DISCRETE COMPUTATION:
THEORY AND OPEN PROBLEMS

Notes for the lectures by

Albert R. Meyer

Preceptorial Introduction to Computer Science for Mathematicians
American Mathematical Society

San Francisco
January, 1974

This work was supported in part by the National Science Foundation under research grant GJ-34671.
LECTURE I

MULTIPLICATION IN BINARY

\[ U = 101100 \]
\[ V = 111111 \]
\[ 101100 \]
\[ 101100 \]

\[ U \times V = 101011010100 \]

\[ \alpha = \text{the add and shift multiplication algorithm} \]

\[ T_{\alpha}(U,V) = \text{time (number of basic operations on digits) to multiply } U \text{ and } V \text{ by method } \alpha. \]

\[ T_{\alpha}(n) = \max \{ T_{\alpha}(U,V) \mid \ell(U) = \ell(V) = n \} \]

Remark: \[ T_{\alpha}(n) = O(n^2) \]
Recursive algorithm for multiplication

\[ U = \begin{bmatrix} u_1 & u_2 \end{bmatrix} = u_1 \cdot 2^{n/2} + u_2 \]

\[ V = \begin{bmatrix} v_1 & v_2 \end{bmatrix} = v_1 \cdot 2^{n/2} + v_2 \]

\[ U \cdot V = u_1 v_1 \cdot 2^n + (u_1 v_2 + u_2 v_1) \cdot 2^{n/2} + u_2 v_2 \]

---

\[ \rho = \text{recursive algorithm} \]

\[ T_{\rho}(n) = 4T_{\rho}(n/2) + (\text{time to add and shift length } n \text{ numbers}) \]

\[ = 4T_{\rho}(n/2) + O(n) \]

\[ = 4 \cdot 4^{(n/2)} + O(n) \]

\[ = 0(4 \log_2 n) = o(n^2) \]
\[ \beta = \text{better recursive algorithm using only three half length multiplications} \]

1. \((u_1 + u_2) \cdot (v_1 + v_2)\)
2. \(u_1 v_1\)
3. \(u_2 v_2\)

\[ u \cdot v = 2 \cdot 2^n + (1 - 2 - 3) \cdot 2^{n/2} + 3 \]

\[ T_\beta(n) = 3T_\beta(n/2) + O(n) \]

\[ = O(3 \log_2 n) \]

\[ = O(n \log_2 3) \]

\[ \approx O(n^{1.6}) \]
Best upper bound known for multiplication:

\( O(n \cdot \log n \cdot \log \log n) \)

by Strassen and Schönhage.

**Question:** What is the fastest possible way to multiply?

Need there even be one?

---

Might have algorithms \( \beta_1, \beta_2, \ldots \):

\[ \beta_1(n) = n \cdot \log n \]

\[ \beta_2(n) = n \cdot \sqrt{\log n} \]

\[ \beta_3(n) = n \cdot (\log n)^{\frac{1}{4}} \]

Then there is no fastest one.
Turing Machine

FLOWCHARTS FOR TM'S

\[
\begin{align*}
\text{start} & \quad \Downarrow \\
\text{MOVE RIGHT} & \quad \Downarrow \\
\text{WRITE} \sigma & \quad \Downarrow \\
\text{READ} \sigma_1 \sigma_2 \ldots \sigma_k & \quad \Downarrow \\
\text{HALT} & \\
\end{align*}
\]

\[\sigma \in \Sigma = \{ \text{tape symbols} \}\]
FLOWCHART FOR F(x) = 2x

(Input x: an integer in binary notation.)
\[ T_\mathcal{M}(x) = \text{number of instructions executed by } \mathcal{M} \text{ on } x \text{ if } \mathcal{M} \text{ halts}; \infty \text{ if } \mathcal{M} \text{ doesn't halt on } x. \]

\[ S_\mathcal{M}(x) = \text{number of tape squares visited by head of } \mathcal{M} \text{ with input } x \text{ if } \mathcal{M} \text{ halts}; \infty \text{ if } \mathcal{M} \text{ does not halt.} \]

\[ \varphi_\mathcal{M}(x) = \text{output of } \mathcal{M} \text{ on } x, \text{ if any; } \infty \text{ if no output.} \]

\[ T = \text{time} \quad S = \text{space} \quad \varphi = \text{function} \]

Church's Thesis:

The effectively (mechanically) computable functions and the Turing machine computable functions are the same.

Extended Church's Thesis:

If a function is computable in time \( T \) on any reasonable computer model, then it is computable in time \( \leq \text{polynomial}(T) \) on a Turing machine.
Infinitely-often Speed-up Theorem:

(M. BLUM). Let $t: \mathbb{N} \to \mathbb{N}$ be any computable function. Then there is a computable function $C_t: \mathbb{N} \to \{0,1\}$ such that

given any $\mathbb{M}$ computing $C_t$ one can construct an $\mathbb{M'}$ also computing $C_t$
with the property that

$$T_{\mathbb{M}'}(x) > t(x) \text{ and } T_{\mathbb{M}''}(x) < \text{constant}$$

for infinitely many $x \in \mathbb{N}$.

---

numbers of steps

$\mathbb{M}'$ is faster than $\mathbb{M}$ infinitely often,

$\mathbb{M}''$ is faster than $\mathbb{M}'$ infinitely often, etc.
Let \( M_0, M_1, \ldots, M_i \) be an orderly list of
of all Turing machines (say in order of the
size of their flowcharts).

Let \( \varphi_i \) abbreviate \( \varphi_{M_i} \),

\[
\begin{array}{c}
\varphi_i \\
M_i \\
\end{array}
\]

**Universal Machine Theorem:**

\( \varphi_i(x) \), regarded as a function of both

i and x, is computable.

---------------------------------------------

**Padding Lemma:**

Given any program, one can "pad" it with
instructions which it never uses. Thus,
we obtain

a new program with the same behavior
as the old one.

More formally,
LEMMA: For any T.M. \( T_e \) there is an infinite set \( \mathcal{E}(e) \subseteq \mathbb{N} \) such that

(1) for any \( e' \in \mathcal{E}(e) \)

\[ \varphi_e = \varphi_{e'}, T_e = T_{e'}, \text{ and } S_e = S_{e'} \]

and (2) there is a T.M. which recognizes elements of \( \mathcal{E}(e) \) in constant time.

(Think of \( \mathcal{E}(e) \) being binary numbers of the form:

| & e & | f | irrelevant |
|---|---|---|---|

Proof of L.O. Speed-up Theorem:

Let

\[ c_e(x) \overset{df}{=} \begin{cases} 1 & \quad \text{if } T_e(x) \leq t(x), \\ 0 & \quad \text{otherwise.} \end{cases} \]

\[ l = y \overset{df}{=} \begin{cases} 0 & \quad \text{if } y \geq 1, \\ 1 & \quad \text{if } y = 0. \end{cases} \]
(1) \( C_t \) is computable (implicit in the Universal Machine Thm.)

(2) If \( \varphi_{e'} = C_t \), then \( T_{e'}(e') > t(e') \)

and \( C_t(e') = 0 \) (by def. of \( C_t \)).

(3) Say \( \varphi_e = C_t \). Then for any \( e' \in \text{PAD}(e) \),

\[ t(e') < T_{e'}(e') = T_{e'}(a') \]

and \( C_t(e') = 0 \).

So speed-up \( \mathcal{R} \) by always testing if the input is in \( \text{PAD}(e) \), and if so immediately print output 0.
Def. \( \text{Time}(t) = \{ \phi : \mathbb{N} \to \mathbb{N} \mid Y_x(t) \leq t(x) \text{ almost everywhere} \} \)

\[ \text{Space}(t) = \cdots S_i \cdots \]

Supplementary. \( C \notin \text{Time}(t) \) for any computable \( t \).

Remark: Time to compute \( C \) depends on time to compute \( t \).

Convention: \( n = \text{length}(x) = f(x) = \log_2 x \). Thus,

\( \text{Time}(2^n) = \text{Time}(2^f(x)) = \text{Time}(x) \),

\( \text{Time}(2^{2n}) = \text{Time}(x^2) \), etc.
Def. A computable \( t: \mathbb{N} \to \mathbb{N} \) is time-honest iff \( t \in \text{Time}(t^3) \) and \( t(x) \geq L(x) \).

Cor. (Compression Theorem, Hartmanis-Stearns)

For any time-honest \( t \), \( C_t \in \text{Time}(t^4) - \text{Time}(t) \).

Remark: Lots of time-honest \( f \)'s.

\[
\left\{ \log_2 x \right\}, \ x, \ 2^x, \ 2^{2^x}, \ldots
\]

closed under +, \(-\), \(\times\), \(\exp\), composition.
Is more time better than less?

Is

\[ \text{Time}(t^4) - \text{Time}(t) \neq \varnothing \]

for all computable \( t \)?

\text{NOT NECESSARILY!}

---

**Gap Theorem:** (Trakhtenbrot, Borellem) For any computable \( g \), there exist arbitrarily large computable \( t \) such that

\[ \text{Time}(t) = \text{Time}(g(t)) \]

---

**Proof of Gap Theorem:**

Given \( g \), define

\[ t(x) = \text{the least } z \text{ such that} \]

\[ \text{Time}(\{ T^4_1(x) < z \text{ or } T^4_1(x) > g(z) \}). \]
Honesty Theorem (McCreight, Meyer)

For every computable \( t \), there is a time-honest \( t' \) such that

\[
\text{Time}(t) = \text{Time}(t')
\]

Summary:

For arbitrarily large \( t, t' \), it can happen that

\[
\text{Time}(2^t) \neq \text{Time}(t) \neq \text{Time}(t') \neq \text{Time}(t')^6
\]

GAP  HONESTY  COMPRESSION
\[ \text{Lines}(f) \text{ is radial lines almost everywhere under } f \]

**Compression:** For any line \( L \neq 0 \),
\[ \text{Lines}(2L) \supset \text{Lines}(L) \]

**Homotopy:** For any function \( f \), there is a line \( L \),
\[ \text{Lines}(f) = \text{Lines}(L). \]

**Gap:** \[ \text{Lines}(t) = \text{Lines}(2^t) = \{ \text{zero line} \} \]
for \( t = \log\log \).
Def. Let \( f \) be a computable function. A sequence \( t_1, t_2, \ldots \) of functions is a complexity sequence for \( f \) iff

1. If \( \varphi_e = f \), then \( S_e \leq t_i \) almost everywhere for some \( i \),

and
2. For every \( i \), there is a \( \varphi_e = f \) such that \( t_i \geq S_e \) almost everywhere.

Def. A sequence of functions \( p_1, p_2, \ldots \) is an r.e. complexity sequence (for \( f \)) iff

1. \( p_{i+1} \leq \left\lceil \frac{1}{2} \cdot p_i \right\rceil \) for all \( i \),

and
2. For each \( i \) there is a \( j \) such that

\[ p_i = S_j \]

and
3. \( p_i(x) \) is a computable function of \( i \) and \( x \).
Theorem (Meyer, Schnorr) Every computable function has an r.e. complexity sequence.

Every r.e. complexity sequence is a complexity sequence for some 0-1 valued computable function.

Example:

\[
\text{Let } t_i(x) = \left\{ \begin{array}{ll}
2^0 & \text{if } x = i \\
2^1 & \\
\vdots & \\
2^n & \text{if } x = i
\end{array} \right. 
\]

So \( t_{i+1} = \log_2 t_i \) almost everywhere.

Cor. Almost everywhere Speed-up

(Blum) There is a 0-1 valued computable function, \( c \), such that for any T.M. computing \( c \) there is another T.M. computing \( c \) which uses exponentially less space at almost all arguments.
LECTURE II

Σ = finite set called the alphabet or vocabulary,
an element σ ∈ Σ is called a letter.

Σ* = set of all finite sequence of letters,
an element x ∈ Σ* is called a word.

Binary operation concatenation, written " • " on Σ*:

    x • y = xy = word x followed by word y.

Example:  001 • 01 = 00101

l(x) = length (number of occurrences of letter)
of the word x.

l(001) = 3

l(x • y) = l(x) + l(y).
\[ \lambda \in \Sigma^* \text{ acts as an identity element under concatenation.} \]
\[ \lambda \cdot x = x \cdot \lambda = x \quad \text{for all } x \in \Sigma^*, \]
\[ I(\lambda) = 0. \]

Remark 1: \( <\Sigma^*, \cdot> \) is the free monoid generated by \( \Sigma \) with identity \( \lambda \).

Remark 2: Remark 1 is irrelevant.

Remark 3: \( \lambda \) is introduced as a technical convenience and could be eliminated in what follows at the expense of some minor awkwardness.

---

A set \( L \subset \Sigma^* \) is called a language. Extending concatenation to languages in the usual way:

\[ L \cdot M, \text{ also written } LM = \text{def.} \]
\[ \{ x \cdot y \mid x \in L \text{ and } y \in M \} \]

Example: \( \{0\} \cdot \{0,1\} = \{00,01\} \)
\[ \{0,00\} \cdot \{1,01\} = \{01,001,0001\} \]
\[ \{0,1,\lambda\} \cdot \{0,1,\lambda\} \cdot \{0,1,\lambda\} = \text{all binary words of length } \leq 3 \text{ (including } \lambda). \]
For $x \in \Sigma^*$, $n \in \mathbb{N}$,

$$x^n = x \cdot x \cdot \ldots \cdot x$$

$$0^0 = \lambda$$

**Example:** $(01)^3 = 010101$

Similarly for $A \subseteq \Sigma^*$

$$A^n = A \cdot A \cdot \ldots \cdot A$$

$$0^A = (\lambda)$$

**Example:**

$$(0,1)^4 = \text{all binary words of length exactly 4}$$

$$(0,1,\lambda)^4 = \text{all binary words of length } 4$$.

$$(((0,1)^2)^2)^2 = (0,1)^4 = \text{all binary words of length 16}.$$
Important example:

\[(01)^n = 0101\ldots01 = \]
\[\overbrace{2^n} \]

\[= [0,1]^{2n} \cup (1\cdot[0,1,\lambda])^{2n} \cup [0,1,\lambda]^{2n}\cdot0 \cup [0,1,\lambda]^{2n}\cdot(00,11)\cdot[0,1,\lambda]^{2n}\]

= all binary words of length 2n which do not

(1) start wrong
or (2) end wrong
or (3) move wrong (contain a forbidden local pattern)

Problem: Given two expressions involving letters
in \(\Sigma,\lambda,\) and operations

"." concatenation
"\cup" union
"2" squaring
"\cap" intersection
"-" set difference

is there a way to tell if they describe the same language?
YES!

BUT NO GOOD WAY!!

Lemma. An expression containing \( n \) operation symbols describes a subset of \((\Sigma \cup \lambda)^{2^n}\).

Proof. By induction on \( n \):

If \( n = 0 \), the expression must consist of a single letter or \( \lambda \).

If \( E \) is an expression containing \( n+1 \) operations, then \( E \) is of the form

\[
E_1 \ast E_2
\]

\[
E_1 \cup E_2
\]

\[
(E_1 \cup E_2)^2
\]

\[
E_1 \cap E_2
\]

\[
E_1 \setminus E_2
\]

where \( E_1, E_2 \) are expressions containing \( \leq n \) operation symbols. Proof follows immediately.
For any expression $E$, let

$$\mathcal{L}(E) \subseteq \Sigma^*$$

be the language described by $E$.

**Remark:** Formally, $E_1 = E_2$ means that $E_1$ and $E_2$ are identical expressions. $E_1$ and $E_2$ are **equivalent** (written $E_1 \equiv E_2$) iff

$$\mathcal{L}(E_1) = \mathcal{L}(E_2).$$

\[ \begin{align*}
E_1 \equiv E_2 \iff & (E_1 - E_2) \cup (E_2 - E_1) = \emptyset \\
& \text{Hence sufficient to test whether an expression describes the empty set.}
\end{align*} \]

\[ \begin{align*}
\text{To test if } \mathcal{L}(E) = \emptyset, \text{ convert } E \text{ to a list of the words in } \mathcal{L}(E) \text{ beginning at the "innermost" subexpressions of } E \text{ and working out.} \\
& \text{See if the list is empty when you finish.}
\end{align*} \]

**Difficulty:** The list for

$$\left( \cdots \left( \left( (0 \cup 1)^2 \right)^2 \right)^2 \right)_{n}$$

contains
Theorem 1. There is (for any finite \( \Sigma \)) a constant \( k > 0 \) and a Turing machine \( \mathcal{M} \) such that

\[
\mathcal{M} = (\Sigma, \phi, k, 2^k, \mathcal{N})
\]

(1) \( \mathcal{M} \) accepts an input \( w \) iff \( w \) is a well-formed expression and \( \mathcal{L}(w) = \phi \).

(2) \( T_{\mathcal{M}}(n) \overset{\text{df}}{=} \max \{ T_{\mathcal{M}}(x) \mid \mathcal{L}(x) = \mathcal{N} \} \leq 2^{kn} 
\]

Theorem 2. There is a finite \( \Sigma \) and a constant \( k > 1 \) such that

if \( \mathcal{M} \) is any T.M. accepting precisely the expressions over \( \Sigma \) describing the empty set, then

\[ T_{\mathcal{M}}(n) \geq \frac{k^n}{n} \]

for infinitely many \( n \). (This is an improvement.)

(That is, \( \{ \mathcal{E} \mid \mathcal{L}(\mathcal{E}) = \phi \} \subseteq \text{Time}(2^{kn}) \) for infinitely many \( n \).

\[ k > 1, \quad 2^{kn} \quad \text{for infinitely many } n \]
To prove Theorem 2:

(i) Define a relation on languages

\[ L_1 < L_2 \]

with intuitive meaning that \( L_1 \) is easy to decide given \( L_2 \).

(ii) Show that for any \( L \in \mathrm{Time}(2^n) \)

\[ L < \{ E \mid L(E) = \emptyset \}. \]

(iii) Deduce from the Compression Theorem that there is an \( L \in \mathrm{Time}(2^n) \) which is hard to decide.

(iv) Conclude that \( \{ E \mid L(E) = \emptyset \} \) is hard to decide.

---

**Def.** For \( L_1 \subseteq \Sigma_1^* \), \( L_2 \subseteq \Sigma_2^* \) we say \( L_1 < L_2 \)

\( (L_1 \text{ is polynomial time reducible to } L_2) \) iff

there exists a function \( f: \Sigma_1^* \to \Sigma_2^* \)

(1) \( f \) is computable in time bounded by a polynomial in the length of its argument

\( (f \in \mathrm{Time}(p \circ \ell )) \) where \( p: N \to N \) is a polynomial and \( \ell : \Sigma_1^* \to N \) is the length function,

(2) \( x \in L_1 \iff f(x) \in L_2 \) for all \( x \in \Sigma_1^* \).
Lemma. Let $t: \mathbb{N} \rightarrow \mathbb{N}$ be nondecreasing, and $t(n) \geq n$.

If $L_1 < L_2$ and $L_2 \in \text{Time}(t(n))$, then $L_1 \in \text{Time}(t(p(n)))$ for some polynomial $p: \mathbb{N} \rightarrow \mathbb{N}$.

Contrapositive. If $L_1 \notin \text{Time}(2^{\frac{n}{4}})$ and $L_1 < L_2$, then $\exists k > 1$ such that $L_2 \notin \text{Time}(2^{\frac{k}{n}})$.

Because $2^{L(x)/4}$ is time-honest, the compression theorem implies $\exists L_1$ such that

$L_1 \in \text{Time}(2^n) - \text{Time}(2^{\frac{n}{4}})$.

Thm. 2 follows immediately from the preceding contrapositive if we show.

Main Construction for Theorem 2.

Lemma. For any $L \in \text{Time}(2^n)$, there is an alphabet $\Sigma$ such that

$L < \{ E \mid E \text{ is an expression over } \Sigma \text{ and } \mathcal{L}(E) = \emptyset \}$. 
Choose one of the above rows

Say $I = 1$. The next alphabet is

(2) Write on any input of length $n$ in

The fourth brother and third in symbol $1$ on the
tape must appear as symbol $1$. On the
Let $Q$ be the states (boxes in the flowchart) of $M$.

Let $W$ be the tape symbols of $M$ including $b \in W$ for the blank tape symbol, let $\#$ be still another symbol.

$\Sigma \overset{\text{def.}}{=} Q \cup W \cup \{\#\}$.

For $x \in \Delta^*$, $l(x)=n$,

Comp($x$) $\in \Sigma^*$ is to be:

$\# b^{2^n} \cdot \text{start} \cdot x b^{2^n} \# ($tape after one step$) \# \ldots$

$\ldots \#$ (tape after $k$ steps) $\#$ (tape after $k+1$ steps) $\# \ldots$

$\ldots \#$ tape $\overset{\text{halt}}{\text{symbols}} \#$

Exactly $2 \cdot 2^n + n+1$ symbols between successive $\#$'s.

$l(\text{Comp}(x)) \leq 2^{3(n+1)} \overset{\text{df.}}{=} N$

--------------------------------------------------------
Comp(x) has the property that any four consecutive letters determine the letter \(2 \cdot 2^n + n\) to their right:

\[
\begin{array}{c|c|c}
\cdots & \cdots & \cdots \\
\end{array}
\]

\(2 \cdot 2^n + n + 1\)

Let \(F = \{(\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5) | \sigma_5 \text{ is not the letter determined by } \sigma_1 \sigma_2 \sigma_3 \sigma_4\} \). This follows from the fact that at any step the next move of \(M\) is determined by the state and the tape symbol being scanned.

\[
\text{Comp}(x) = (\text{starts right}) \cap \\
(\text{ends right}) \cap \\
((\Sigma \cup \lambda)^N - (\text{moves wrong}))
\]

\begin{align*}
\text{starts right:} & \quad \# b^n \cdot \underline{\text{start}} \cdot x \cdot b^n \cdot \# \cdot (\Sigma \cup \lambda)^N \\
\text{ends right:} & \quad (\Sigma \cup \lambda)^N \cdot \underline{\text{halt}} \cdot (\Sigma \cup \lambda)^N \cdot \#
\end{align*}

\[
\text{moves wrong:} \\
(\Sigma \cup \lambda)^N \cdot (\bigcup_{F \in F} (\Sigma_1 \sigma_2 \sigma_3 \sigma_4, \Sigma^{2 \cdot 2^n + n - 1} \cdot \sigma_5) \cdot (\Sigma \cup \lambda)^N)
\]
Let \( \text{Rejects}(x) = \) 
\[ (\Sigma \cup \lambda)^N \cdot \text{halt} \cdot (\Sigma -(1)) \cdot (\Sigma \cup \lambda)^N. \]

Then 
\[ x \in L \Leftrightarrow \text{TM halts reading a 1} \]
\[ \Rightarrow \text{Comp}(x) \cap \text{Rejects}(x) = \emptyset. \]

But expressions for \( \text{Comp}(x) \) and \( \text{Rejects}(x) \)
can be constructed in polynomial time in 
\( L(x) \), so 
\[ L \prec \{ E \text{ over } \Sigma \mid L(E) = \emptyset \}. \]
Q.E.D.

-----------------------------

Remarks: (1) Thm. 2 holds for expressions
using only "\( \cdot \)" , "\( \cup \)" , "2" and letters 0,1.

(2) If we allow "\( [0,1]^* \)" to be used in
expressions Stockmeyer has shown that

\[ \{ E \text{ with } [0,1]^* \mid L(E) = \emptyset \} \in \text{Time } 2^{2^n}. \]

but \( \notin \text{Time } 2^{2^n \cdot \log_2 n} \)
for some fixed \( \varepsilon > 0. \)

(3) If we allow only "\( \cup \)" , "\( \cdot \)" , the
equivalence problem is complete in \( \text{NP} \)
(discussed in Karp's lecture).

-----------------------------
Remark: Most decidable theories studied in mathematical logic require exponential time or worse. (An important exception being the propositional calculus, for which lower bounds larger than a polynomial are unknown.)

Open problems:

1. Can the satisfiable formulas of the propositional calculus be recognized in polynomial time? (This is the $P = \mathcal{NP}$ question of Cook and Karp).

2. Can a multi-tape Turing machine multiply integers (in binary notation) in linear time?

3. What is the relation between time and space?

   Known: $S_M(n) \leq T_M(n) \leq c S_M(n)$

   ($c > 1$ depends on $M$)

   Open: If $L \in \text{Time}(2^n)$ is $L \in \text{Space}(n)$?

4. Is $\text{Space}(n) = \text{Nondeterministic Space}(n)$? (The LBA problem of Myhill)
(5) Are linear time 3 tape T.M.'s more powerful than linear time 2-tape T.M.'s?

(6) Can the primes (represented in binary) be recognized in linear time?
Can the context-free languages?

(7) Can two n x n matrices be multiplied in proportional to \( n^{2.8} \) arithmetic operations?
\( n^{2.9} \) is known to be possible.)
References:

Abstract Complexity


Fast arithmetic

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12. Sponsoring Organization Name and Address</td>
<td>Associate Program Director Office of Computing Activities National Science Foundation Washington, D. C. 20550</td>
<td>11. Contract/Grant No.</td>
<td>GJ34671</td>
<td></td>
</tr>
<tr>
<td>17a. Descriptors</td>
<td>17. Key Words and Document Analysis.</td>
<td>17b. Identifiers/Open-Ended Terms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18. Availability Statement</td>
<td>Unlimited Distribution</td>
<td>Write Project MAC Publications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19. Security Class (This Report)</td>
<td>UNCLASSIFIED</td>
<td>20. Security Class (This Page)</td>
<td>UNCLASSIFIED</td>
<td></td>
</tr>
<tr>
<td>21. No. of Pages</td>
<td>36</td>
<td>22. Price</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DISCRETE COMPUTATION:
THEORY AND OPEN PROBLEMS

Albert R. Meyer

January 1974