönhage in [10] has given such an algorithm whose time is

$$M(\log q_1)(\log \log q_1)$$

and this bounds the total time for the 1-step Chinese Remainder Algorithm.

We now apply the 1-step algorithm as we proceed down the tree. Then the total time for problem 3 is

$$\begin{aligned} & \sum_{1 \leq i \leq k-1} 2^{i-1} M(2^{k-i-1})(\log 2^{k-i-1}) \\ &= \sum_{1 \leq i \leq k-1} 2^{i-1} (2^{k-i-1}) (k-i-1)^2 \\ &= 2^{k-2} \sum_{1 \leq i \leq k} (k-i-1)^2 = O(n(\log n)^3) \end{aligned}$$

This result was first obtained by Heindel and Horowitz in [4], using the same tree construction as used here. Moenck and Borodin in [9] improve on this bound by a $(\log n)$ factor by making their computational model somewhat different than the one assumed so far. This will be discussed in Section 3.

Problem 4: Given n values $y_1, \dots, y_n$ at $n=2^{k-1}$ points $x_1, \dots, x_n$ we wish to compute the unique interpolation polynomial $P(x)$ of degree $\leq n-1$ such that $P(x_i) = y_i$, $1 \leq i \leq n$.

It follows from Lagrange's theorem that this polynomial exists and is unique, e.g. see [7], p. 428. For this problem as for problem 2 we will assume that the binary tree in figure 3 has already been computed. Again we need to go down the tree at each stage computing a new interpolating polynomial from its two ancestors. So for example at level k we compute polynomials $R_{1k}(x), \dots, R_{nk}(x)$ such that $R_{ik}(x_i) = y_i$. Then at level $k-1$ we compute $R_{1,k-1}, \dots, R_{\frac{n}{2},k-1}$ such that

$$\left. \begin{aligned} R_{i,k-1}(x_i) &= y_i \\ R_{i,k-1}(x_{i+1}) &= y_{i+1} \end{aligned} \right\} \text{ for } 1 \leq i \leq \frac{n}{2}$$

and so on until $R_{11}(x) = P(x)$ since $R_{11}(x_i) = y_i$ for $1 \leq i \leq n$. So first we need an algorithm which combines 2 interpolating polynomials to give a third which interpolates at both sets of points.

One-Step Interpolation Algorithm

Input: $U_1(x), U_2(x), Q_1(x), Q_2(x)$ $\gcd(Q_1, Q_2) = 1$ where
 $Q_1(x) = (x-x_1) \dots (x-x_{\frac{m}{2}})$, $Q_2(x) = (x-x_{\frac{m}{2}+1}) \dots (x-x_m)$

Output: $U_3(x)$ such that $U_3(x_i) = U_1(x_i)$ for $1 \leq i \leq \frac{m}{2}$
 $U_3(x_i) = U_2(x_i)$ for $\frac{m}{2}+1 \leq i \leq m$
 and $\deg(U_3) \leq m-1$.

- 1) $\bar{U}_1(x) \leftarrow U_1 \bmod Q_2$
- 2) $\bar{Q}_{12}(x) \leftarrow Q_1 \bmod Q_2$

$$3) C_{12}(x) \leftarrow \bar{C}_{12}^{-1} \bmod Q_2$$

$$4) U_3(x) \leftarrow U_1(x) + (U_2 - \bar{U}_1)C_{12}Q_1$$

We note that steps (1), (2) and (3) imply that there exist quotients C_1, C_2 and polynomial C_3 such that

$$\begin{aligned} U_1 &= Q_2 C_1 + \bar{U}_1, & \deg(\bar{U}_1) < \deg(Q_2) \\ Q_1 &= Q_2 C_2 + \bar{C}_{12}, & \deg(\bar{C}_{12}) < \deg(Q_2) \\ C_{12}(\bar{C}_{12}) + C_3 Q_2 &= 1, & \deg(C_{12}) < \deg(Q_2). \end{aligned}$$

C_{12} is the multiplicative inverse of $\bar{C}_{12}$ modulo Q_2 .

Therefore,

$$U_3 = U_1 + (U_2 + Q_2 C_1 - U_1) \left(\frac{1 - C_3 Q_2}{\bar{C}_{12}} \right) Q_1$$

and

$$U_3(x) \Big|_{x_1, \dots, x_{\frac{m}{2}}} = U_1(x) \Big|_{x_1, \dots, x_{\frac{m}{2}}}$$

since $Q_1(x)$ evaluated at $x = \{x_1, \dots, x_{\frac{m}{2}}\}$ is zero. Also

$$\begin{aligned} U_3(x) \Big|_{x_{\frac{m}{2}+1}, \dots, x_m} &= U_1 + (U_2 - U_1) \left(\frac{Q_1}{\bar{C}_{12}} \right) \\ &= U_1 + (U_2 - U_1) \\ &= U_2(x) \Big|_{x_{\frac{m}{2}+1}, \dots, x_m} \end{aligned}$$

So 1-step interpolation produces the correct results. Notice that the two one-step algorithms are essentially identical, the mod operation being applied either to integers or polynomials.

Now steps (1) and (2) take $M(m/2)$ time. In order to compute the multiplicative reciprocal of $\bar{C}_{12}$ we use the extended gcd algorithm of Schönhage, [10], which takes time $M(m/2)(\log m/2)$. The time for step(4) is no more than $M(m)$ so the total time for one-step interpolation is $O(M(m)(\log m))$.

Now applying this one-step algorithm as we proceed down the nodes of the tree in figure 3 we get that the total time for problem 4 is

$$\begin{aligned} & \sum_{1 \leq i \leq k-1} 2^{i-1} M(2^{k-i-1}) \log 2^{k-i-1} \\ &= \sum_{1 \leq i \leq k-1} 2^{i-1} 2^{k-i-1} (k-i-1)^2 = 2^{k-2} \sum_{1 \leq i \leq k-1} (k-i-1)^2 \\ &= O(n(\log n)^3). \end{aligned}$$

The first interpolation algorithm which took less than $O(n^2)$ operations was given by Horowitz in [5], but it used preconditioning of the x_i 's. Moenck and Borodin in [9] used the notion of fast division to aid in interpolation and were the first to show that the general problem could be done in less than $O(n^2)$ operations. Also their paper reveals the relationships between problems 1-4. Using a different approach for polynomial division and the same interpolation algorithm in [5], H. T. Kung in [8] also presents an interpolation algorithm. The idea of fast interpolation as given here was first presented by Hopcroft in [12].

3. Preconditioning

The notion of preconditioning has been successfully applied to the problem of evaluating a polynomial at a single point. Advanced knowledge of the coefficients allows one to reduce the number of multiplications by a factor of about 2. We have seen in the previous section that problems 1 and 2 take time $M(n)(\log n)$ and problems 3 and 4 take $M(n)(\log n)^2$. By appropriately preconditioning we can reduce the time for problems 3 and 4 to $M(n)(\log n)$.

The preconditioning that is needed comes directly from step (3) of the 1-step Chinese Remainder and the 1-step Interpolation algorithms. If the moduli $(p_1, \dots, p_n$ or $(x-x_1), \dots, (x-x_n))$ are known in advance then the multiplicative reciprocals of their various products can be precomputed. More specifically the products that must be formed consist of the two binary trees whose nodes are the multiplicative reciprocals of the elements of the binary trees in figures 2 and 3. This reduces the time for both 1-step algorithms by an appropriate log factor which similarly affects the time for problems 3 and 4.

We recall that the reason problems 3 and 4 required an extra $(\log n)$ factor over problems 1 and 2 was the use of Schönhage's extended gcd algorithm. The reason for using Schönhage's method is that it guarantees us the exact result with no loss of precision. But since we have not considered the accuracy question for other aspects of the

algorithms, perhaps we should consider loosening the constraint there.

If we do this, we can merely compute $c_{12} = 1/\bar{c}_{12}$ which for c_{12} an integer simply requires a reciprocal operation and for c_{12} a polynomial also requires the same operation with appropriate scaling of the exponents. Both division steps take $M(n)$ time and we would only keep the first n digits or coefficients for our answer. The proof that U_3 is correct carries through in essentially the same way. This model of computation was used both by Moenck and Borodin in [9] and by Kung in [8] who all obtain $M(n)\log n$ as the time for problems 3 and 4.

It is noted that a meta-algorithm could have been written for both 1-step algorithms. This meta-algorithm, using the preceeding ideas, would produce the correct output for the proper inputs. However this would only further abstract the process at no apparent gain in clarity, plus the parallel expositions serve to emphasize the similarities between the problems. An important point to note is that by using the extended gcd algorithm the inputs can most generally come from an arbitrary Euclidean domain. Since the integers form such a domain and this naturally extends to a Euclidean domain for multivariate polynomials over the integers, all operations within problems 1-4 are appropriately made. However, when we remove the extended gcd algorithm then problems 3 and 4 can no longer be accomplished over a Euclidean domain, but instead we must assume that the elements come from a field.

Acknowledgment: Credit is due to John Hopcroft for several enlightening discussions and to Robert Tarjan for asking me to prepare the original lectures upon which this paper is based.

References

- 1) Borodin, A. and I. Munro, "Evaluating Polynomials at Many Points", Information Processing Letters, Vol. 1, No. 2, July 1971, pp. 66-68.
- 2) Brown, W. S., "On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors", JACM, Vol. 18, No. 4, Oct. 1971, pp. 478-504.
- 3) Fiduccia, C., "Polynomial Evaluation via the Division Algorithm: the Fast Fourier Transform Revisited", Proceeding of the 4th Symposium on Theory of Computing, May 1972, pp. 88-93.
- 4) Heindel, L. and E. Horowitz, "On Decreasing the Computing Time for Modular Arithmetic", Proceedings of the 12th Symposium on Switching and Automata Theory, Oct. 1971, pp. 126-128.
- 5) Horowitz, E., "A Fast Method for Interpolation Using Preconditioning", Information Processing Letters 1, (1972), pp. 157-163.
- 6) Horowitz, E., "The Efficient Calculation of Powers of Polynomials", JCSS, Oct. 1973.
- 7) Knuth, D. E., The Art of Computer Programming - Seminumerical Algorithms, Vol. 2, 2nd edition, Addison-Wesley.
- 8) Kung, H. T., "Fast Evaluation and Interpolation", Computer Science Technical Report, Carnegie-Mellon University, Jan. 1973.
- 9) Moenck, R. and A. Borodin, "Fast Modular Transform Via Division", Proceedings of the 13th Symposium on Switching and Automata Theory, IEEE Computer Society, Oct. 1972, pp. 90-96.
- 10) Schönhage, A., "Fast Computation of Continued Fraction Expansions", English translation of "Schnelle Berechnung von Kettenbruchentwicklungen" (to appear)
- 11) Sieveking, M., "An Algorithm for Division of Power Series", Technical Report, University of Zurich.
- 12) Aho, A., J. Hopcroft and J. Ullman, Analysis of Algorithms, Book in Preparation.