ON THE COMPUTATIONAL COMPLEXITY OF ALGORITHMS

BY
J. HARTMANIS AND R. E. STEARNS

I. **Introduction.** In his celebrated paper [1], A. M. Turing investigated the computability of sequences (functions) by mechanical procedures and showed that the set of sequences can be partitioned into computable and noncomputable sequences. One finds, however, that some computable sequences are very easy to compute whereas other computable sequences seem to have an inherent complexity that makes them difficult to compute. In this paper, we investigate a scheme of classifying sequences according to how hard they are to compute. This scheme puts a rich structure on the computable sequences and a variety of theorems are established. Furthermore, this scheme can be generalized to classify numbers, functions, or recognition problems according to their computational complexity.

The computational complexity of a sequence is to be measured by how fast a multitape Turing machine can print out the terms of the sequence. This particular abstract model of a computing device is chosen because much of the work in this area is stimulated by the rapidly growing importance of computation through the use of digital computers, and all digital computers in a slightly idealized form belong to the class of multitape Turing machines. More specifically, if $T(n)$ is a computable, monotone increasing function of positive integers into positive integers and if α is a (binary) sequence, then we say that α is in complexity class S_T or that α is T-computable if and only if there is a multitape Turing machine $\mathscr{T}$ such that $\mathscr{T}$ computes the nth term of α within $T(n)$ operations. Each set S_T is recursively enumerable and so no class S_T contains all computable sequences. On the other hand, every computable α is contained in some complexity class S_T. Thus a hierarchy of complexity classes is assured. Furthermore, the classes are independent of time scale or of the speed of the components from which the machines could be built, as there is a "speed-up" theorem which states that $S_T = S_{kT}$ for positive numbers k.

As corollaries to the speed-up theorem, there are several limit conditions which establish containment between two complexity classes. This is contrasted later with the theorem which gives a limit condition for noncontainment. One form of this result states that if (with minor restrictions)

Received by the editors April 2, 1963 and, in revised form, August 30, 1963.

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

then S_U properly contains S_T. The intersection of two classes is again a class. The general containment problem, however, is recursively unsolvable.

One section is devoted to an investigation as to how a change in the abstract machine model might affect the complexity classes. Some of these are related by a "square law," including the one-tape-multitape relationship: that is if α is T-computable by a multitape Turing machine, then it is T^2-computable by a single tape Turing machine. It is gratifying, however, that some of the more obvious variations do not change the classes.

The complexity of rational, algebraic, and transcendental numbers is studied in another section. There seems to be a good agreement with our intuitive notions, but there are several questions still to be settled.

There is a section in which generalizations to recognition problems and functions are discussed. This section also provides the first explicit "impossibility" proof, by describing a language whose "words" cannot be recognized in real-time $[T(n) = n]$.

The final section is devoted to open questions and problem areas. It is our conviction that numbers and functions have an intrinsic computational nature according to which they can be classified, as shown in this paper, and that there is a good opportunity here for further research.

For background information about Turing machines, computability and related topics, the reader should consult [2]. "Real-time" computations (i.e., $T(n) = n$) were first defined and studied in [3]. Other ways of classifying the complexity of a computation have been studied in [4] and [5], where the complexity is defined in terms of the amount of tape used.

II. Time limited computations.

In this section, we define our version of a multitape Turing machine, define our complexity classes with respect to this type of machine, and then work out some fundamental properties of these classes.

First, we give an English description of our machine (Figure 1) since one must have a firm picture of the device in order to follow our paper. We imagine a computing device that has a finite automaton as a control unit. Attached to this control unit is a fixed number of tapes which are linear, unbounded at both ends, and ruled into an infinite sequence of squares. The control unit has one reading head assigned to each tape, and each head rests on a single square of the assigned tape. There are a finite number of distinct symbols which can appear on the tape squares. Each combination of symbols under the reading heads together with the state of the control unit determines a unique machine operation. A *machine operation* consists of overprinting a symbol on each tape square under the heads, shifting the tapes independently either one square left, one square

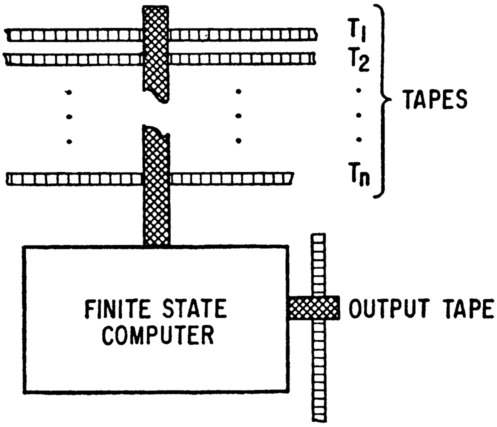

FIGURE 1. An n-tape Turing machine

right, or no squares, and then changing the state of the control unit. The machine is then ready to perform its next operation as determined by the tapes and control state. The machine operation is our basic unit of time. One tape is signaled out and called the *output tape.* The motion of this tape is restricted to one way movement, it moves either one or no squares right. What is printed on the output tape and moved from under the head is therefore irrevocable, and is divorced from further calculations.

As Turing defined his machine, it had one tape and if someone put k successive ones on the tape and started the machine, it would print some $f(k)$ ones on the tape and stop. Our machine is expected to print successively $f(1), f(2), \cdots$ on its output tape. Turing showed that such innovations as adding tapes or tape symbols does not increase the set of functions that can be computed by machines. Since the techniques for establishing such equivalences are common knowledge, we take it as obvious that the functions computable by Turing's model are the same as those computable by our version of a Turing machine. The reason we have chosen this particular model is that it closely resembles the operation of a present day computer; and being interested in how fast a machine can compute, the extra tapes make a difference.

To clear up any misconceptions about our model, we now give a formal definition.

DEFINITION 1. An n-tape Turing machine, $\mathscr{T}$, is a set of $(3n + 4)$-tuples,

$$\{(q_i; S_{i_1}, S_{i_2}, \cdots, S_{i_n}; S_{j_0}, S_{j_1}, \cdots, S_{j_n}; m_0, m_1, \cdots, m_n; q_j)\},$$

where each component can take on a finite set of values, and such that for every possible combination of the first $n + 1$ entries, there exists a unique $(3n+4)$-tuple in this set. The first entry, q_i, designates the present state; the next n entries, $S_{i_1}, \cdots, S_{i_n}$, designate the present symbols scanned on tapes $T_1, \cdots, T_n$, respectively; the next $n + 1$ symbols $S_{j_0}, \cdots, S_{j_n}$, designate the new symbols to be printed on tapes $T_0, \cdots, T_n$, respectively; the next n entries describe the tape motions (left, right, no move) of the $n+1$ tapes with the restriction $m_0 \neq$ left; and the last entry gives the new internal state. Tape T_0 is called the output tape. One tuple with S_{i_j} = blank symbol for $1 \leqq j \leqq n$ is designated as starting symbol.

Note that we are not counting the output tape when we figure n. Thus a zero-tape machine is a finite automaton whose outputs are written on a tape. We assume without loss of generality that our machine starts with blank tapes.

For brevity and clarity, our proofs will usually appeal to the English description and will technically be only sketches of proofs. Indeed, we will not even give a formal definition of a machine operation. A formal definition of this concept can be found in [2].

For the sake of simplicity, we shall talk about binary sequences, the generalization being obvious. We use the notation $\alpha = a_1 a_2 \cdots$.

Definition 2. Let $T(n)$ be a computable function from integers into integers such that $T(n) \leqq T(n+1)$ and, for some integer k, $T(n) \geqq n/k$ for all n. Then we shall say that the sequence α is *T-computable* if and only if there exists a multitape Turing machine, $\mathcal{T}$, which prints the first n digits of the sequence α on its output tape in no more than $T(n)$ operations, $n = 1, 2, \cdots$, allowing for the possibility of printing a bounded number of digits on one square. The class of all T-computable binary sequences shall be denoted by S_T, and we shall refer to $T(n)$ as a *time-function*. S_T will be called a *complexity class*.

When several symbols are printed on one square, we regard them as components of a single symbol. Since these are bounded, we are dealing with a finite set of output symbols. As long as the output comes pouring out of the machine in a readily understood form, we do not regard it as unnatural that the output not be strictly binary. Furthermore, we shall see in Corollaries 2.5, 2.7, and 2.8 that if we insist that $T(n) \geqq n$ and that only (single) binary outputs be used, then the theory would be within an ε of the theory we are adopting.

The reason for the condition $T(n) \geqq n/k$ is that we do not wish to regard the empty set as a complexity class. For if α is in S_T and $\mathcal{T}$ is the machine which prints it, there is a bound k on the number of digits per square of output tape and $\mathcal{T}$ can print at most kn_0 digits in n_0 operations. By assumption, $T(kn_0) \geqq n_0$ or (substituting $n_0 = n/k$) $T(n) \geqq n/k$. On the other hand, $T(n) \geqq n/k$ implies that the sequence of all zeros is in S_T because we can print k zeros in each operation and thus S_T is not void.

Next we shall derive some fundamental properties of our classes.

Theorem 1. *The set of all T-computable binary sequences, S_T, is recursively enumerable.*

Proof. By methods similar to the enumeration of all Turing machines [2] one can first enumerate all multitape Turing machines which print binary sequences. This is just a matter of enumerating all the sets satisfying Definition 1 with the added requirement that S_{j_0} is always a finite sequence of binary digits (regarded as one symbol). Let such an enumeration be $\mathscr{T}_1, \mathscr{T}_2, \cdots$. Because $T(n)$ is computable, it is possible to systematically modify each $\mathscr{T}_i$ to a machine $\mathscr{T}'_i$ with the following properties: As long as $\mathscr{T}_i$ prints its nth digit within $T(n)$ operations (and this can be verified by first computing $T(n)$ and then looking at the first $T(n)$ operations of $\mathscr{T}_i$), then the nth digit of $\mathscr{T}'_i$ will be the nth output of $\mathscr{T}'_i$. If $\mathscr{T}_i$ should ever fail to print the nth digit after $T(n)$ operations, then $\mathscr{T}_i'$ will print out a zero for each successive operation. Thus we can derive a new enumeration $\mathscr{T}'_1, \mathscr{T}'_2, \cdots$. If $\mathscr{T}_i$ operates within time $T(n)$, then $\mathscr{T}_i$ and $\mathscr{T}_i'$ compute the same T-computable sequence α_i. Otherwise, $\mathscr{T}_i'$ computes an ultimately constant sequence α_i and this can be printed, k bits at a time [where $T(n) \geqq n / k$] by a zero tape machine. In either case, α_i is T-computable and we conclude that $\{\alpha_i\} = S_T$.

Corollary 1.1. *There does not exist a time-function T such that S_T is the set of all computable binary sequences.*

Proof. Since S_T is recursively enumerable, we can design a machine $\mathscr{T}$ which, in order to compute its ith output, computes the ith bit of sequence α_i and prints out its complement. Clearly $\mathscr{T}$ produces a sequence α different from all α_i in S_T.

Corollary 1.2. *For any time-function T, there exists a time-function U such that S_T is strictly contained in S_U. Therefore, there are infinitely long chains*

$$S_{T_1} \subset S_{T_2} \subset \cdots$$

of distinct complexity classes.

Proof. Let $\mathscr{T}$ compute a sequence α not in S_T (Corollary 1.1). Let $V(n)$ equal the number of operations required by $\mathscr{T}$ to compute the nth digit of α. Clearly V is computable and $\alpha \in S_V$. Let

$$U(n) = \max [T(n), V(n)],$$

then $V(n)$ is a time-function and clearly

$$S_U \supset S_T.$$

Since α in S_U and α not in S_T, we have

$$S_U \neq S_T.$$

Corollary 1.3. *The set of all complexity classes is countable.*

Proof. The set of enumerable sets is countable.

Our next theorem asserts that linear changes in a time-function do not change the complexity class. *If r is a real number, we write $[r]$ to represent the smallest integer m such that $m \geqq r$.*

THEOREM 2. *If the sequence α is T-computable and k is a computable, positive real number, then α is $[kT]$-computable; that is,*

$$S_T = S_{[kT]}\,.$$

Proof. We shall show that the theorem is true for $k = 1/2$ and it will be true for $k = 1/2^m$ by induction, and hence for all other computable k since, given $k, k \geqq 1/2^m$ for some m. (Note that if k is computable, then $[kT]$ is a computable function satisfying Definition 2.)

Let $\mathcal{T}$ be a machine which computes α in time T. If the control state, the tape symbols read, and the tape symbols adjacent to those read are all known, then the state and tape changes resulting from the next two operations of $\mathcal{T}$ are determined and can therefore be computed in a single operation. If we can devise a scheme so that this information is always available to a machine $\mathcal{T}'$, then $\mathcal{T}'$ can perform in one operation what $\mathcal{T}$ does in two operations. We shall next show how, by combining pairs of tape symbols into single symbols and adding extra memory to the control, we can make the information available.

In Figure 2(a), we show a typical tape of $\mathcal{T}$ with its head on the square marked 0. In Figure 2(b), we show the two ways we store this information in $\mathcal{T}'$. Each square of the $\mathcal{T}'$-tape contains the information in two squares of the $\mathcal{T}$-tape. Two of the $\mathcal{T}$-tape symbols are stored internally in $\mathcal{T}'$ and $\mathcal{T}'$ must also remember which piece of information is being read by $\mathcal{T}$. In our figures, this is indicated by an arrow pointed to the storage spot. In two operations of $\mathcal{T}$, the heads must move to one of the five squares labeled $2, 1, 0, -1$, or -2. The corresponding next position of our $\mathcal{T}'$-tape is indicated in Figures 2(c)–(g). It is easily verified that in each case, $\mathcal{T}'$ can print or store the necessary changes. In the event that the present symbol read by $\mathcal{T}$ is stored on the right in $\mathcal{T}'$ as in Figure 2(f), then the analogous changes are made. Thus we know that $\mathcal{T}'$ can do in one operation what $\mathcal{T}$ does in two and the theorem is proved.

COROLLARY 2.1. *If U and T are time-functions such that*

$$\inf_{n\to\infty} \frac{T(n)}{U(n)} > 0\,,$$

then $S_U \subseteq S_T$.

Proof. Because the limit is greater than zero, $kU(n) \leqq T(n)$ for some $k > 0$, and thus $S_U = S_{[kU]} \subseteq S_T$.

COROLLARY 2.2. *If U and T are time-functions such that*

$$\sup_{n\to\infty} \frac{T(n)}{U(n)} < \infty\,,$$

then $S_U \supseteq S_T$.

Proof. This is the reciprocal of Corollary 2.1.

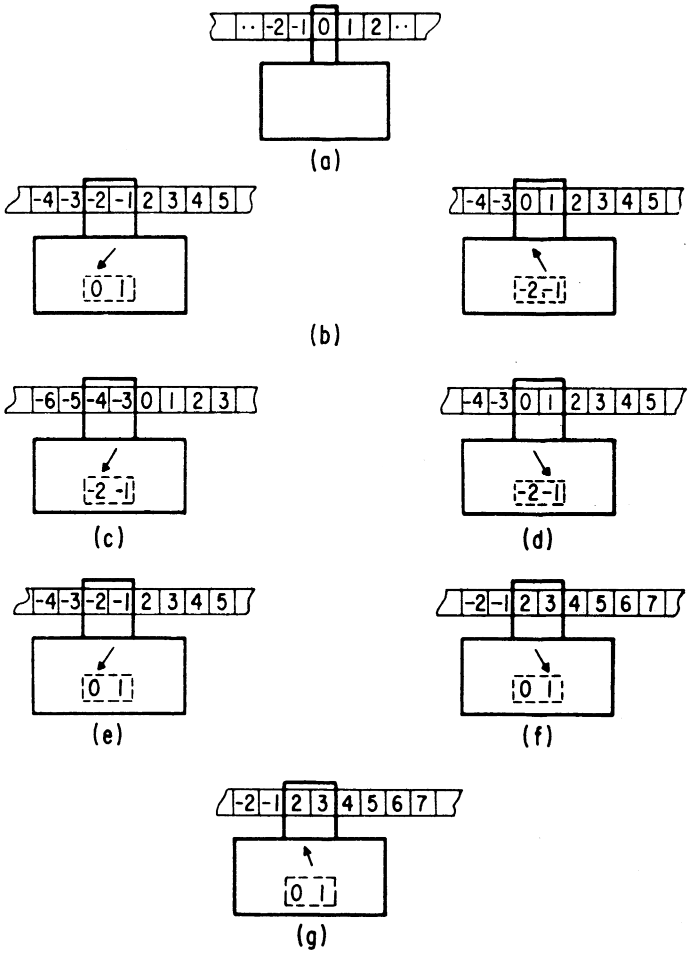

FIGURE 2. (a) Tape of $\mathscr{T}$ with head on 0. (b) Corresponding configurations of $\mathscr{T}'$. (c) $\mathscr{T}'$ if $\mathscr{T}$ moves two left. (d) $\mathscr{T}'$ if $\mathscr{T}$ moves to -1. (e) $\mathscr{T}'$ if $\mathscr{T}$ moves to 0. (f) $\mathscr{T}'$ if $\mathscr{T}$ moves to 1. (g) $\mathscr{T}'$ if $\mathscr{T}$ moves two right

COROLLARY 2.3. *If U and T are time-functions such that*

$$0 < \lim_{n\to\infty} \frac{T(n)}{U(n)} < \infty,$$

then $S_U = S_T$.

Proof. This follows from Corollaries 2.1 and 2.2.

COROLLARY 2.4. *If $T(n)$ is a time-function, then $S_n \subseteq S_T$. Therefore, $T(n) = n$ is the most severe time restriction.*

Proof. Because T is a time-function, $T(n) \geqq n/k$ for some positive k by Definition 2; hence

$$\inf_{n\to\infty} \frac{T(n)}{n} \geqq \frac{1}{k} > 0$$

and $S_n \subseteq S_T$ by Corollary 2.1.

Corollary 2.5. *For any time-function T, $S_T = S_U$ where $U(n) = \max\,[T(n), n]$. Therefore, any complexity class may be defined by a function $U(n) \geqq n$.*

Proof. Clearly $\inf(T/U) > \min(1, 1/k)$ and $\sup(T/U) < 1$.

Corollary 2.6. *If T is a time-function satisfying*

$$T(n) \geqq n \quad \textit{and} \quad \inf_{n\to\infty} \frac{T(n)}{n} > 1\ ,$$

then for any α in S_T, there is a multitape Turing machine $\mathscr{T}$ with a binary (i.e., two symbol) output which prints the nth digit of α in $T(n)$ or fewer operations.

Proof. The inf condition implies that, for some rational $\varepsilon > 0$, and integer N, $(1 - \varepsilon)\,T(n) > n$ or $T(n) > \varepsilon T(n) + n$ for all $n > N$. By the theorem, there is a machine $\mathscr{T}'$ which prints α in time $[\varepsilon T(n)]$. $\mathscr{T}'$ can be modified to a machine $\mathscr{T}''$ which behaves like $\mathscr{T}'$ except that it suspends its calculation while it prints the output one digit per square. Obviously, $\mathscr{T}''$ computes within time $[\varepsilon T(n)] + n$ (which is less than $T(n)$ for $n > N$). $\mathscr{T}''$ can be modified to the desired machine $\mathscr{T}$ by adding enough memory to the control of $\mathscr{T}''$ to print out the nth digit of α on the nth operation for $n \leqq N$.

Corollary 2.7. *If $T(n) \geqq n$ and $\alpha \in S_T$, then for any $\varepsilon > 0$, there exists a binary output multitape Turing machine $\mathscr{T}$ which prints out the nth digit of α in $[(1 + \varepsilon)\,T(n)]$ or fewer operations.*

Proof. Observe that

$$\inf \frac{[(1 + \varepsilon)\,T(n)]}{n} \geqq 1 + \varepsilon$$

and apply Corollary 2.6.

Corollary 2.8. *If $T(n) \geqq n$ is a time-function and $\alpha \in S_T$, then for any real numbers r and ε, $r > \varepsilon > 0$, there is a binary output multitape Turing machine $\mathscr{T}$ which, if run at one operation per $r - \varepsilon$ seconds, prints out the nth digit of α within $rT(n)$ seconds. If $\alpha \notin S_T$, there are no such r and ε. Thus, when considering time-functions greater or equal to n, the slightest increase in operation speed wipes out the distinction between binary and nonbinary output machines.*

Proof. This is a consequence of the theorem and Corollary 2.7.

Theorem 3. *If T_1 and T_2 are time-functions, then $T(n) = \min\,[T_1(n), T_2(n)]$ is a time-function and $S_{T_1} \cap S_{T_2} = S_T$.*

Proof. T is obviously a time-function. If $\mathscr{T}_1$ is a machine that computes α in time T_1 and $\mathscr{T}_2$ computes α in time T_2, then it is an easy matter to construct a third device $\mathscr{T}$ incorporating both $\mathscr{T}_1$ and $\mathscr{T}_2$ which computes α both ways simultaneously and prints the nth digit of α as soon as it is computed by either $\mathscr{T}_1$ or $\mathscr{T}_2$. Clearly this machine operates in

$$T(n) = \min\,[T_1(n), T_2(n)]\,.$$

THEOREM 4. *If sequences α and β differ in at most a finite number of places, then for any time-function T, $\alpha \in S_T$ if and only if $\beta \in S_T$.*

Proof. Let $\mathscr{T}$ print α in time T. Then by adding some finite memory to the control unit of $\mathscr{T}$, we can obviously build a machine $\mathscr{T}'$ which computes β in time T.

THEOREM 5. *Given a time-function T, there is no decision procedure to decide whether a sequence α is in S_T.*

Proof. Let $\mathscr{T}$ be any Turing machine in the classical sense and let $\mathscr{T}_1$ be a multitape Turing machine which prints a sequence β not in S_T. Such a $\mathscr{T}_1$ exists by Theorem 1. Let $\mathscr{T}_2$ be a multitape Turing machine which prints a zero for each operation $\mathscr{T}$ makes before stopping. If $\mathscr{T}$ should stop after k operations, then $\mathscr{T}_2$ prints the kth and all subsequent output digits of $\mathscr{T}_1$. Let α be the sequence printed by $\mathscr{T}_2$. Because of Theorem 4, $\alpha \in S_T$ if and only if $\mathscr{T}$ does not stop. Therefore, a decision procedure for $\alpha \in S_T$ would solve the stopping problem which is known to be unsolvable (see [2]).

COROLLARY 5.1. *There is no decision procedure to determine if $S_U = S_T$ or $S_U \subset S_T$ for arbitrary time-functions U and T.*

Proof. Similar methods to those used in the previous proof link this with the stopping problem.

It should be pointed out that these unsolvability aspects are not peculiar to our classification scheme but hold for any nontrivial classification satisfying Theorem 4.

III. **Other devices.** The purpose of this section is to compare the speed of our multitape Turing machine with the speed of other variants of a Turing machine. Most important is the first result because it has an application in a later section.

THEOREM 6. *If the sequence α is T-computable by multitape Turing machine, $\mathscr{T}$, then α is T^2-computable by a one-tape Turing machine $\mathscr{T}_1$.*

Proof. Assume that an n-tape Turing machine, $\mathscr{T}$, is given. We shall now describe a one-tape Turing machine $\mathscr{T}_1$ that simulates $\mathscr{T}$, and show that if $\mathscr{T}$ is a T-computer, then $\mathscr{T}_1$ is at most a T^2-computer.

The $\mathscr{T}$ computation is simulated on $\mathscr{T}_1$ as follows: On the tape of $\mathscr{T}_1$ will be stored in n consecutive squares the n symbols read by $\mathscr{T}$ on its n tapes. The symbols on the squares to the right of those symbols which are read by $\mathscr{T}$ on its n tapes are stored in the next section to the right on the $\mathscr{T}_1$ tape, etc., as indicated in Figure 3, where the corresponding position places are shown. The

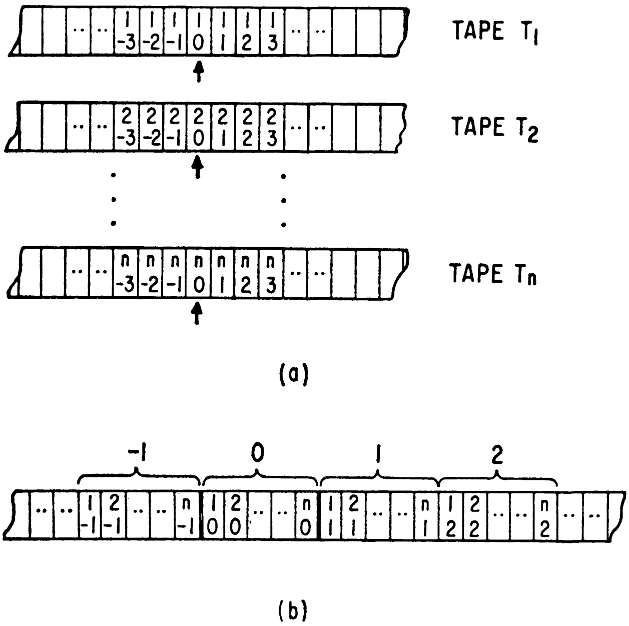

FIGURE 3. (a) The n tapes of $\mathscr{T}$. (b) The tape of $\mathscr{T}_1$

machine $\mathscr{T}_1$ operates as follows: Internally is stored the behavioral description of the machine $\mathscr{T}$, so that after scanning the n squares $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}2\\0\end{smallmatrix}\right], \cdots, \left[\begin{smallmatrix}n\\0\end{smallmatrix}\right], \mathscr{T}_1$ determines to what new state $\mathscr{T}$ will go, what new symbols will be printed by it on its n tapes and in which direction each of these tapes will be shifted. First, $\mathscr{T}_1$ prints the new symbols in the corresponding entries of the 0 block. Then it shifts the tape to the right until the end of printed symbols is reached. (We can print a special symbol indicating the end of printed symbols.) Now the machine shifts the tape back, erases all those entries in each block of n squares which correspond to tapes of $\mathscr{T}$ which are shifted to the left, and prints them in the corresponding places in the next block. Thus all those entries whose corresponding $\mathscr{T}$ tapes are shifted left are moved one block to the left. At the other end of the tape, the process is reversed and returning on the tape $\mathscr{T}_1$ transfers all those entries whose corresponding $\mathscr{T}$ tapes are shifted to the right one block to the right on the $\mathscr{T}_1$ tape. When the machine $\mathscr{T}_1$ reaches the *right most printed* symbol on its tape, it returns to the specially marked (0) block which now contains the n symbols which are read by $\mathscr{T}$ on its next operation, and $\mathscr{T}_1$ has completed the simulation of one operation of $\mathscr{T}$. It can be seen that the number of operations of $\mathscr{T}_1$ is proportional to s, the number of symbols printed on the tape of $\mathscr{T}_1$. This number increases at most by $2(n + 1)$ squares during each operation of $\mathscr{T}$. Thus, after $T(k)$ operations of the machine $\mathscr{T}$, the one-tape machine $\mathscr{T}_1$ will perform at most

$$T_1(k) = C_0 + \sum_{i=1}^{T(k)} C_1 i$$

operations, where C_0 and C_1 are constants. But then

$$T_1(k) \leqq C_2 \sum_{i=1}^{T(k)} i \leqq C[T(k)]^2 .$$

Since C is a constant, using Theorem 2, we conclude that there exists a one tape machine printing its kth output symbol in less than $T(k)^2$ tape shifts as was to be shown.

COROLLARY 6.1. *The best computation time improvement that can be gained in going from n-tape machines to* $(n + l)$*-tape machines is the square root of the computation time.*

Next we investigate what happens if we allow the possibility of having several heads on each tape with some appropriate rule to prevent two heads from occupying the same square and giving conflicting instructions. We call such a device a *multihead Turing machine.* Our next result states that the use of such a model would not change the complexity classes.

THEOREM 7. *Let* α *be computable by a multihead Turing machine* $\mathscr{T}$ *which prints the nth digit in* $T(n)$ *or less operations where* T *is a time-function; then* α *is in* S_T.

Proof. We shall show it for a one-tape two-head machine, the other cases following by induction. Our object is to build a multitape machine $\mathscr{T}'$ which computes α within time $4T$ which will establish our result by Theorem 2. The one tape of $\mathscr{T}$ will be replaced by three tapes in $\mathscr{T}'$. Tape $\underline{a}$ contains the left-hand information from $\mathscr{T}$, tape $\underline{b}$ contains the right-hand information of $\mathscr{T}$, and tape $\underline{c}$ keeps count, two at a time, of the number of tape squares of $\mathscr{T}$ which are stored on both tapes $\underline{a}$ and $\underline{b}$. A check mark is always on some square of tape $\underline{a}$ to indicate the rightmost square not stored on tape $\underline{b}$ and tape $\underline{b}$ has a check to indicate the leftmost square not stored on tape $\underline{a}$.

When all the information between the heads is on both tapes $\underline{a}$ and $\underline{b}$, then we have a "clean" position as shown in Figure 4(a). As $\mathscr{T}$ operates, then tape

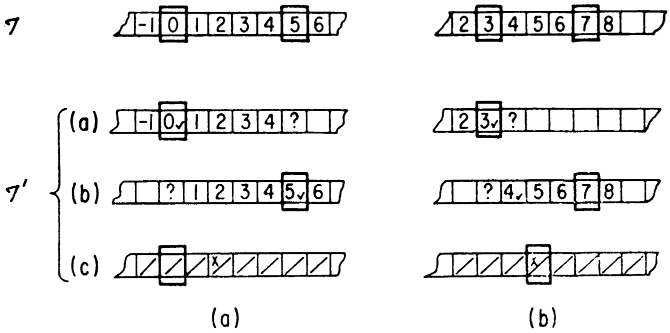

FIGURE 4. (a) $\mathscr{T}'$ in clean position. (b) $\mathscr{T}'$ in dirty position

$\underline{a}$ performs like the left head of $\mathscr{T}$, tape $\underline{b}$ behaves like the right head, and tape $\underline{c}$ reduces the count each time a check mark is moved. Head $\underline{a}$ must carry the check right whenever it moves right from a checked square, since the new symbol it prints will not be stored on tape $\underline{b}$; and similarly head $\underline{b}$ moves its check left.

After some m operations of $\mathscr{T}'$ corresponding to m operations of $\mathscr{T}$, a "dirty" position such as Figure 4(b) is reached where there is no overlapping information. The information (if any) between the heads of $\mathscr{T}$ must be on only one tape of $\mathscr{T}'$, say tape $\underline{b}$ as in Figure 4(b). Head $\underline{b}$ then moves to the check mark, the between head information is copied over onto tape $\underline{a}$, and head $\underline{a}$ moves back into position. A clean position has been achieved and $\mathscr{T}'$ is ready to resume imitating $\mathscr{T}$. The time lost is $3l$ where l is the distance between the heads. But $l \leqq m$ since head $\underline{b}$ has moved l squares from the check mark it left. Therefore $4m$ is enough time to imitate m operations of $\mathscr{T}$ and restore a clean position. Thus

$$\alpha \in S_{4T} = S_T$$

as was to be shown.

This theorem suggests that our model can tolerate some large deviations without changing the complexity classes. The same techniques can be applied to other changes in the model. For example, consider multitape Turing machines which have a fixed number of special tape symbols such that each symbol can appear in at most one square at any given time and such that the reading head can be shifted in one operation to the place where the special symbol is printed, no matter how far it is on the tape. Turing machines with such "*jump instructions*" are similarly shown to leave the classes unchanged.

Changes in the structure of the tape tend to lead to "square laws." For example, consider the following:

DEFINITION 3. A two-dimensional tape is an unbounded plane which is subdivided into squares by equidistant sets of vertical and horizontal lines as shown in Figure 5. The reading head of the Turing machine with this two-dimensional tape can move either one square up or down, or one square left or right on each operation. This definition extends naturally to higher-dimensional tapes.

Such a device is related to a multitape Turing machine by the following square law.

THEOREM 8. *If α is T-computable by a Turing machine with n-dimensional tapes, then α is T^2-computable by a Turing machine with linear tapes.*

Proof. We restrict ourselves to a single two-dimensional tape machine, $\mathscr{T}$, the generalization being straightforward.

We proceed to simulate $\mathscr{T}$ on a multitape machine $\mathscr{T}'$.

At the beginning of the simulation of a $\mathscr{T}$ operation on our machine $\mathscr{T}'$, the entire history of the two-dimensional tape is written chronologically on the right-hand part of a linear tape as shown in Figure 5. The squares record alternately the print and shift operations made by machine $\mathscr{T}$. To operate, the machine $\mathscr{T}'$ inspects the history to determine what symbol is being scanned by $\mathscr{T}$ and then returns to the end to record the next print and shift of $\mathscr{T}$. Since the squares have no names, the problem is to decide which of the past symbols (if any) is supposed to be scanned by $\mathscr{T}$. But one can easily test to see if two squares are the same by keeping track of the instructions between them and seeing if the up instructions equal the down instructions and the right equal the left. This can be done in real-time by the use of two tapes. Thus we build $\mathscr{T}'$ such that the head on the information tape shifts right until it finds the first square equal to the "present" square of $\mathscr{T}$ (this square of $\mathscr{T}'$ contains the scanned information of $\mathscr{T}$) or until the end is reached (in which case the "present" square is blank). The control remembers the symbol in this square and returns left to the end printing the next instruction and resetting en route the two tapes which test for equivalent squares. The process repeats and the ith operation of $\mathscr{T}$ requires at most $4(i+1)$ operations of $\mathscr{T}'$. Thus if $\mathscr{T}$ operates in time T, $\mathscr{T}'$ can be speeded up to operate in T^2 as was to be shown.

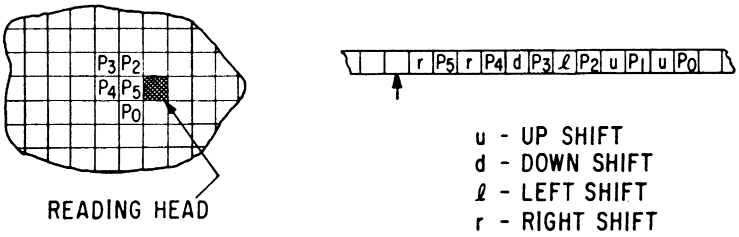

FIGURE 5. Contrast between two-dimensional tape and corresponding linear tape

As long as one can tell in real-time using ordinary tapes when the machine with a generalized tape returns to a given tape square, then a similar technique yields another "square law." Thus there is a wide class of devices that classify sequences within a square of our complexity classes.

IV. **A limit theorem.** In Theorem 2 and its corollaries, we established some conditions under which two specific complexity classes are the same. The purpose of this section is to establish a condition under which two classes are different. First we need a preliminary definition.

DEFINITION 4. A monotone increasing function T of integers into integers is called *real-time countable* if and only if there is a multitape Turing machine $\mathscr{T}$ which, on its nth operation, prints on its output tape a one whenever $T(m) = n$ for some m and prints a zero otherwise. We shall refer $\mathscr{T}$ as a T-counter.

This definition was used by H. Yamada in [3] where he shows that such common functions as n^k, k^n, and $n!$ are all real-time countable. He also shows that, if T_1 and T_2 are real-time countable, then so are kT_1, $T_1(n) + T_2(n)$, $T_1(n) \cdot T_2(n)$, $T_1[T_2(n)]$, and $[T_1(n)]^{T_2(n)}$.

It is also true that, given a computable sequence α, there is a real-time countable function $T(n)$ such that $\alpha \in S_T$. For observe that, for any $\mathscr{T}$ which prints α, one can change the output of $\mathscr{T}$ to obtain a $\mathscr{T}'$ which prints a zero whenever $\mathscr{T}$ does not print an output and prints a one whenever $\mathscr{T}$ does print an output; and $\mathscr{T}'$ is a real-time counter for some function T such that $\alpha \in S_T$. Thus the next theorem, in spite of its real-time countability restriction, has considerable interest and potential application.

THEOREM 9. *If U and T are real-time countable monotone increasing functions and*

$$\inf_{n \to \infty} \frac{T(n)^2}{U(n)} = 0,$$

then there is a sequence α that is in S_U but not in S_T.

Summary of proof. Because T is real-time countable, a set $\{\mathscr{T}_i\}$ of binary output multitape Turing machines can be enumerated such that each $\mathscr{T}_i$ operates in time $2T$ and such that each $\alpha \in S_T$ is the α_i printed by some $\mathscr{T}_i$. A device can be constructed which, when supplied (on a tape) with the instructions for $\mathscr{T}_i$, gives out the nth digit of α_i within $C_i\,[T(n)]^2$ operations, where C_i is a constant for each i which accounts for the fact that the transitions are taken off a tape and where the square accounts for the fact that the number of tapes available to simulate all the $\mathscr{T}_i$ is fixed and Theorem 6 must be appealed to. A device can now be specified to run a diagonal process on the $\mathscr{T}_i$ and print in time U. Because the inf is zero, the simulation of $\mathscr{T}_i$ will eventually catch up to U regardless of how much initial delay D_i before the simulation begins, i.e., $D_i + C_i[T(N_i)]^2 < U(N_i)$ for some large N_i. Thus if we put out an output only at times $U(n)$ [and this is possible since $U(n)$ is real-time] there is enough time to simulate all the $\mathscr{T}_i$, one after another, and print out the complement of $\mathscr{T}_i(N_i)$ at time $U(N_i)$, $N_1 < N_2 < \cdots < N_i < \cdots$. Thus we are able to print out a U-computable sequence different from all T-computable sequences.

Details of proof. A binary output machine $\mathscr{T}$ can be made into a $2T$-computer $\mathscr{T}'$ by hooking it up to a real-time $2T$-counter and extra tape. When $\mathscr{T}$ has an output, it is printed out and a mark is made on the extra tape. When the counter reaches some $2T(n)$, a mark is erased from the extra tape. If there is no mark, then $\mathscr{T}$ has not computed within $2T(n)$, and we print a zero output for that time and each time thereafter. Thus $\mathscr{T}'$ is a $2T(n)$-computer and prints the same thing as $\mathscr{T}$ whenever $\mathscr{T}$ is a $2T(n)$-computer. By modifying a list of all binary output multitape Turing machines in this way, we get a list of machines $\mathscr{T}_i$ which compute all $2T(n)$-computable sequences. Since any α printed by a binary output $2T(n)$-computer is printed by some $\mathscr{T}_i$, we know by Corollary 2.7 that each $\alpha \in S_T$ is printed by some $\mathscr{T}_i$. We can assume without loss of generality that the tapes of $\mathscr{T}_i$ are binary since one multivalue tape is equivalent to several parallel operating binary tapes. Since the $\mathscr{T}_i$ can be generated by an algorithm, it is possible to design a Turing machine which, upon the ith command, prints out the machine transitions for $\mathscr{T}_i$ on a "state tape" as shown in Figure 6. The written part of the "state tape" is enclosed by end markers and is divided into "state sections" separated by appropriate markers, the left-most section corresponding to the starting state. Each state section has the code name of the state followed by transitions sections, all separated by appropriate markers. Each transition section has a list of tape symbols followed by the corresponding output, print instruction, shift instructions, and next state code name. Since the tapes of $\mathscr{T}_i$ are binary, only a finite number of symbols are required for this tape.

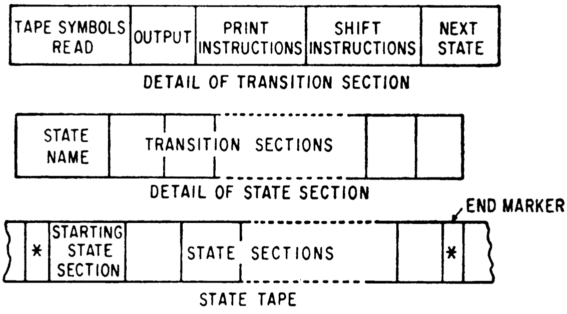

FIGURE 6. Details of state tape

Next, we show how the "state tape" can be used to simulate $\mathscr{T}_i$ in time $C_i[T(n)]^2$ where C_i is some constant. For this we use two more tapes—a "tape tape," and a "scratch tape." The tape tape stores the information on the tapes of $\mathscr{T}_i$ in the same manner as in the proof of Theorem 6 (see Figure 6). At the beginning, the reading head is to the left in the starting state section. Restricting itself to this section, the machine searches for the transition section whose "tape symbols read" correspond to the actual symbols on the tapes as listed on the tape

tape (which is blank for this first step). Then the machine looks at the output instruction and sends the output (if any) to the output control unit for further processing (this would be the printed output of $\mathscr{T}_i$). Then the print instructions are carried out on the tape tape. Then the information on the tape tape is shifted as in the proof of Theorem 6 according to the shift instructions, the "scratch tape" being used to write down the information being shifted. Finally, the head comes to the next state instruction and it then searches for the corresponding "state section." Since the time for the state transition is bounded (these being only a finite number of state transitions), we have clearly simulated the machine in the proof of Theorem 6 within a constant factor which in turn is within a constant factor of the square of $T(n)$ since $\mathscr{T}_i$ is $2T(n)$-computable. In other words, $\mathscr{T}_i$ is simulated within $C_i[T(n)]^2$.

Next we describe the "output control" which has three tapes—an "active tape," a "reserve tape," and the output tape for our device. The output control receives the output from the simulator described in the previous paragraph and from a U-counter. At the beginning of a major cycle (to be described below) of our device, the active tape has a mark for each one signal of the U-counter, the reserve tape is blank, and the output tape has as many symbols as one signals from the U-counter. Each time any one signal is received from the U-counter and there are some marks remaining on the active tape, another mark is added to the active tape and a zero is printed on the output tape. Whenever an output signal is received from the simulator, a mark is put on the reserve tape and a mark (if any) removed from the active tape. If there is no mark on the active tape, the signal is remembered by the control, and further outputs of the simulator are ignored. When another one signal is received from the U-counter, the complement of the remembered signal is printed on the tape and a signal is sent to the "central control." The number of outputs and the number of one signals from the U-counter are now equal to the number N of marks on the reserve tape and the active tape is blank. N is also the number of signals received from the simulator during this major cycle and (as we shall see) the Nth output of our device was in fact the complement of some $\mathscr{T}_i(N)$ and thus the device does not print out the T-computable sequence α_i. With the signal to the central control, the reserve tape becomes the active tape and the active tape becomes the reserve tape; and a new major cycle begins.

The parts of our device are all hooked up to a "central control" (as shown in Figure 7) which governs the major cycles. To start off, the central control signals the enumerator to print out the state tape for $\mathscr{T}_1$. Then it instructs the simulator to simulate $\mathscr{T}_1$. When the output control signals that it has just printed, for some N_1, the complement of $\mathscr{T}_1(N_1)$ as the N_1th output, the central control stops the simulation, erases the state tape and the tape tape, and then signals the enumerator to print out the state tape of the next machine.

It only remains to be shown that our device goes through a major cycle for

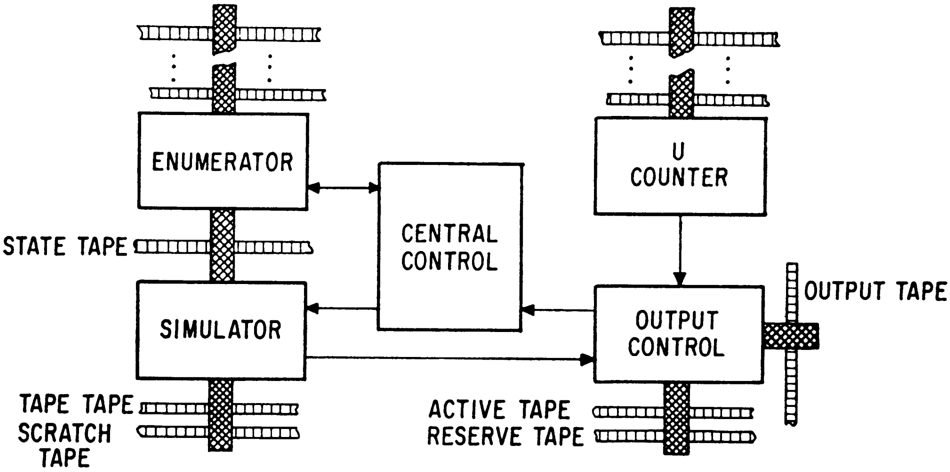

FIGURE 7. Device to print a U-sequence that is not a T-sequence

each machine $\mathscr{T}_i$. Suppose that after time D_i, the simulator begins to simulate $\mathscr{T}_i$. The device will complete this cycle and start simulating $\mathscr{T}_{i+1}$ if and only if $D_i + C_i[T(N)]^2 \leqq U(N)$ for some large N. But $D_i + C_i[T(n)]^2 \geqq U(n)$ for all n implies

$$1 \leqq \inf_{n \to \infty} \left(\frac{D_i}{U(n)} + C_i \frac{[T(n)]^2}{U(n)} \right) = 0,$$

a contradiction.

Next we give a corollary that enables us to construct specific complexity classes related by proper containment.

COROLLARY 9.1. *If U and T are real-time countable monotone increasing functions and*

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

then S_T is properly contained in S_U.

Proof. This follows from the theorem and Corollary 2.3.

Now we disprove some conjectures by using our ability to construct distinct classes.

COROLLARY 9.2. *There exist time-functions T such that $S_{T(n)} \neq S_{T(n+1)}$. Thus a translation of a time-function may give a different complexity class.*

Proof. Choose $T(n) = 2^{n!}$ which we know to be real-time countable by Yamada [3]. Clearly,

$$\inf_{n\to\infty} \frac{T(n)^2}{T(n+1)} = 0$$

and the result follows from the theorem.

COROLLARY 9.3. *There exist functions U and T such that $S_T \nsubseteq S_U$ and $S_U \nsubseteq S_T$.*

Proof. It is clearly possible to construct real-time functions U and T such that:

$$\inf \frac{T^2}{U} = \inf \frac{U^2}{T} = 0 .$$

But then Theorem 9 applies.

We close this section with some side remarks about Definition 4. Obviously, the concept of real-time countability easily generalizes to $T(n)$-countability Yamada's constructions [3] can be used to prove the closure of these classes under the rules of combination

$$U(n) + V(n), \qquad U[V(n)], \quad \text{and} \quad U(n) \cdot V(n).$$

These constructions involve the operation of one counter or the other and thus operate within time $2T$ which is just as good as T by Theorem 2.

V. **Classification of numbers.** A sequence α can be regarded as a binary expansion of a real number α. It is interesting to inquire as to whether or not numbers that we regard as simple in some sense fall into small complexity classes. Although we believe that the complexity classes to which a number belongs depend on its base, the theorems given here apply to any base.

THEOREM 10. *All the rational numbers are contained in S_n.*

Proof. It is both well known and obvious that any rational number, being ultimately periodic, can be printed one digit per operation by a zero tape machine (finite automaton).

THEOREM 11. *All the algebraic numbers are contained in S_{n^2}.*

Proof. Let $f(x) = a_r x^r + a_{r-1} x^{r-1} + \cdots + a_0$ be a smallest degree polynomial with α as a root, $0 \leqq \alpha \leqq 1$. Let α_m be a number equal to α through the mth decimal place and zero in the other bits, where m is chosen large enough so that α is the only root of $f(x)$ so approximated. We may assume without loss of generality that $f(\alpha_m) \leqq 0$ and $f(\alpha_m + 2^{-m}) > 0$.

Our machine operates as follows: The numbers α_m^i $(1 \leqq i \leqq r)$ are stored on separate tapes with reading heads on the $r \cdot m$th place (to obtain the least significant bit of α_m^r). The first m outputs are given from a separate memory tape. Then $b = f[\alpha_m + 2^{-(m+1)}]$ is computed. If $b \leqq 0$, the $(m + 1)$st output is one and we store α^i_{m+1} $(1 \leqq i \leqq r)$ [where $\alpha_{m+1} = \alpha_m + 2^{-(m+1)}$] on separate tapes, set reading heads on the $r(m + 1)$st place, and erase the α^i_m. If $b > 0$, the output is zero and we take $\alpha_{m+1} = \alpha_m$, set the heads on the r $(m+1)$st place of the $\alpha^i_m (= \alpha^i_{m+1})$ and erase scratch work. The cycle then repeats itself to calculate the other bits.

It remains to be shown that this procedure can be done in m^2 operations. This will follow if we can show that each cycle can be completed in a number of operations proportional to m. Now k numbers of length $r \cdot m$ with reading heads on the least significant bit can be added and heads returned in $2(r \cdot m + k)$ times $(r \cdot m + k$ to account for carries). Multiplication by an integer h can also be done in proportion to m since this is equivalent to adding h numbers.

The number

$$(\alpha_m + 2^{-m})^i = \alpha^i_m + \binom{i}{1}\alpha^{i-1}2^{-m} + \cdots + 2^{-im} \qquad (1 \leqq i \leqq r)$$

can thus be computed in time proportional to m since this involves adding and constant multiplication of stored numbers and since the factors 2^{-i} affect only decimal places. Then b can be computed in proportion to m by a similar argument. Finally, the $(\alpha_m + 2^{-m})$ or the α_m (depending on the output) and b can be erased in proportion to m and thus the whole operation is in proportion to m from which it follows that α is $T(m) = m^2$-computable.

THEOREM 12. *There exist transcendental numbers that belong to S_n.*

Proof. Since $n!$ is real-time countable, its counter prints (in real-time) the binary form of

$$\alpha = \sum_{n=1}^{\infty} \frac{1}{2^{n!}}$$

which is a Liouville number and known to be transcendental. For the base k, the 2 must be replaced by k. Since altering a finite set of digits does not alter the complexity or the transcendental character of α, S_n contains an infinity of such numbers which form a dense subset of the real numbers.

It would be interesting to determine whether there are any irrational algebraic numbers which belong to S_n. If this is not the case, we would have the strange result that in this classification some transcendental numbers are simpler than all irrational algebraic numbers.

VI. **Generalizations.** In this section, we consider briefly some generalizations of our theory. To start off, we investigate the complexity of recognition problems.

DEFINITION 6. Let R be a set of words of finite length taken over some finite alphabet A. Let $\mathcal{T}$ be a multitape Turing machine with a one-way input tape which uses the symbols in A. $\mathcal{T}$ is said to *recognize* R if and only if, for any input sequence α on A, the nth output digit of $\mathcal{T}$ is one if the first n digits of α

form a word in R and is zero otherwise. If T is a time-function, $T(n) \geqq n$, then R is said to be *T-recognizable* if and only if there is a $\mathscr{T}$ which recognizes R and, for any input sequence α, prints the nth output digit in $T(n)$ or fewer operations.

The restriction that inputs be taken one digit at a time prevents some of the previous results from being carried over exactly. The generalizations are nevertheless easy and some important ones are summarized in the next theorem. Had we not permitted several output digits per square in our previous work, the previous results might have also taken this form.

Theorem 13. 1. *The subset of Turing machines which are T-recognizers is recursively enumerable and therefore there are arbitrarily complex recognition problems.*

2. *If R is T-recognizable and $T(n) = n + E(n)$, $E(n) \geqq 0$, then R is U-recognizable where $U(n) = n + [kE(n)]$, $k > 0$.*

3. *If T and U are time-functions such that*

$$\inf_{n\to\infty} \frac{T(n) + n}{U(n)} > 0,$$

then if R is U-recognizable, R is T-recognizable.

4. *If R is T-recognizable by a multitape, multihead Turing machine, then it is T^2-recognizable by a single-tape, single-head Turing machine.*

5. *If U and T are real-time, monotone increasing functions and*

$$\inf_{n\to\infty} \frac{T^2(n)}{U(n)} = 0,$$

then there exist an R which is U-recognizable but is not T-recognizable.

Proof. Part 1 is proved as Theorem 1. Part 2 is proved like Theorem 2, except that n extra operations need to be added to account for the one-at-a-time inputs. Part 3 is proved like Corollary 2.1. Part 4 is just Theorem 6 and part 5 is Theorem 9.

One possible application of this is to language recognition problems. We have a straightforward proof that a context free language (see [6]) can be recognized in time $T(n) = k^n$ where k depends only on the grammar. The proof amounts to performing all possible constructions of length n and then checking the list for the nth word. The next example gives a C. F. language which cannot be recognized in real time.

Example. Let R be the set of words on the set $\{0, 1, s\}$ such that a word is in R if and only if the mirror image of the zero-one word following the last s is the same as a zero-one word between two consecutive s's or preceding the initial s. Thus $0s110s1s011$ and $110s11s011$ are in R because 110 is the mirror image of 011 whereas $011s1101s011$ is not in R. The reason that we use the mirror image instead of the word itself is that R is now a C. F. language, although we will not give the grammar here.

Proof that R is not n-recognizable. Suppose that $\mathscr{T}$ is a machine which recognizes R. Let $\mathscr{T}$ have d states, m tapes, and at most k symbols per tape square. Assume that $\mathscr{T}$ has already performed some unspecified number of operations. We wish to put an upper bound on the number of past input histories $\mathscr{T}$ can distinguish in i additional operations. The only information in storage available to $\mathscr{T}$ in i operations is the present state and the tape information within i squares of the head. From this information, at most $d \cdot k^{(2i+1)m}$ cases can be distinguished. Now the set of zero-one words of length $i-1$ has 2^{i-1} elements and this set has $2^{(2^{i-1})}$ subsets. Any one of these subsets could be the set of words of length $i-1$ which occured between various s symbols among the previously received inputs. Any pair of these subsets must be distinguishable by $\mathscr{T}$ in i operations since, if the next i inputs are an s followed by the mirror image of a word in one set but not the other, $\mathscr{T}$ must produce different outputs. But

$$2^{(2^{i-1})} > dk^{(2i+1)m} \qquad \text{for large } i,$$

and thus $\mathscr{T}$ cannot operate as it is supposed to.

Thus we have our first impossibility result. It would seem that the study of recognition problems offers an easier approach to impossibility results than the study of sequences, since the researcher can control what the machine must do.

Obviously, this theory can also be generalized to the classification of functions of integers into integers. The exact form of the theory will depend on what form one wants the answers printed, but the techniques of the proofs apply quite generally.

VII. Problems and open questions.

1. Improve upon the conditions of Theorem 9. The real-time condition on T can be eliminated by a small modification in our proof, but this is not the real issue. There is a distinct gap between Theorem 9 which states there is an α in $S_U - S_T$ and Corollary 2.1 which states that there is not; and the question is, what can be said about S_U and S_T when

$$\inf_{n\to\infty} \frac{T(n)}{U(n)} = 0\,?$$

We are inclined to believe that this condition insures $S_T \neq S_U$, especially if the functions are real-time countable, but a better approach than used in Theorem 9 will probably have to be found.

2. Are there problems which need time T^2 on a one-tape machine, but which can be done in time T using several tapes? An improvement on Theorem 6 would automatically give an improvement on Theorem 9.

3. Let Q_T be the set of all α such that for all U, $\alpha \in S_U$ implies $S_T \subseteq S_U$. Intuitively, Q_T is the set of sequences that have T as a lower complexity bound. If $\alpha \in Q_T \cap S_T$, we are justified in saying that T *is* the complexity of α. For which

time functions is $Q_T \cap S_T$ nonempty? Are there some specific properties of a sequence or problem which insure that it belongs to some specific Q_T?

4. Which, if any, irrational algebraic numbers are in S_n? If there are none, then one could exhibit numerous transcendental numbers by constructing real-time multitape Turing machines known not to be ultimately periodic. For example,

$$\alpha = \sum_{n=1}^{\infty} 2^{-n^2}$$

would be transcendental.

Acknowledgments. The authors express their gratitude to G. M. Roe, R. L. Shuey, and an anonymous referee for their helpful suggestions and constructive criticism of this work.

Added in proof. Some improvements on the results in this paper may be found in [7]. Related results are reported in [8].

References

1. A. M. Turing, *On computable numbers, with applications to the Entscheidungs problem*, Proc. London Math. Soc. (2) **42** (1937), 230–265.

2. M. Davis, *Computability and unsolvability*, McGraw-Hill, New York, 1958.

3. H. Yamada, *Real-time computation and recursive functions not real-time computable*, IRE Trans. **EC-11** (1962), 753–760.

4. J. Myhill, *Linear bounded automata*, WADD Tech. Note 60–165, Rep. No. 60–22, Univ. of Pennsylvania, June, 1960.

5. R. W. Ritchie, *Classes of predictably computable functions*, Trans. Amer. Math. Soc. **106** (1963), 139–173.

6. A. N. Chomsky, *On certain formal properties of grammars*, Information and Control **2** (1959), 137–167.

7. J. Hartmanis and R. E. Stearns, *Computational complexity of recursive sequences*, Proc. Fifth Annual Sympos. on Switching Theory and Logical Design, Princeton, N. J. 1964.

8. M. O. Rabin, *Real-time computation*, Israel J. Math. **1** (1963), 203–211.

General Electric Research Laboratory,
Schenectady, New York