From 1966 PhD Thesis "On the Minimum Computation Time of Functions" by Stephen A. Cook

Harvard University

Dept. of M[illegible]

CHAPTER III

POSITIVE RESULTS

16. The Toom aborithm for multiplication.

A. L. Too,n [14] has shown that for: every n, there is a logic net of no more than $c_1 n c_2^{\sqrt{\log n}}$ components which realizes the multiplication of two arbitrary n-digit binary numbers, for some constants c_1, c_2. Implicit in his construction is an algorit!m for carrying out multiplication that is considerably more efficient than the standard multiplication algorithm, provided the numbers of digits in the factors are sufficiently large. In fact, the main purpose of this chapter is to show that the Toom algorithm can be utilized by a multitape Turing machil-e to multiply any two n-digit numbers within $n\,2^{5\sqrt{\log n}}$ steps, for all n. We shall also show that the method can be extended to cover other computations besides multiplication, and derive some consequences concerning the relative difficulty of computing the radix e.'tpansions of real numbers in different radix bases (section 21). One result (20.1) concerning the rate at which a Turing machine can compute the radix expansions of algebraic numbers improves on a theorem of llartmanis and Stearns [6].

Toom's method (slightly modified) can be presented *as* follows. Suppose M and N ar:e the integers to be multiplied, and suppose the base b expansions of !1 and N are $\mu_n\mu_{n-1}\cdots\mu_0$ and $v_n v_{n-1}\cdots v_0$, respectively. Thus

$$M = \sum_{k=0}^{n} \mu_k b^k, \quad \text{and}$$

16.1.

$$N = \sum_{k=0}^{n} \nu_k b^k.$$

Choose integers q, r such that $n + 1 = q(r + 1)$ and divide the expansions of M and N into $r + 1$ segments of q digits each. Thus, if α_k is the number represented by the k^{th} segment of M and β_k is the number represented by the k^{th} segment of N, then

16.2.

$$\left.\begin{aligned} \alpha_k &= \sum_{i=0}^{q-1} \mu_{i+kq} b^i \\ \beta_k &= \sum_{i=0}^{q-1} \nu_{i+kq} b^i \end{aligned}\right\} \quad k = 0, 1, \ldots, r,$$

where we define $\mu_k = \nu_k = 0$ for $k > n$. Next define the two polynomials $P(x)$ and $Q(x)$ by

16.3.

$$P(x) = \sum_{k=0}^{r} \alpha_k x^k, \quad \text{and}$$

$$Q(x) = \sum_{k=0}^{r} \beta_k x^k,$$

so that $M = P(b^q)$ and $N = Q(b^q)$. To find the product MN it is sufficient to find the coefficients of the polynomial $P(x)Q(x)$ and evaluate the polynomial at $x = b^q$. Find the coefficients by first calculating the values

16.4.
$$\left.\begin{array}{l} m_k = P(k) \\ n_k = Q(k) \end{array}\right\} \quad k = 0,1,\ldots,2r,$$

then finding the product $m_k n_k$ by applying the general method all over again, and finally solving for the γ_k's the system of equations

16.5.
$$\gamma_0 + \gamma_1 k + \gamma_2 k^2 + \ldots + \gamma_{2r} k^{2r} = m_k n_k, \quad k = 0,1,\ldots,2r.$$

A near-optimal relationship between q and r is given by $r = 2^{[\sqrt{\log q}]}$, where $[x]$ is the greatest integer $\leq x$.

Although Toom designed his logic net to multiply integers in radix notation it turns out the same method works for multiplying polynomials over the integers modulo 2; or more generally, polynomials over Z_b where b is prime, or polynomials over any finite field. A multitape Turing machine can utilize the algorithm to multiply such polynomials (i.e. compute the function f defined in 5.7) at the same rapid rate as for ordinary multiplication: $T(n) \leq n\, 2^{5\sqrt{\log n}}$.

It seems especially interesting that this type of multiplication can be performed so rapidly, since it is so easy to visualize the dependence of the output on the input (say for the case $b = 2$). Each successive output digit is a sum of products of the proceeding input symbols, and if each output calculation were independent of the others, the computing time would grow as n^2. It is far from obvious that some of the work used in generating early outputs can be used for later outputs as well, but in fact the method of Tb0111 shows a great time savings can be affected by working on producing long se$_q$uences of outputs sUIUl.taneously.

To adapt the Toom algorithm to polynomial multiplication, treat H • M(\I} **and** N • N(W} as polynomi.118 **ln** the **indeterainant V** with coefficients µ0, ••• ,un and v0, ••• ,vn from Zb. Thus equations 16.1 hold when b i s replaced by W. The ak's and Sk's become polynOlrlals in II of degree q - 1 or **less** so that 16.2 holds when b i • replaced **by** W and 16.3 holds as it is. Thus **P(x)** • P(W,x) and Q(x) • Q(W,x) are polynomials in the two indeter111inants II and x such that M • P(W,W^q) and N • Q(W,11^q). In equations 16.4, k must be replaced by k*, where k* ie the polyn.Ol'llial in II whose string of coefficients is the base b notation for the integer **k.** Thus 0*,1*,···•<2r)* are 2r + 1 distinct polynomi.als none of whose degrees exceeds [1o¾ (2r + 1) I. Then the system of e$_q$uations 16.5, with k replaced by k*, can be solved for the yk's as before, **as we** shall see in section 18. This is tba step in the algorithm that requires b to be prime. since if b i s not prime, then the polynO!!li.als over Zb do not fora an integral domain, and the ayetC!II 16.5 may not have a unique solution.

It is interesting that the time estimates derived here for carrying out the various parts of the algorithm for ordinary multiplication turn out to be essentially the same as the estimates made by Toom on the amount of e$_q$uipment required by the logic net. A little thought shows that this cannot be an instance of a general pl,enomenon, since logic nets c:in be designed to coopute non-recursive functions; somethiDg Turing machines cannot do at any speed. In fact, the estimates for the logic net are very easily verified, but ve shall spend some effort in showing that the Turing machine is able to carry out the necessary bookkeeping and set-up operations fast enou h so that the corresponding time estilllates apply. The main difficulty comes in solving the system of linear equations: the inverse of the matrix of the system c'ID be built into the logic net, but not into the finite state control of the Turing aachine, since q and r can be arbitrarily large. Renee the Turing machine 11ust co.apute the inverse.

The problem, in other words, is to show the method is uniform in n, the number of digits in the fac:tors. It might be of some interest to isolate and study the class of fWJctions which have the property that the !rlnilllum amount of equipment needed to realize them by a logic net grows at the same rate as the computation time of the fastest Turing aachi.oe which computes them. Of course we have no proof that multiplication is such a "uniform" function, but it does seem plausible.

17. Turing 11acbine which incornorates the a1godthm.

In this section we give a very general description of a multitape Turin machine vhich realizes the Toom algorlthm, and then ve estimate the computing time of the machine. The estimate depends on an inequality (17.6) whose proof is postponed until section 19, where ve give a more detailed descrlption of the aacbine.

17.1. Theorem: For every base b, there is a multitape Turing machine which, given an arbitrary pair of n-digit input integers in base b notation, will compute their product in base b notation within $n\ 2^{5\sqrt{\log n}}$ steps, for all n. The same applies to polynomial multiplication over Zb (cf. 5.7), provided b is prime.

The input-output arrangements of the Turing machine are those discussed in section 5 (cf. 5.6 and 5.7). In sum, the digits of the factors are written on a special read-only input tape, a pair of digits on each square of the tape. The machine advances the tape at irregular intervals to read the information on this tape. The actual computation is carried out on several ordinary linear tapes, which have one read-write head per tape. The output digits are given by an output function A from the displays of the machine to the set of digits $\{0,1, \cdots ,b - 1\}$. Although 6.1 could be taken as the definition of a multitape Turing machine, it will be convenient to assume that the multiplying machine has a finite set of internal states (which are not provided for in definition 6.1). Thus we can think of the machine as having special output states which designate the product digits. There is no on-line restriction; that is the machine reads input digits faster than it designates output digits.

The machL e described 1n section 19 has many tapes, but it is possible to take advantage of a result of Bennie and Stearns [9] to show the nwnber of tapes can be reduced to two with only slight loss in time. Hennie and Stearns show that any multitape Turing machine can be si. ulated by a two-tape Turing machine at the tate of n log n steps on the t o-tape machine for n steps on the multitape 111achine. Th.a slo'Wer rate of the t'-i'o-tape machine boosts thr. computation time from $n\ 2^{5\sqrt{\log n}}$ to $n\ 2^{(5\text{-M:})\sqrt{\log n}}$, but 'E can be taken to be an arbitrarily small positiv e nwnber by applying the speed-up theorem of Rartmanis and Stearns [6].

The :aultiplying machine, whcthar it :nulti?lies polyno111ials or i tegcrs, utilizes the algorithm described in t o previous section. Thu discussion belo11 assumes integel: multi?lication, but it applies, with only a fzw modifications, to polynomial multiplication. It is convenient to think of the 111achine as having a program which is orga:tlzed into a main routine and a subroutine. T e main routine handles the input and output, and initialize& the parameters, while the subroutine does most f the computation.

The 'llachine does not operate on the entire input string at onc"', but rather the computation proceeds in stages. During the first stage, th a,_;.inroutine sets the paramete1:s r and q equal to the initial values r1 and q1 (described in section 19) and reads in q1(r1 + 1) pairs of input digits, thus determining the initial values for the factor& Mand N. The main routine then makes Mand N available to the subroutine, which calculates the product ;IN and returns control to the main routine. The main routine designates the product as output, updates th.? values of q and r, and the process begins all over again. In general, the values of q and 1: are selected from the &equences q1,q2, .•• and r1,r2, ••• , which the main routine computes using the recursion equations

$$q_{i+1} = r_i q_i$$

17.2.

$$r_{i+1} = 2^{[\sqrt{\log q_{i+1}}]},$$

where [1<) means the greatest integer in x. On the i^{th} sta e, the factors M and N consist of the first (i.e. lowest order) qi(ri + 1) pairs of input digits. In order for the subroutine to calculate the product MN, it must first calculate the lesser products of the form mknk appearing on the right side of equation 16.5, and hence it must be able to call (i.e. re-enter) itoelf with n w input parameters, while not destroying the old ones. We shall prove in lemma 17.3 below that if q1 and r1 are properly chosen, then the numbers mk and nk defined in 16.4 have at most qi + qi-1 • qi-1 (ri-1 + 1) digits each, so that the subroutine is able to set the parameters q and r equal to qi-1 and ri-1 while calculating mknk (whereas the values were qi and ri for calculatin). In the process of calculating mknk the subroutine must in general call itself sgain, but since the values of q and r decrease with each increase in the depth of nesting of the calls, the maxi.mW! possible **nesting** depth on the 1^{th} **stage** of the computation is at most i.

17.3. <u>Lemma.</u> Suppose the values of q and r are taken to be qi and r1 in the execution of the Toom algorithm. Then the base b notations for the integers mk and nk defined in 16.4 have no more than qi-1 (ri-1 + 1) di?,its eacn, provided only the initial values r1 .md q1 are chosen so that

17.4. $$r_1 = 2^{[\sqrt{\log q_1}]},$$

and q1 is sufficiently large.

Proof. We have

$$m_k = \sum_{j=0}^{r_i} \alpha_j k^j < (r_i + 1) b^{q_i} (2r_i)^{r_i},$$

since each Clj has at most qi digits. Thus, if t(11\) is the number of digits in 11\• then (since b 2)

17.5. 1(-mk) logb "k + 1 log(ri + 1) + q i + r1 loc(2r1) + 1

q i + r/

if r1 is sufficiently large. Now s e t t s i - 1. Then q i = qtrt, and

$$\log r_i^2 = 2 \log r_i = 2[\sqrt{\log q_i}] \leq 2 \overline{\log(a_t r_t}f$$

$$\leq 4\sqrt{\log q_t} \leq \log q_t,$$

if qt is sufficiently large. Thus ri2 qt, so by 17.5, L(11\)

q i + qi-1 • q1_1(r1_1 + 1), which proves the lemma for the case of"\.·
The proof f o r ¾ is the same.

We shall now find an upper bound for the time required by the Turing machine to multiply, using the time estiltlates calculated in section 19. First, let ti be the maximum ,ossible number of steps the

CjUbr=:iutine coul! r c;uirc to x::ul ! !y t1,,oIHJ_·.aber3for Jilic:.iit s t:s '1 qi anrl r • r1• The t1.tie taken Ly the su:,routine to perfom the ,ultirlic-ttion c.:.r he divided into two parts, depenJini:t on P1Cthcr t',e "l:tchine is currently directly under the control of t,1e original entrnnca to the suhroutine, or whether it is under the control of a 1:tter entr ncu. .Accord in:; to 9,;,ction 19, e:,e fir:,t part does not ex- cP.ed $nr1^5$ 11 step for some constnn D. ince t su routine ust call itseJ.f 2r1 + 1 tbe to CJmputa the .!r1 + 1 rroducts 'ltnk, and ,ince '>• 1 = a 17. 3 t!le vnlues **of** q a:-d ;:-for eacil ***oi*** these times are at nost '1• q1_1, r • r1_1, the second pnrt of the oultiplyi g tine docs not exce d (2r1 + l)t1_1• Thtrofore,

17.6. $$t_i \leq Dr_1^5 q_1 + (2r_i + 1)t_{i-1}, \quad i = 2,3, \ldots.$$

From 17.6 we shall prove by induction

17.7. $$t_i \leq Cq_{i+1}2^{4\sqrt{\log q_{i+1}}}, \quad i = 1,2,\ldots$$

for some constant C. ge assume 17.4 holds, and that q1 is larr.e enough for the assertions below to be valid. When i e 1. 17.7 certainly bolds for sufficiently lar e c. Now suppose i > 1. By the induction hypothesis, we have

$$t_{i-1} \leq Cq_i 2^{4\sqrt{\log q_i}},$$

and **by 17**.2,

$$r_i = 2^{[\sqrt{\log q_i}]}.$$

Since (2r$_i$ ' 1', S ,' .^{1}r1, 17.a no gives

11.e. $\cdot t_i \; S \; qi^{2d+5(/QJ} + qi^{21.1+[fQ]+4,'Q-1-c}$,

where <1 = log D, Q • log q1, and c = log C. To carry out the induction, we must establish 17.7. Since qi+1 • q1r1, this is equivalent to showing

17.9. $$t_i \le q_i 2^{[\sqrt{Q}]+4\sqrt{Q+[\sqrt{Q}]}+c}.$$

Thus we need only show the right side of 17.3 does not exceed the ri ht side of 17.9. We do this by showing the second term of 17.8 does not exceed $2^{-\cdot 2}$ times the right side of 17.9, and the first term of 17.8 does not exceed (1 - 2-·2) times the right side of 17.9. That is, we establish the two inequalities

17.10. d + 5[/QJ [/q] + 4/q I· (/Q] + c - R.

and

17.11. 1.1 + [«i] + 4/ii + C .S. [/Q] + 4iQ+[f(j"J + C - .2,

where -R. • log(1 - $2^{-\bullet 2}$). If we choose C so large that c • log C d + *l*., then 17.10 is valid, and 17.11 certainly holds for sufficiently large Q (and hence for sufficiently large q1). Hence the induction is complete and 17.7 is established.

Now let T1(n) be the number of steps required for the machine to designate the first o (low-order) digits of the product of the two numbers on the input tape. Let qi be the value of q selected by the subroutine in multiplying an input segment which includes the first n input pairs of digits. Then $q_i < n$. Previous to this pass of the subroutine, the subroutine was entered at most $i + 1$ times from the main routine, each time with a value of q less than q_i. Since the main routine does much less computation than the subroutine, it is clear that at any point in the computation after [illegible] one output digit has been designated, the total time to date used by the subroutine exceeds the time used by the main routine. From these facts [illegible] $j < r_j$ [illegible] r_j [illegible] $j-1 < t_j$ [illegible] $1, 2, \ldots,$ [illegible] conclude

17.12. $$T_1(n) < 4t_i.$$

Since $q_i < n$, $q_{i+1} = q_i r_i$, and $r_i = 2^{[\sqrt{\log q_i}]}$, we have by 17.7 and 17.12

17.13. $$T_1(n) < 4Cn\ 2^{4\sqrt{\log n} + \sqrt{\log n} + \sqrt{\log n}}$$

But for any $x > 0$, [illegible] so 17.13 yields

17.14. $$T_1(n) < 16Cn\ 2^{5\sqrt{\log n}}.$$

To establish the time bound in theorem 17.1, it suffices to eliminate the constant 16C. According to the speed-up theorem of Hartmanis and Stearns [6], the computation time T(n) of a multitape Turing machine can always be decreased by a constant factor c > 0, provided cT(n) [illegible] n for all n, and provided the machine is a sequence generator (i.e. has no inputs). The speed-up theorem is proved by replacing the old machine with a new one with a larger alphabet of cape s1::.bols. Eacho(the uew tape symbols encodes a flnitc sequE<nce of the 01.:, so ouc step of tile uew machine dons the woi:k of several st:eps of tnc olt.. Tl.c condition oi no inputs does not hold for our multiplyinsi nlaclline, hut the method of llartmanis and Stearns can 1:e made to apply anyway. All that is necessary is to have the multi·· plying wacL.ine encode segments of the input tai,e into th" more compact format: as tuey are read in.

Tne tiue bow,d T(n) :, n $2^{5\ 1\ 1\text{-}0\ \overline{\quad n}}$ state<l in theorem 17 .1 requires that tne ruachine put out product digits at the rate of one per l'lachine cycle for some initial period of time. This is easily accomplished by attaching a large finite-state multiplier wnich multiplies by ''table look-up" until ttie subroutine is able to supply the first string of product digits.

Thus the proof of **17.1** will be complete as soon as the inequality **17.b** is establisbe<l in section **19.**

18. <u>Solution of the linear eauations.</u>

The algorithln used to solve the system 16.5 of linear equations is the standard one of applying elementary row operations to the coefficfont matrix until it is reduced to the identity 111atrix. We shall discuss this algorithm in some detail to show that it can be carried out sufficiently rapidly both for the case of integer amlti- plication and polynomial multiplication.

It will be convenient to cast the discussion in general setting. Thus we assume the problem is to solve for the yk's the system

18.1. $$y_0 + X_i y_1 + X_i^2 y_2 + \dots + X_i^{n-1} y_{n-1} = Y_i, \quad i = 1,2,\dots,n,$$

where $X_1, X_2, \dots, X_n$ and $Y_1, Y_2, \dots, Y_n$ are elements from some integral domain R. For the case of integer multiplication, R is the ring of ordinary integers, while for polynomial multiplication, R is the ring of polynomials in W over Z_b, where b is prime.

Elementary row operations are applied to the coefficient matrix

$$A = \begin{pmatrix} 1 & X_1 & X_1^2 & \cdots & X_1^{n-1} \\ 1 & X_2 & X_2^2 & \cdots & X_2^{n-1} \\ \vdots & & & & \\ 1 & X_n & X_n^2 & \cdots & X_n^{n-1} \end{pmatrix}$$

to transform it to the identity matrix. When these same row operations are applied in order to the column matrix

$$\begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}$$

the result is a column matrix giving the values of $y_0, \dots, y_{n-1}$. Since these values lie in R, it is always possible to choose the row operations in such a way that all entries in the n x n matrix are in R after each step in the computation. One simply multiplies rows by suitably large members of R. It is less obvious, but nevertheless true, that the entries in the matrix remain in R without any "extra" multi-plications. This fact will come out of the description of t,1e algorithm below and we shall use it in the next section to keep the computation time down.

In describing the row operations, we shall restrict our attention to then x n matrix of X's, and ignore the column matrix. Notice that since y0, ••• ,yn-1 all lie in R, the entries in the colw:nn matrix will automatically lie in Rafter each row operation has been applied, provided the er:tries in the n x n matrix remain in R.

It will be convenient to think of "½,X_2, ••• ,Xn not as n- mbers of R, but as indeterminants or variables for polynomials with coefficients in R. The specific case we are interested in can then be obtained by substituting the p-roper values in R for tbe variables Xi•···,Xn. W shall denote the set of all polynomials in the variables x1, ... , with coefficients in R by R[Xi,···• J. If P(X) is a polynomial in X with coefficients in R[X1, ... , J, then P(X) is monic provided the coefficient of the highest power of X is l; or, if X does not occur, then P(X) is monic exactly if it is the constant polynomial 1. Thus 1 and X + $3X_1^2$ are monic as polynomials in X, but x_1 +½•and XXi are not.

The algorithm consists of n major steps. We denote by the result of applying the first k of these steps to the matrix A, k • 0,1, ••• ,n. We shall prove by induction on k that consists of four rectangular sub-matrices, as follows:

$$A_k = \begin{pmatrix} I_k & N_k \\ O_k & T_k \end{pmatrix},$$

where I_k is the $k \times k$ identity matrix, O_k is the $(n - k) \times k$ matrix of zeroes, N_k is a $k \times (n - k)$ matrix of polynomials over the integers, and T_k is a $(n - k) \times (n - k)$ matrix whose $(i, j)^{\text{th}}$ entry is

$$P^{(k)}_{k+j}(X_{k+i}).$$

Here $P^{(k)}_{k+1}(X)$, $P^{(k)}_{k+2}(X), \ldots, P^{(k)}_{n}(X)$ are monic polynomials in X of degrees $0, 1, 2, \ldots, n - k - 1$ with coefficients in $R(X_1, \ldots, X_k)$. Thus, in particular, $P^{(k)}_{k+1}(X)$ is simply the polynomial 1.

Notice that for $k = 0$, $A = A_0$ satisfies this hypothesis; for in this case and T_k coincide, and the polynomials $P^{(0)}_1(X), \ldots, P^{(0)}_n(X)$ are successive powers of X.

The, $(k + 1)^{\text{st}}$ major step of the algorithm consists of the following.

(1) Subtract the $(k + 1)^{\text{st}}$ row from each succeeding row, and appropriate multiples of the $(k + 1)^{\text{st}}$ row from each preceding row, so that each entry except the diagonal entry of the $(k + 1)^{\text{st}}$ column is 0. Then the entries of the row $k + i$, $i > 1$, are

$$P^{(k)}_{k+2}(X_{k+i}) - P^{(k)}_{k+2}(X_{k+1}) \quad \cdots \quad P^{(k)}_{n}(X_{k+i}) - P^{(k)}_{n}(X_{k+1}).$$

But notice that for any monic polynomial $P(X)$, we have $P(X) - P(Y) = (X - Y)Q(X)$, where $Q(X)$ is monic of degree one less than $P(X)$ (and has polynomials in Y as coeffi.cients). Thus ,.,ecan rewrite tile rov in the form

<¾+i - ¾+1 >Qi+2<¾ +i> $\cdots$ $(X_{k+i} - X_{k+1})Q_n(X_{k+i})$.

k + 1

The second part of the (k + 1) step is

(ii) Divide row k + i by (+i - \c+i>• for 1 • 2,3, ••• ,n. The result is

Qk+2<\;+1> $\cdots$ $Q_n(X_{k+i})$

k + 1

(i) and (11) complete the (k + 1)$\frac{8t}{}$ step of the algorithm. The induction hypothesis is satisfied fork+ 1 if we set P 1)(X) • Q_{k+j}(X).

After all n major steps have been carried out, the result is the matrix An• then x n identity matrix, as desired.

Time estimates for carrying out the algorithm are given in the **next section.**

19. Time estimates for th Turinq machine.

The purpose of this section is to justify the inequality 17.6 by showing that if the subroutine chooses q • qi and r = ri for performing a given multiplication, then the tillle spent by the subroutine during that "pass" does not exceed Dri^{5}qi steps, for some constant D (independent of i). (The notion of "pass" will be expltLned belo,.r). In order to show this, it is necessary to give a rather explicit description of the Turing machine. The description will he of a :.:.J<:hinewhich multiplies integers in base b notation, but only slight modifications are necessary to handle the case of polynomial multiplication.

The machine we shall describe has t,,enty linear tapes (with one read-i,rite head per tape) in addition to the read-only input tape. The number can certainly be substantially reduced, hut onJ.y with some sacrifice of clarity of exposition. We shall call ten of the twenty tapes work tapes. Of the remaining ten, three are designated scratch tapes, and the other seven are named as follows: storage tape, output tape, argument tape, r-tape, q-tape, R-tape, Q-tape. Any other specific names we use will refer to work tapes.

The heads can read and write any of the b + 6 symbols 0,1, •••, b - 1 (the b digits), 6, c, A, i, - **(minus)**, blank. In general, integers are stored in base b notation on the tapes, with the symbol ,_, to the right oE negative integers. Initia1ly the base b expansions of two inte_\lers and N are assumed to be on the input tape w-..i.tthe read-only head scanning the low order (right-most) digit. All squares of all other tapes are blank. The machine acts as though all the work tapes are one-way to the right. That is, initially ic places the symbol on each of the work **tapes,** and the heads never raove to the left of this symbol. The macb.i.ne operates in such a way that it never leaves blanks interspersed with the other symbols on the work-tapes; that is, the only blanks are either to the left of all non-blank symbols or to the right of them.

Each time the subroutine is entered, tile heads of the <fork tapes each move to the first blank square to the right of their present location and print a *A*. The heads remain to the right of this *A* uutil exiting from that pass through the subroutine, at which time the" and all information to the right of it is erased. In this way the information fron, lll'1nydifferent passes through the subroutine i,; simultaneously stored on the work tapes in segments separated by ,.•s. (Thi should clarify the notion of "pass").

In exiting from a given pass through the subroutine the machine must know whether to return control to the subroutine or to the main routine. (There is no ambiguity about where to return within these routines since each calls the subroutine from only one point). Thus when the 01ain routine clllls the subroutine the machine prints a T to the right of the information on the work tapes instead of a"·

'the input to the subroutine - namely the inte urs to be multiplied - is always furnished to the subroutine via the argument **tape,** And the subroutine writes the prod•u:t on the argument tape before exiting. Since this information is transferred onto 1lork tapes immediately after entering and after exiting respectively there is no need to preserve the information on the argument tape between pasRes to the subroutine. The same applies to the three scratch tapes. The subro tine does not use the remaining non-work tepes, except the storage tape, where the use is uniform for all passes.

The initial segment of the input tape currently being operated on is always stored on the storage tape. Once the main routine has read in the next increnent of the inpu tape and determined q and r for a given iteration of the algorithm, the machine indicates the division of tne numbers on the storage tape into segments of length q by placing tne symbol 3 every q digit-pairs, starting from t,1e right d (except in all but the initial iteration an c precedes the first q digits so no o is needed). The (r + 1)1!!. segment of length q is deline.1ted by an c instead of a 6 Thus, for example, after two iterations the storage tape looks like

& 0 ... SG&O ... 0

q2 digits q2 digits q1 digits q1 digits

The output tape is used simply to keep track of 1ihich output digits have been designated so as to prevent repetitions.

The following are outlines of the main routine and subroutine.

<u>1',ain ,!loutine</u>

(1) Update the values of q and r which appear on the q and r tapes.

(2) Initiate or continue the copying of the input tape onto the storage tape until a total of (r + 1)q digit-pairs appeal' on the storage tape. In the process, insert several •sand then an c, so that r + 1 blocks of q ?airs each are delineateJ (in addition to any smaller blocks already delineated).

(3) Copy the storage tape up to the last (left-most) c onto the argument tape, on,itting all 5's and c's. (Erase all else on the argument tape) •

(4) Sbift all heads on work tapes to the ri.tht until a blank syr.i!:ol is reached, print a There, and enter the subroutine.

(5) After return:!.ng from the subroutine, copy the product from t:1c argument tape onto the outimt tape and designate as output the portion not already appenrin!t on the utput tape.

(6) Go to 1.

ubroutine

(1) Copy the wo integers on the argument tape onto the alpha tape and beta tape (work tapes}; one integer for each tape. Use 6's to separate the information on each tape into at raost r + l segments of length q each. fiere q and r must be a pair (qi,ri) from the sequences $q1,q_2,\cdots$ and $r1,r_2,\cdots$ of values which have occurred (or preser,tly occur) on the q and r tapes, anu are chosen as silla.11 as possible such that the number of digit-p&..irs on the argument tape does not exceed q(r + 1). q and r are r,ot determinetl explicitly, but rather the position of the o's is detemined by reference to the 6's and c's already in place on tne storage tape.

(2) If the argument has fewer than $q1r_1$ digit-pairs, perform the multiplication directly, copy the result back onto the argument tape, and go to (10).

(3) Calculate ")c - $\underset{i!0}{\overset{r}{}}$«1k^{1}, k • 0,1, ••• ,2r and place the sequence m0, m1, ••• ,m2r, separated by 6's, on the Tl work tape. The «1 are determined from the alpha tape; if there are fewer than r + 1 of the.m, fill in with zeroes. Do the same for the '\'sand the beta tape, putting the result on the T2 work tape.

(4) :!nrk mO on t:le Tl tape i:>yplacing an t to the rigt,t of it. Do the sai.'\efor nO on the T2 tape.

(5) Super:i.r.q,ose the 7(and '\. which are li'\arke<lby t•s onto the &.rgu-t!cnt tape, print :i.s to tbe right of t!,c infon,atior, on all work tapes, :md go to (1) (i.e. reenter the subroutine).

{1) After returning from the subroutine append the product nk which appears on the argument tape to the list ueing formed on the TJ work tape, separating it from previous entr1.es by a 6.

(7) If th,:; last '\: on the Tl tape (that is, m2r) is the one currently marked witl• an c, r.o to (8). Otherwise change the c's on the Tl and T2 tapes to o's, and .ark the next mk o.nd (i.e. '\:+1and 1\:+i) l'rl.the's and go to (5).

(8) The sequence mOnO, .•• ,m2rnZr now appears on the TJ tape. Use it to solve for rO, ..• ,y2r the system tf linear equations

$$Y_o + y_1 k + \ldots + Y_{2r} k^{2r} \quad k \quad 0,1,\ldots,2r.$$

(9) From the values obtained in (8), calculate

$$s = Y_o + y_1 b^q + \ldots + y_{2r} b^{2rq}$$

and place S on the argument tape.

(10) Erase the right-most A or"I and all infonnation to the right thereof on all work tapes. If this symbol is a :i.,go to (6) of the subroutine; if the symbol is a I. go to (5) of the main routine.

In estimating the time required by each part of the program we shall make use of the standard o notation, If f and g are functions of positive integers, then f{n) • O(g(n)) means there is a constant C so that f(n) Cg(n) for all positive n. We shall show that part (8) of the subroutine (solution of the linear equations) takes more time than any other part, but (8) requires no more than $O(r^5 q)$ machine steps. This will justify the assertion made at the beginning of this section.

The machine can make use of the three scratch tapes Sl, S2, S3 to carry out the additions, multiplications, and divisions required by the algorithm. It is not hard to see that if ans-digit number appears on Sl, and at-digit number on S2, then using the standard methods and no tapes besides Sl, S2, and S3 the machine can write on S3 the sum of the two numbers in O(s + t) steps, the product in O(st) steps, and the quotient (provided this is an integer) in O(st) steps.

We shall now analyze each of the parts of the program. First, the main routine:

(1) Update the values of q and r. The machine uses the four tapes labelled q, r, Q, R for this purpose. The initial values q1 and r1 are chosen so as to satisfy 17.4 with q1 large enough to enable the estimates in section 17 to be valid. The new values r', q', R', Q' are calculated by the equations

Q' D R + Q

R' ,. (,'Q]

$$q' = 2^{Q'}$$

$$r' = 2^{R'};$$

so that the recursion equations 17.2 are satisfied. The square root in the second equation is easily calculated digit by digit in O(Q') • O(log q') ste?s, while the third operation requtres Q' multiplications of numbers with no more than Q' dipits each and so requires $O((\log q')^3)$ steps. Thus, conservatively. part (1) requires O(q') steps.

(2) - (7) These p:irts require O(r'q ') machine steps plus the time required to make a pass through the subroutine.

Now let us analyze the subrout.ine. The values of q and r in this analysis are the values **selected** by part **(1)** of the subroutine.

(1), (2) These parts in the subroutine require O(rq) steps.

(3) Calculating each mk and nk requires r multiplicAtions of numbers of O(r log r) digits and r multiplications of q-di,zit n=bers by numbers with O(r log r) digits. The 4r + 2 numbers m0, ... ,m2r, n0, ••• ,n2r can be produced using the three scratch tapes and three index **tapes** in $O(qr^3 \log r)$ **steps.**

(4) - (7) These parts require O(rq) steps plus the time for the 2r + 1 posses through the subroutine.

(8) To carry out the algorithm described in section **18,** the machine first 1rrites the au ented coefficient matrix (the matrix including then+ 1st column containing the products "'k°k) on the Tl tape, with successive rows following each other,

T1: ARocI'-1£ •.. £R_2 r

Where R_K is the k^{th} row:

I\' 1̄ōk2ōk o...ōk^{2r} ō '1<

Here j indicates base b notation for **j.** (The negative entries which will subsequently appear will have minus (-) signs to the right of them).

There are two types of elementary row operations which the machine must carry out. The first is to divide a row by an integer, and the second is to subtract a multiple of one row from another. Let us consider the second. The machine is to replnce a row I\ with I\ - CRj, where C is an integer, and j k. If we asswne that the parameters **j,** k and C are specified by three tapes: the j-tape, k-tape, and C-tape, then the machine might proceed as follows, First, scan the T1 tape from left to right until Rj is reached. Then multiply the entries of Rj by *C* and put the result on a new tape T2. Next, copy the tape T1 onto a tape T3 from left to right until the row I\ is reached, then continue, except put I\ - CRj (as COllpu ed using T2) on T3 in place of I\• and then coinplete the copying. Finally, copy T3 onto T1.

To estimate the time required, it is necessary to find an upper bound for the magnitude of the entries of the coefficient matrix (excluding the n + 1^{st} column) during the course of the algorithm. Before execution of the k^{th} step in the algorithm, the entries of the k^{th} row have only been decreased, hence none of them exceeds $(2r)^{2r}$.

Thus the maximum of the entries of the matrix increases by at most a factor of $(2r)^{2r}$ on the k^{th} step. Hence after the $2r + 1$ steps of the algorithm have been completed, none of the entries exceeds (or has exceeded) $(2r)^{4r^2}$ in absolute value (notice that the first two steps increase no entries). Also, the maximum of the entries in the $n + 1^{st}$ column has increased by at most a factor of $(2r)^{4r^2}$.

Now consider again the elementary row operation "replace R_i by $R_i - CR_j$." The numbers of digits of the entries of the transformed coefficient matrix are all $O(r^2 \log r)$ and the number of digits of the $n + 1^{st}$ column is $O(q + r^2 \log r) = O(q)$. Thus the most time consuming operation is the multiplication C times the entry in the $n + 1^{st}$ column, which requires $O(r^2(\log r)q) = O(r^3 q)$ steps.

A similar analysis shows the other type of row operation — division by a constant - also requires $O(r^3 q)$ steps. Since there are $(2r + 1)^2$ row operations required to execute the algorithm, and the time for the "bookkeeping" operations is small compared with the execution time for the row operations, we find that the total number of steps required for solution of the system of equations is $O(r^5 q)$.

Besides the six work tapes already mentioned (T1, T2, T3, j, k, C), three additional work tapes easily suffice for overall control: one to specify 2r, one to keep track of the major step $(1, 2, \ldots, 2r + 1)$ of the algorithm, and one to keep track of which row is currently being operated on. This brings the total number of work tapes to nine.

(9) Since multiplication by 8 power o f b amounts simyly to writing a string of zeroes, the number of steps required to evaluate S i s comoarahle to the number of steps required to perform the additions; that is $O(r^2q)$.

(10) This requires at most $O(rq)$ steps.

20. Extensions of the algorithm.

Once le have a method for multiplying rapidly, it is possible to use a simple polynomial iteration to devise a method of dividing rapidly, and once ue have a method of dividing rapidly, we can apply Newton's nethod of approxi,nating zeroes of functions to compute algebraic functions rapidly. We present some of tl:ese metho ds in this section by provinr. theorems concernin the rate at which Turing machines can compute. The first result concerns the calculation of algebraic numbers, and has a clean proof that does not depend on a Turing machine's ability to diviJe.

20.1. Theorem ·r: For every real algebraic nur.iber a and base u 2 there is a multitape Turing machine which computes the base b expansi o n of a in time ·r(n) x:.2s $\overline{\text{log}}$ n.

Proo[. The machine doing the computation differs from machines introduced previously in that it ha.R no input. The theorem asserts that the machine designates the first n digits of the expansion Within n 25 $\overline{\log^n}$ machine steps, for all n.

---

t Hartmauis and Stearns [6] prove a weaker version with T(n) • n^2 .

To prove the theorem, we need the following lemma.

20:2. - For every algebraic number a there is a polynomial F(x) with rational coefficients anl a constant ***B*** such that IF(x) - a/ *S* Bjx - a/2 for all x with Ix - aj *S.* 1.

- Let f(x) be a polynomial of least degree over the rationals such thnt f(a) s 0. Then f(x) and **f'(x)** (the derivative of f(x)) are relatively prime, so there are polynomials (over th.e rationals) r(x) and s(x) such that

f(x)r(x) + f'(x)s(x) s -1.

Let

F(x) • f(x)s(x) + x.

Then F(a) = a, and since

F'(x) s f'(x)s(x) + f(x)s'(x) + 1,

it follows that F'(a) = 0. Thus a is a root of both the polynomial F(x) ·- a and its derivative, so there is a polynomial P(x) such that

$$F(x) - a \bullet (x - a)^2 P(x).$$

Th<? lemma follows when 1l is set equal to the :naximum of /i>(x)I on the interval /x - al *S* 1.

Theorem 20.1 is proved by constructing a r.-.achine w!lich approximates a by using an iterative technique based on the equation $x_{n+1} = F(x_n)$. Ou the first iteration, the machine producc:; a finite string of digits representing an initial segment of the base b expansion of a. This segment must be sufficiently lone so that if x1 is the number re,resented by the seglilent, then

$$\left|x_1 - \alpha\right| \; .5.b^{-4-c} ,$$

where c is a positive integer exceeding logb B.

After n iterations, the machine has produced an **initial** segment representing

20.3. $$\left|x_n - \alpha\right| \le b^{-2^n-2-c}.$$

tlo:, let

20.4. $$d \bullet 2^{n+1} + 2 + c.$$

On t h e n + $1^{\underline{St}}$ iteration, the machine proceeds according to the following instructions:

(i) Calculate an approximation Y_{n+1} to F(xn) such that

20.5. $$\left|Yn{+}l - F(xn)\right| < b^{-d-2} .$$

This is done by carrying out the required multiplications and additions to d + m places, for sufficiently large m (m is independent of n).

(ii) Round y_{n+1} to exactly d places to the right of the radix point. Thus we obtain $\mathbf{z_{n-t}1}$ such that

20.6. $$\left|zn{+}l - Y_{n+1}\right| \; .5\tfrac{1}{2} b^{-d} \bullet$$

Si.lice **P(x)** satisfies **the** conditions of the le,,-..ma,we **have** by **20.3,** 20.4, and the definition of c,

20.7. $$\left|F(x_n) - \alpha\right| \le Bb^{-d-2-c} \le b^{-d-2} \bullet$$

From 20.5, 20.6, 20.7, and the fact b 2 we obtain

20.8. $$\left|z_{n+1} - \alpha\right| < b^{-d}.$$

{iii) Let f{x) be a polynomial with integer coefficients such that f{a) = 0, and f'(a) > D. Calculate $f(z_{n+1})$ and determine whether or not $f(z_{n+1}) > 0$, and hence whether or not $z_{n+1} > a$. Now set

$x_{n+1} = z_{n+1} - b^{-d}$ if $z_{n+1} > a$, and

$x_{n+1} = z_{n+1}$ otherwise.

In either case the base b expansion of x_{n+1} ter.n:f.nates after d places to the right of the radix point. From 20.8 we see that these d places, together with those to the left of the radix point, are an initial segment of the base b expansion of a.

It remains to estimate T(k), the nl.Qllber of steps required to designate the first k output digits. These k digits will all be designated by the n^{th} iteration, where n is the least integer such that 2^n k. Thus there are constants C_1, C_2 such that on the- .!. iteration c1 multiplications are perforiued, and the number of digits involved in each does not exceed C_2k. Accordin!! to theorem 17.1, these multiplications can be performed within $c_3k\, 2^{5\sqrt{\log 1'}}$ steps for some constant c3. Since the nUllber of steps required at least doubles with each successive iteration, and since the nwnber of steps for a given iteration is bounded by a constant multiple of the number of steps required for multiplication in that iteration, there is a constant C_4 such that

$$T(k) \quad C_4k\, 2^{5\sqrt{\log k}}.$$

The constant c_4 can be reduced to 1 by applyin?- the s eed-up theorem of Rartmanis and Stearns [6].

We no- turn our attention to the p oblem of computing reciprocals. The input-output arrangements require special consideration in this case, since a radix e,<pansion of the reciprocal of an integer may not terminate. One interesting model is to assUllle the input stands foi- a real nu:uber " between 0 and 1 and is presented to the machine with hi?-h order digits first, starting with the di<?it to the rip:ht of the radix point. 1'he machine would then read in the digits of x and desi,P-nate the digits o f as fast as possible, starting with the hi h order dil(its. A little thought shows, however, that the machine could not necessarily desip;nate any of the output digits exactly until it had access to the entire input strin . For ex3l!lple, suppose the argument appearing on the input tape is in decimal notation, and be,;1ns wit:b a decimal point followed by a long string of 3's:

$$X \quad 0.3333 \cdots,.$$

Then $\frac{1}{x}$ is aporoxiwately 3, but the machine cannot determine whether the first digit in the output should be a 2 or a 3 without examining the entire strinr, of :l's to see H the digit followin11 the string is greater than or is less than 3.

This su9, ests that we relax the requirement of an exact decimal exyansion of the reciprocal, and be satisfied with an output which approximates the reciprocal. To do this, we can introduce an eleventh decimal digit t, which stands for ten. We s'1all call a string of decimal digits including t a pseudo decimal expansion. Thus fo r example, 50 = St, and 3 e 2.9t • 2.99t .• 2.999t, etc. Now when the machine is confronted with .333 ••• it proceeds to give out 2.999 ••• until either a digit e.. ceedin 3 occurs, in which case it proceeds as it would no l:1111y, or a digit less than 3 occurs, in which case it r,ives out a t followed by suitable decimal digits.

In general we can give the fol.lowinr, definition.

20.9. Def i nition. The sequence encn••1 ... c0 • d1d2d3' .. is a pseudo base b e,cpansion for a number x provided the ci's and d1's are cho sen from the set o f digits {0,1, ••• ,b} and

$$x = \sum_{i=0}^{n} c_i b^i + \sum_j d_j b^{-j}.$$

-=fore proving the theo rem concerning reciprocals and pseudo expansions, it will be useful to pro ve a simpler proposition which avoids mention of the pseudo expansions.

20 .10. '£°' .2.re'!:Fo r every base b 2 there is a multitape Turing machine sucn that for all d, when d digits representing the base b expansio n for a number A ; 0 (not necessarily an integer) appear on the input tape, the machine wi l l calculate the d highest order digits (the highest order digit is by definitio n not zero) of $\frac{1}{A}$ within d $2^{s\,l\,\overline{\log d}}$ steps.

<u>Proof.</u> By ar,plying i:ewton's method to find the zero of the fw1ction $y \quad \frac{1}{x} \bullet \bullet$ A we obtain tl1c iterative equation

20.11. $$\cdot'_{n+1} \; a2y_n - Ay_n^2 .$$

If 1,e set

20.12. $$\varepsilon_n = \frac{1}{A} - y_n ,$$

then a si..-nple calculation s!lowo

20.13. $$c_{n+1} \; a \; AEn^2 .$$

In can:ying out the approximation of¼, the machine does not use 20.11, because of the excessive time required to multiply all d di5its of A. Instead, the machine considers initial segments of the representation of A whose length successively doubles. Say the $n^{\underline{th}}$ such segment consists of the 2^n highest order digits of A, and represents **the ntttrtbPr An. The ru.achina c:ilcula.teo t1 sequence of successive ap-** proximations z1,z2, ••• to $\frac{1}{A}$ using the formula

20.14. $$z_{n+1} = 2z_n - A_{n+1} z_n^2 .$$

The right side of 20.14 is evaluated exactly, since the base b expansions of all te.rms involved terminate.

It: remains to shoe, how the machine uses tltc final approximation z_m to obtain the proper output. In all calculations, the machine ignorcs leading zeroes of the input and acts as t ough the radix point is immediately to the left oi the highest order non-zero digit of A. We shall make this assumption in estimating the error. The machine has no difficulty in locating the correct position for the radix point in $\frac{1}{A}$ after the proper string of output digits has been found.

Since A_{n+1} is a 2^{n+1}-digit approximation to A, we have

20.15. $$\left|A - A_{n+1}\right| \le b^{-2^{n+1}}.$$

Thus, setting

20.16. $$\varepsilon_n = \left|\frac{1}{A} - z_n\right|,$$

we have by observing 20.11 - 20.14,

20.17. $$\varepsilon_{n+1} \le A\varepsilon_n^2 + b^{-2^{n+1}} z_n^2$$

$$\le \varepsilon_n^2 + b^{-2^{n+1}+2}$$

Hence the number of correct places approximately doubles with each iteration, so after m iterations, where m is approximately log d, we have

$$\varepsilon_m < b^{-d}.$$

The machine now obtains $\tilde{z}_m$ by rounding z_m to exactly d+1 places. (Notice that z_m has one digit to the left of the radix point). Thus

$$\left|\tilde{z}_m - \frac{1}{A}\right| < b^{-d+1}.$$

The machine now compares $A\tilde{z}_m$ with 1 to decide whether or not $\tilde{z}_m \ge \frac{1}{A}$.

The final output B is determined by

$$B = \tilde{z}_m - b^{-d+1} \quad \text{if } \tilde{z}_m > \frac{1}{A}$$

and

$$B \bullet \tilde{z}_{m} \quad \text{if } \tilde{z}_{m} \le \frac{1}{A}$$

The estimation of the nwuber of steps required to carry out the process is essentially the same as in the proof of theorem 17.1.

20.18. Theorem: For every base b 2..2 there is a multitape Turing machine such that if a string of base b digits is written on the input tape, the first digit not zero, vhich represents **a** nw:iber A when the radix point is il:lmediately to the left of the first digit, then the machine will give out¼ in pseudo base b notation such that the first n digits are designated within $n\, 2^{5\sqrt{\log n}}$ steps, for all n.

?roof. The machine is **similar** to that of the previous theorem, but now the result of each iteration is transmitted ill!lllediately as output. Also the nW!lber A_n is nov fomed by adding b^{-2^n-3} to the nuober represented by tha first $2^n + 3$ digits of the input A so that both the conditions

20.19. $$\left|\frac{1}{A_n} - \frac{1}{A}\right| = \left|\frac{A - A_n}{A_n A}\right| \le b^{-2^n-1}.$$

and

20.20. $$A_n > A$$

are satisfied. The sequence of successive approxiilations z1,z2, ... to $\frac{1}{A}$ is again defined by 20.14, and the initial approximation z1 is chosen to be close enough so that for all n,

20.21. $$\left|\frac{1}{A_n} - z_n\right| < b^{-2^n-1}.$$

Using z_n it is possible to determine a number w_n whose base b expansion terminates after 2^n places to the right of the radix point such that

20.22. $$0 \le \frac{1}{A_n} - w_n \le b^{-2^n}.$$

This is done by rounding z_n and comparing $A_n w_n$ with 1. Now it is easily proved by induction that the machine is able to designate additional digits and pseudo digits as output after each iteration so that after the iteration the totality of digits designated forms a 2^n-place pseudo base b expansion for the number w_n. In fact, by 20.19, 20.20, and 20.22,

$$0 \le \frac{1}{A} - w_n \le b^{-2^n} + b^{-2^n-1}.$$

That is, w_n under-approximates $\frac{1}{A}$, but the error is not so big that it cannot be corrected by designating pseudo digits on the next two outputs.

The time estimates for the computation are the same as for the previous two theorems.

Of course, once it is possible to multiply and compute reciprocals in time $n\,2^{s\sqrt{\log n}}$, it becomes possible to divide in the same time and so compute any rational function in the same time. The method can be extended to well-defined algebraic functions such as $\sqrt{x}$ by applying Newton's method, but we shall not present these results in detail.

21. The ef £! computation of changing the radix base.

If one :Is to obtain a sensible classification of functions according to their minimum computation time, it is necessary to discuss how the computation time d ends on the particular notation chosen for the arguments and values of the functions. In case the functions take integers into integers. the mo,;t interesting notations are the radix exp sions, and so the problem becomes one of determining how the computation time depends on the choice of radix base. The first theorem in this section states that a multitape Turing machine can convert notation in one ba'"! to that of another in time $n\ 2^{6\overline{1\log 11}}$, so if the computation time of a function grows any faster than this then the choice of radix base has no effect on the time.

The same question of notation arises when one tries to classify real nwnbers according to their computational c001plexity. In this case. the choice of radix base can 1118ke an enormous difference in the computation time of a radix expansion because of the ·•ripple carry"; caused, for e ample, by a long string of 9's in decimal notation. In fact, theorem 21.18 below states that no matter hm• fast the computable function T(n) grows, there is a real number x coml>utable base 2 in time $n\ 2^{6\{\overline{\log n}}$, but not even computable base 3 in time T(n). Obviously, computation time of the radix expansion is not always a good way to classify nU!llbers.

Perhaps a more natural way to classify a number x as to complexity is to determine, given a nwnber £ > 0, how long it takes to produce a rational number which approximates x to within c. This idea is formalized as definition 21.6, and theorem 21.8 states that the resulting time functions do not depend heavily on the choice of radix base. We then prove theorem 21.12, which states that the base b approximability (21.6) of a nlltlber x and the ability to calculate a pseudo base b expansion for x lead to about the sa1:le time function, even though in the former case the Turing machine has available a preassigned error bound ϵ it must satisfy. This suggests that the minimum possible time needed to compute some pseudo base b expansion of x is a good way to classify x, and theorem 21.17 states that the resulting classification does not depend heavily on the choice of b.

In this section, "machine" means multitape Turing machine.

21.1 Theorem: For any pair of bases a and b there is a machine 11hich converts a giveu integer N base a into its base b notation within $n\,2^{6\sqrt{\log n}}$ steps, where n is the maximum of the lengths of the two notations for n.

Proof. r. e method is similar to the one used for multiplication. The computation is divided into successive iterations such that on the i^{th} iteratio.i the parameters q and r are chosen to be q_i anrl r_i, as in the proof of 17.1. The rq lowest order digits of the bases notation for N are divided into r pieces $a_0, a_1, \ldots, a_{r-1}$ of q digits each, so that

21.2. $$N = \sum_{i=0}^{r-1} \alpha_i a^{qi}.$$

ext each cii is converted to base **b** notation **hy** calling t.'leprocedure again r times, and tr.e n bers $a^q, a^{2q}, \bullet\bullet\bullet, a^{rq}$ are calculated using ba.qe b arithmetic. T:lis computation is ,-.,:le easier because the base b notation for a^q is available from the previous iteration. Finally, 21.2 is evaluated using base b arithmetic, completing the iJh iteration.

To estimate the time required, we note that all multiplications on the $i\underline{^{th}}$ iteration can be performed by successive multiplications of pairs of q-digit numbers; and the total number of such q-digit multiplications is $O(r^2)$. Thus, if si is the total number of steps required for the conversion 1Jhen r e r_i and q ₅ qi' then by the inequality **17.7** and the fact the routine must caıl itself ri times, we have

21.3. $$s_i \le Dr_i^2 q_i \, 2^{4\sqrt{\log q_i}} + r_i s_{i-1}$$

for so,.le constant D. We c.in now establish

21.4. $$s_i \le Eq_{i+1} \, 2^{5\sqrt{\log q_{i+1}}}$$

for sOlle consta1:t E by induction on 1. Certainly the inequality holds for i a 1, if E is chosen large enough. Asstllle nolJ i > 1 and

$$s_{i-1} \le q_i \, 2^{5\sqrt{\log q_i}}.$$

Then by applying **21.3** with ri • $2^{[\overline{(\log} q_i)}$ (cf. **17.2)** we have

21.5. $$s_i \le Dr_i q_i \, 25\{\overline{\log q_i} + Er_i q_i \, 25/\overline{\log q_i}$$

$$\le (D + E) q_{i+1} \, 2^{5\sqrt{\log q_i}},$$

where the second line uses the equation $q_{i+1} = r_i q_i$ given in 17.2. Now in general,

$$\sqrt{x} < \sqrt{x + c[\sqrt{x}]} - \frac{1}{4}$$

for sufficiently large x, so that

$$\sqrt{\log q_i} < \sqrt{\log q_i + [\sqrt{\log q_i}]} - \frac{1}{4} = \sqrt{\log q_{i+1}} - \frac{1}{4}$$

if q_i is sufficiently large. Therefore 21.4 follows from 21.5, provided E is large enough that

$$(D + B)\, 2^{-5/4} \leq E.$$

This completes the induction, and establishes 21.4. The time bound $n\, 2^{6\sqrt{\log n}}$ for the conversion follows from 21.4 by the same argument that proved theorem 17.1 from 17.7.

21.6. Definition.[7] A real number x is computable base b in T(n) if there is a machine which, upon being given an integer n (say in base b notation) as input, will designate the base b expansion of a number x_n within T(n) steps, where the base b expansion for x_n terminates after at most n places to the right of the radix point, and $|x - x_n| < b^{-n}$.

21.7. Terminology. The n^{th} place (base b) of a real number x is the n^{th} digit to the right of the radix point in the base b expansion of x. An n-place number is one whose base b expansion terminates after n places to the right of the radix point.

---

[7] This definition was suggested to the author by Michael Fischer.

21.8. Theor : Suppose a and bare radix bases and T(n) is a time function such that

$$T(n) \quad n\ 2^{6}/\overline{\log n}.$$

If a real nwnber x is approximable base a in time T(n), then x is approximable base b in time T({n $\log_a$ b} + 1), where {y} is the least integer not smaller than y.

?.roof;. Suppose the machine M1 ap1>roximates x base a in time T(n) in the sense of definition 21.6. The machine t_2 then proceeds as follows. Upon being iven n, M2 calculates m • {n loga b} + 1 (for example, by findin the number of digits in b^n base a and possibly addin 1) and ives m as input to M1. The machine M1 then designates a number xm to m places base a such that

21.9. $$|x - x_m| < a^{-m} \leq \frac{1}{a} b^{-n}.$$

This is done within T(m) steps.

The output xm can be written in the form $\frac{N}{a}$• for some integer N. No,1 M2 converts N to base b notation and calculates a^m base b. This requires O(n $2^{6/\overline{\log n}}$) steps by the proof of the previous theorem. FinaJ.ly, M2 cor:iputes the quotient $\frac{N}{a}$ base b rounded exactly to n base b places, obtaining a number z_n such that

21.10. $$|z_n - x_m| \leq \frac{1}{2b^n}.$$

A slight modification at the end of the proof of theorem 20.10 shows this can be done within n $2^{\mathrm{Sl}\overline{\log n}}$ steps. By 21,9 and 21.10 we have

$|x - z_n| < b^{-n}$, so that zn is the desired appro imation. Since T(n) n $2^{6}/\overline{\log}$ n, the total time required is O(T(m)) = O(T({n $\log_8$ b} + 1)) steps. The time can be reduced to T(m) by the speed-up theorem L-i [6].

21.11. D finition.[7] An increasing function T(n) on the positive integers is real-time countable if there is some multitape Turinl machine which g;enerates a sequence a1,ı2,... in real time such that a1 • l if i is in the range of T(n), and a1 0 otherwise.

21.12. Theorem: Suppose T(n) is real-time countable, and let x be a real nu:nber. Then

(i) If a pseudo base b expansion of x (20.9) is computable in time T(n), then x is appraximable base b in time T(n), and

(ii) If x is approximable base b in time T(n), then a pseudo base b expansion of x is computable in time T(n + 1).

Prgof. (i) To obtain an a-place approximation to x from the first n digits of a pseudo expansion of x, add b^{-n} to the pseudo expansion aud convert the f:irst n digits to standard base b notation.

{ii) ɑı suppose x is nppro:dmable base b in time T(n) by a mach1.ne M_1. Let the sequence 3i. a2 a_3 .•. be defined by the condition a_{ra} is the largest integer such that T(am) 2^m. The rAachine M2 proceeds to find a pseudo base b expansion by successive iterations. By the end of themth iteration, the machine has designated an (a_m - 1)-!)lace pseudo notation for a number u_m such that

---

7 This definition was introduced by Yamada in (16].

21.13. $$0 \quad x - um \quad b^{-¾+1} + o^{-11m}.$$

On the iteration the number am is given to I\ as input, thus obtaining an a_m-place aprroximation y to x which satisfies

21.14. $$|x - y| < b^{-a}m.$$

Next y is used to find an $(a_{ta} - 1)$-place approximation z to x satisfying

21.15. $$0 \quad x - z \quad b^{-a_{m}+1} + b.8^{1}.$$

In fact, z is (y - b-"m) truncated to am - 1 places. Now if z *S* um--l (where u_{m-1} is the previous output) then the new outout consists of enough zeroes to bring the number of output places to a_m - 1: so $u_m = u_{m-1}$. Then z u_m x, so 21.13 follows from 21.15 in this case. On the other hand, 1.f $z > u_{m-1}$, then the new output consists of the ri ht digits and pseudo digits to make u_m a z (so again 21.13 follows frOlll 21.15). That such pseudo dii:its exist .follows **from** the lnequality

21.16. $$u_{m-1} < z \leq u_{m-1} + b^{-a_{m-1}+1} + b^{-a_{m-1}}$$

and the facts that u_{Ill1} has a_{m-1} - 1 base b places and z has am - 1 base b places.

The time required for the iteration is $O(T(a_{II\backslash})) \bullet 0(2^{m})$ steps. Now given an integer n, let m be the smallest integer such that $T(n + 1) \quad 2^{m}$. Thus $n + 1 \leq a_m$ so that the first n pseudo digits are designated **by** the end of the m^{th} iteration. Thus, for some constant C the total nl.1mber of steps required to designate c:ie flrst n pseudo digits does not exceed

$$\sum_{k-1}^{m} C\, 2^{k} < 2C\, 2^{m} < 4CT(n + 1).$$

The constant **4C** can be reduced to **1** by the speed-up theoreml.

21.17. Theorem: r.iven **bases** a, **b** and **a** ti, function T(n) **satisfying**

$$T(n) \quad n\, 2^{6/\overline{\log n}},$$

if a pseudo base a expansion of a real nwnber x is computable in til!le T(n), then a pseudo base b expansion is computahle in time $T(\{(n + 1) \log_a b\} + 1)$.

Pr2_of. We may assume T(n) is real-time countable, because in fact the actual time used by the first machine is real-time countable and does not exceed T(n). Thus theorem **21.17** is an im_mediate corollary **of theorem,:: 21.8 and 21.12.**

Finally, to justify the use of the notion.s of approximability and n eudo expansion, we show there is no analog of theorems **21.8** and 21.17 for the case of ordinary radix notation.

21.18. The "l' For v JJ!S..l!.!:.Sive function T(n), there is a real number x such th.'.lt the base 2 expansion of x is computable in time $n\, 2^{6/\overline{\log n}}$, but the base 3 expansion of x is not computable **in** t:l.me T(n).

Proof. We may assume T(n) is increasing, real-time countable, and satisfies T(n) > u, for if T(n) docs not satisfy these conditions there is a largsr function that does. Now define O(n) by the primitive recursion

$$U(1) = 1$$

$$U(n + 1) = (T(U(n) + 1))^3 .$$

Then $U(n)$ is readily verified to be real-time countable. Further,

$$\lim_{n\to\infty} \frac{(T(U(n - 1) + 1))^2}{U(n)} = 0,$$

so by Corollary 9.1 of Hartmanis and Stearns [6] there is a sequence (ε_n) with values in $\{-1, 1\}$ such that ε_n is computable in time $U(n)$, but not in time $T(U(n - 1) + 1)$. Assume $c_1 = 1$, and set

$$x = \sum_{n=1}^{\infty} \frac{\varepsilon_n}{3^{U(n)}} .$$

If x were computable base 3 in time $T(n)$, then the digit in position $U(n - 1) + 1$ would be available within $T(U(n - 1) + 1)$ steps. But $\varepsilon_n = 1$ if and only if this digit is 0, and $\varepsilon_n = -1$ if and only if this digit is 2. Hence ε_n would be computable in time $T(U(n - 1) + 1)$, contrary to assumption. Thus x is not computable base 3 in time $T(n)$.

To see that x is computable base 2 in time $n2^{6\sqrt{\log n}}$, we note that $U(n)$ is real-time countable, and hence x is approximable base 3 in real time. By theorem 21.8, x is approximable base 2 in time $n2^{6\sqrt{\log n}}$. Hence by the lemma below, x is in fact computable base 2 in time $n2^{6\sqrt{\log n}}$, as required by the theorem.

21.19. Lemma. Suppose A and B are positive integers such that B is odd and $B < 2^m$. Then the base 2 expansion of $\frac{A}{B}$ nowhere contains m consecutive 1's.

Proof. Suppose, to the contrary, the binary expansion of $\frac{A}{B}$ has m consecutive 1's. Then there are integers A_1, n such that

$$\frac{A}{B} = \frac{A_1}{2^n} + \frac{2^m - 1}{2^{m+n}} + \frac{\theta}{2^{m+n}},$$

where $0 \leq \theta < 1$. Thus

$$2^{m+n}A = A_1 B 2^m + 2^m B - (1 - \theta)B.$$

Hence 2^m divides the integer $(1 - \theta)B$, which is impossible because $0 < (1 - \theta) \leq 1$ and $0 < B < 2^m$.

UIBLIOGRAP!!Y

1. Atrubin, A. J. A one-dimensional Teal-time iterative multiplier. IEEE Trans. on Elec. Comp. EC-14, No. 3 (1965), 394-399.

2. Becvar, Jiri. Real-time and cO!llplexity problems in automata theory. To be published.

3. Cobhao, A. The intrinsic ccr.:,putational difficulty of fuc.ctiona. Proceedin3s of the 1964 International Congress for Logic, :l!.; !todologv, and P!1losophy of Science, orth-!tolland Publishing Co., Amstcrd:im, 24-30.

4. Cole, S. N. Real-time cOUlputatio1l by iterative arrays of finite-state machines. Doctoral Thesis, and Report BL-36, Computation Laboratory, Harvard University, 1964.

5. Fischer, P. C. Generation of primes by a one-dimensional real-time iterative array. J, Assoc. Col!lo, :t:ic:1.12, No. 3 (1965), 388-394.

6. Hartmanis, J., and R. E. Stearns. On the computational complexity of al!lorith:ns. 1:rans. Air.er. lath, Soc. 117 (1%5), 285-306.

7. l!ennie, F. C. On-line Turing machine COl!lputationo. IEEE Trans. on Elec. Comp. EC-15, No. 1 (1966), 35-44.

8. Rennie, F. C. Ono-tape, off-line Turing machine CO'llputations. Info. and Control 8 (1965), 553-57S.

9. Hennie, F. C., and R. E. Stearns. Tua-tape simulation of multitape Turing r.uichines. J. Assoc. Comp. "lach. (to appear).

10. Hi man, G. Subgroups of finitely presented groups. Proc. Royal Soc. A, 262 (1961), 455-475.

11. McNaughton, R. The theory of automata, & survey. Advances in Compu , vol. 2, F. L. Alt ed., Academic Press, New York, 1961, 37\1-421.

12. Rabin, ;,i. 0. Reill-time computation. l J. of !1ath., Dec., 1963, 203-211.

13. Shepherdson, J. C., and H. E. Sturgis. Computability of Recursive Functions. J. Assoc. Comp. Mach. 10 (1963), 217-255.

14. Toom, A. L. The comple;dty of a scheme of functional elements realizing the multiplication of integers. Sov. Hath., Translations of Dokl, Adad. Nauk. SSSR, 4, No. 3 (1963), 714-716. (Original Vol. 150, No. 3 (1963), 496-498).

15. !vinograd, S. On the time required to perform multiplication. Research Paper RC-1564, !-'.arch3, 1966, I1!M lfat,;on Researca Center, Yorktown Heights, New York.

16. Yamada, ll. Real-time computation and recursive functions not real-tfa,e conputable. IRli Trans. on Elec. CC?. · EC-11 {1962), 753-760.