ON TIME VERSUS SPACE AND RELATED PROBLEMS[†]

John Hopcroft
Cornell University

Wolfgang Paul
Cornell University

Leslie Valiant
University of Leeds

1. Introduction

The main result of this paper is to settle in the affirmative the fundamental question as to whether space is strictly more powerful than time as a resource for multi-tape Turing machines. As a consequence we establish the existence of a context-sensitive language requiring more than linear time for recognition on a multi-tape Turing machine.

Let DTIME(t(n)) [NTIME(t(n))] be the class of sets accepted by deterministic [nondeterministic] multi-tape Turing machines of time complexity t(n). Let DSPACE(s(n)) [NSPACE(s(n))] be the class of sets accepted by deterministic [nondeterministic] multi-tape Turing machines of space complexity s(n).

We show

Theorem: For all functions t: $\mathrm{DTIME}(t\log t) \subsetneq \mathrm{DSPACE}(t)$.

This implies by the standard diagonalization arguments [5].

Corollary: If the functions t and δ are constructable on tape t, and

$\inf_{n\to\infty} \delta(n)=\infty$, then $\mathrm{DTIME}(t\log t/\delta) \not\subseteq \mathrm{DSPACE}(t)$

In particular, the deterministic context-sensitive languages cannot be recognized in time less than or equal to n log n/loglogn on a deterministic multi-tape Turing machine.

In addition to the above results we prove the following:

Let

$$\log^* n = \min\{k \mid \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{k} \geq n\}.$$

Theorem: If the time required to sort n/logn binary numbers, each of length log n, on a nondeterministic multi-tape Turing machine is less than or equal to nlogn/log*(n), then for small functions t (say $t(n) \leq c^n$) the class of sets recognizable in time t on a nondeterministic single tape Turing machine is properly contained in the class recognizable in time t on a nondeterministic multi-tape Turing machine.

At the present time the containment is known to be proper only for $t(n) \leq n^2$ [6]. We observe that the above sorting problem on a single tape Turing machine can be done in less than n^2 steps. In particular we exhibit a single tape sort requiring only $\frac{n^2}{\log n}\log\log n$ steps. However, if the alphabet size is four rather than binary, then there exists a c such that cn^2 steps are required.

The last result is a fast simulation of a multi-tape Turing machine by a RAM.

Theorem: If M is a deterministic multi-tape Turing machine of time complexity $t(n) \geq n\log n$, and if log t(n) is computable on a deterministic RAM in time t/logt, then M can be simulated by a deterministic RAM in time O(t/logt).

2. Efficient space simulation of time bounded Turing machines.

In this section we present an algorithm for simulating a deterministic multi-tape Turing machine of time complexity tlogt by a deterministic Turing machine of space complexity t. For ease in understanding the constructions involved we first present a nondeterministic simulation.

Partition each tape into blocks of size $t^{2/3}$. Next modify the time bounded Turing machine so that it is block respecting that is, tape heads will cross boundaries between blocks only at times which are integer multiples of $t^{2/3}$. The modified Turing machine must be so constructed so that it

† This research was supported in part by DAAD (German Academic Exchange Service) grant No. 430/402/563/5 and the Office of Naval Research under grant N-00014-67-A-0077-0021.

runs slower by at most a constant factor c. To do this block boundaries are marked on the tapes. An additional tape is added with one block marked on it. The head on this new tape simply moves back and forth over the block and serves as a clock to indicate multiples of $t^{2/3}$ units of time. If either head on one of the original tapes attempts to cross a block boundary at a time other than a multiple of $t^{2/3}$, the simulation temporarily halts until the clock tape indicates the next multiple.

A difficulty arises in that a tape head may simply move back and forth between two adjacent tape cells which are in different blocks. In this case the simulation would be slower by a factor of $t^{2/3}$. This difficulty is overcome as follows. Let

w_1	w_2	...	w_r

be an inscription of the jth tape of the original Turing machine after $mt^{2/3}$ steps where $|w_i| = t^{2/3}$. Then the corresponding inscription for the block respecting Turing machine after $c\ m\ t^{2/3}$ steps is

	w_1^r	w_2^r		w_{r-1}^r
w_1	w_2	w_3	...	w_r
w_2^r	w_3^r			

where w^r stands for w reversed. The extra tracks are used to guarantee that $t^{2/3}$ moves can be simulated without crossing a block boundary. After $t^{2/3}$ such moves of actual simulation, the next $(c-1)t^{2/3}$ moves are used to update the tapes of the block respecting machine. The details of the simulation and of marking the block boundaries are left to the reader. (For help, consult the proof of Theorem 10.3 in [10]).

Without loss of generality let M be a block respecting deterministic k-tape Turing machine of time complexity tlogt and let w be an input for M. Divide the computation of M on input w into time segments. Each segment Δ corresponds to a time period of length $t^{2/3}$. Note that a tape head can cross a boundary only at the end of a segment. Since the original Turing machine makes at most tlogt moves and there are always $t^{2/3}$ moves between crossings of block boundaries, there are at most $t^{1/3}\log t$ time segments Δ.

Let $h(i,\Delta)$ be the position of the tape head on the ith tape of M after time segment Δ. Let $h(\Delta) = [h(1,\Delta),..,h(k,\Delta)]$ and let $h = [h(1),..., h(t^{1/3}\log t)]$.

From the sequence of head positions h we construct a directed graph G with a vertex corresponding to each time segment of the computation. Let $v(\Delta)$ be the vertex corresponding to time segment Δ. For each tape i, let Δ_i be the last time segment prior to Δ such that the ith head is scanning the same block during the segment Δ_i as during the segment Δ. The edges of G are $v(\Delta-1) \to v(\Delta)$ and for $1 \leq i \leq k$: $v(\Delta_i) \to v(\Delta)$. Because there are only $t^{1/3}\log t$ time segments Δ, the above graph has at most $t^{1/3}\log t$ vertices. To write down a description of of the graph requires space on the order of $(k+1)\ t^{1/3}\ \log^2 t$ since approximately logt bits are needed to specify each edge.

Let $c(i,\Delta)$ be the content after time segment Δ of that block on the ith tape of M, scanned by the tape head during the time segment Δ. Let $c(\Delta) = [c(1,\Delta),..., c(k,\Delta)]$. Let $f(\Delta)$ be the initial contents of those blocks, which are visited by M for the first time during the time segment Δ. Now with each vertex $v(\Delta)$ in G we can associate the information $c(\Delta)$ and $f(\Delta)$.

Let $q(\Delta)$ be the state of M after time segment Δ and let $q = [q(1),...,q(t^{1/3}\log t)]$. In order to determine the outcome of the original computation of M, we need only determine the state $q(\Delta)$ after the final time segment Δ.

Suppose we have guessed the sequence of head positions h and the sequence of states q. We want then to simulate M without using too much space and to verify that we guessed h and q correctly. Because M is deterministic and block respecting, the following holds for any Δ:

(2.1): $q(\Delta)$, $h(\Delta)$ and $c(\Delta)$ can be uniquely determined from $q(\Delta-1)$, $h(\Delta-1)$, $c(\Delta_1),...,c(\Delta_k)$ and $f(\Delta)$ by direct simulation of time segment Δ. The simulation requires space to store the contents of k blocks or $O(kt^{2/3})$.

(2.2): To store $c(\Delta)$ requires space $O(kt^{2/3})$.

$f(\Delta)$, $q(\Delta-1)$ and $h(\Delta-1)$ can be determined from the guessed sequences q and h, and the edges into vertex $v(\Delta)$ in the graph G give us the vertices associated with $c(\Delta_1),...,c(\Delta_k)$. This suggests several strategies to simulate M. Our strategy will be first to determine c(1), then c(2), c(3) and so on.

Now observe, that in order to carry out the whole simulation we need not keep in memory the contents of the blocks corresponding to all vertices since we can go back and reconstruct certain blocks whenever we need them. The question is what is the minimum number of blocks one must store at any one time in order to carry out the simulation. To answer this question we study a game on graphs.

Let G_k be the set of all finite directed acyclic graphs with indegree at most k. Vertices with indegree 0 are called <u>input vertices</u>. The game consists in placing pebbles on the vertices of such a graph G according to the following rules:

(1) A pebble can always be placed on an input vertex.

(2) If all fathers of a vertex v have pebbles, then a pebble can be placed on vertex v.

(3) A pebble can be removed at any time.

The goal of the game is to eventually place a pebble on a particular vertex v, designated in advance, by a scheme which minimizes the maximum number of pebbles simultaneously on the graph at any instance of time. Let $P_k(n)$ be the maximum over all graphs in G_k with n vertices of the number of pebbles required to place a pebble on an arbitrary vertex of such a graph. We will show that for each k, $P_k(n) \leq 0(n/\log n)$.

<u>Lemma 1</u>: For each k, $P_k(n) \leq 0(n/\log n)$.

<u>Proof</u>: For convenience let $R_k(n)$ be the minimum number of edges of any graph in G_k which requires n pebbles. Showing that $R_k(n) \geq cn\log n$ for some c is equivalent to proving that $P_k(n) \leq 0(n/\log n)$.

Let $G = (V,E)$ be a graph in G_k with $R_k(n)$ edges which requires n pebbles. Let V_1 be the set of vertices of G to which a pebble can be moved using n/2 or fewer pebbles. Let

$V_2 = V - V_1$,

$E_1 = \{(u{\rightarrow}v)/(u{\rightarrow}v) \in E,\ u,v \in V_1\}$,

$E_2 = \{(u{\rightarrow}v)/(u{\rightarrow}v) \in E,\ u,v \in E,\ u,v \in V_2\}$,

$G_1 = (V_1,E_1)$ and $G_2 = (V_2,E_2)$.

Let $A = E - (E_1 \cup E_2)$, that is A is the set of edges from vertices in V_1 to vertices in V_2.

We claim that there exists a vertex in G_2 which requires n/2-k pebbles if the game is played on G_2 only. Otherwise move a pebble to any vertex v of G with less than n pebbles by the following strategy. If v is in V_1 then only n/2 pebbles are needed. Thus assume v is in V_2. Move a pebble to v in G by using the strategy for G_2. Whenever we need to place a pebble on a vertex w of G_2 which in G has a father in V_1, move pebbles one at a time to each father of w in V_1. Since w has at most k fathers in V_1 at most n/2+k pebbles are ever placed on vertices in V_1. As soon as a pebble is placed on w remove all pebbles from vertices in V_1. Since at most n/2 - k-1 pebbles are ever used, a contradiction. Thus G_2 must have at least $R_k(n/2-k)$ edges.

Next observe that G_1 has a vertex which requires at least n/2-k pebbles. This follows from the fact that a vertex requiring n pebbles must have an ancestor which requires at least n-k pebbles. Thus G_1 must have at least $R_k(n/2-k)$ edges.

Now either $|A| \geq n/4$ in which case $R_k(n) \geq 2R_k(n/2-k) + n/4$ or else $|A| < n/4$. In the latter case, pebbles can be placed simultaneously on all vertices of V_1 which are tails of edges in A using at most 3n/4 pebbles in the process. Leaving n/4 pebbles on these vertices we have 3n/4 pebbles free after this has been accomplished. Thus G_2 must require 3n/4 pebbles for otherwise G would not require n pebbles. Now a graph which requires 3n/4 pebbles must have at least $\frac{1}{k}$ n/4 edges more that a graph requiring n/2 pebbles. (This is an immediate consequence of the fact that a vertex requiring n pebbles must have a father requiring at least n-k pebbles).

Thus in both cases

$$R_k(n) \geq 2R_k(n/2-k) + \frac{1}{k}\, n/4.$$

Solving this recurrence gives $R_k(n) \geq c\, n\log n$ for some constant c. ∎

Cook [3] has shown that $P_2(n) \geq c\sqrt{n}$ for some constant c > 0. It is an interesting question whether or not one can prove a lower bound of n/logn on the number of pebbles $P_k(n)$ for some fixed k.

<u>Lemma 2</u>: $\mathrm{DTIME}(t\log t) \subseteq \mathrm{NSPACE}(t)$.

<u>Proof</u>: Let M be a tlogt time bounded deterministic k-tape Turing machine. We construct a nondeterministic machine M' which simulates M in space t.

Make M block respecting and guess a sequence of states q' and a sequence of head positions h'. We denote these by q' and h' as opposed to the correct sequences q and h. Each such sequence has length at most $t^{1/3}\log t$. It requires space $t^{1/3}\log t$ to write down q' and space $t^{1/3}\log^2 t$ to write down h'.

From h' construct a graph G as described earlier. G has $t^{1/3}\log t$ vertices and requires space $t^{1/3}\log^2 t$ to write down.

By Lemma 1 there is a strategy to move a pebble to the output node of G never using more than $t^{1/3}$ pebbles. We can assume that this strategy has at most $\tau = 2^{t^{1/3}\log t}$ moves because there are only τ patterns of pebbles on G and there is no sense in repeating a pattern in a strategy. Having guessed the sequences q' and h', M' simulates M as follows:

```
begin
    for x = step 1 until τ do
    begin
        nondeterministically guess x^th
        move of above strategy;
        if x^th move places pebble on v(Δ)
                                        then
        begin
            compute and store q(Δ), h(Δ)
                                  and c(Δ);
            if q(Δ) ≠ q'(Δ) or h(Δ) ≠ h'(Δ)
                              then reject;
            if space used ≥ O(t) then
                                  reject;
        end else if x^th move removes pebble
                          from v(Δ) then
            erase q(Δ), h(Δ) and c(Δ) from
                          the working tape;
    end
    if after stage τ simulation is not
                     complete then reject
end
```

The essential feature of this simulation is, that after stage x, M' has computed and stored $\{c(\Delta) \mid \text{vertex } v(\Delta) \text{ has a pebble after the } x^{th} \text{ move}\}$. By (2.2) storing $c(\Delta)$ for one Δ takes space $O(t^{2/3})$. Thus if M' happens to guess a strategy which uses at most $O(t^{1/3})$ pebbles at a time (by Lemma 1 such a strategy always exists), the simulation can indeed be carried out in space $O(t)$, in which case M' accepts iff the last component of q' is the accepting state of M.

The global correctness of the above simulation is proven by induction on x and follows from (2.1), (2.2), the construction of G and the rules of the pebble game. A small but important point is that there are the edges $v(\Delta-1) \to v(\Delta)$. This guarantees that the time segment Δ of the computation of M cannot be simulated until it has been verified that $q(1),\ldots,q(\Delta-1)$ and $h(1),\ldots, h(\Delta-1)$ had been guessed correctly. The details of the correctness proof are left to the reader.

At this point the reader should be familiar with all the important ideas. We now explain how to make the simulation deterministic. There are two nondeterministic steps in the simulation algorithm. Guessing the sequences q' and h' can be replaced by cycling through all possible such sequences.

In order to determine the next move in the strategy which moves the pebbles, first construct a nondeterministic machine which given a description of G, a pattern D of pebbles on G and a number x between 1 and $2^{t^{1/3}}\log t$ (each of which can be written down in space $t^{1/3}\log^2 t$ or less) prints out the first move in a strategy which starting from D, moves a pebble on the output vertex of G never using more than $O(t^{1/3})$ pebbles and making at most $2^{t^{1/3}\log t}-x$ moves, provided such a strategy exists.

This machine can be constructed in a straightforward way using space $O(t^{1/3}\log^2 t)$. Using techniques from [11] it can be made deterministic in space $O(t^{2/3}\log^4 t)$. Using this machine during the simulation as a submachine, one can always from the achieved pattern of pebbles and from x deterministically find the next move in the right strategy. Thus we have shown

Theorem 1: If t is tape constructable, then

$$\mathrm{DTIME}(t\log t) \subseteq \mathrm{DSPACE}(t).$$

Some easy corollaries of this have been stated in the introduction. In addition one can show

Corollary 1: For all t:

$$\mathrm{DTIME}(t\log t) \subseteq \mathrm{DSPACE}(t).$$

Proof: Instead of precomputing t, successively try simulation of the proof of theorem 1 for $t(n) = 1,2,3,\ldots$ until one can carry out the simulation.

Corollary 2: If t and δ are constructable on tape t and $\lim \delta(n) = \infty$, then DTIME(t) is properly contained in the class of sets recognizable in time $\delta t \log \delta t$ on space t.

Proof: Instead of blocksize $t^{2/3}$ choose blocksize $t/(\log\delta)^{1/2}$. A time analysis of the proof of theorem 1 with this modification shows that each k-tape machine of time complexity t can be simulated by a k+1-tape machine of time complexity less than δt and tape complexity $O(t/\log\log \delta)$. Now an appeal to [15] yields the desired result.

It is an interesting open problem whether $\mathrm{NTIME}(t\log t) \subseteq \mathrm{NSPACE}(t)$. The difficulty here is that in going back and repeating a portion of a computation, we cannot be sure that the same sequence of choices is made the second time.

3. On the complexity of sorting

In this section we consider the time necessary to sort n/logn integers, each of length log n on a nondeterministic k-tape Turing machine. By using a generalization of the 4-Russians algorithm [2] we show that if sorting requires time at most $n\log n/\log^* n$ on a nondeterministic $k \geq 2$ tape Turing machine, then for small functions t (say $t(n) < 2^{cn}$) nondeterministic time t on a k-tape Turing machine is more powerful then nondeterministic time t on a one tape Turing machine.

Let $NTIME_k(t)$ be the set of languages accepted by a nondeterministic k-tape Turing machine of time complexity t and let $s_k(n)$ be the time required to sort n/logn integers, each of length log n, on a nondeterministic k-tape Turing machine.

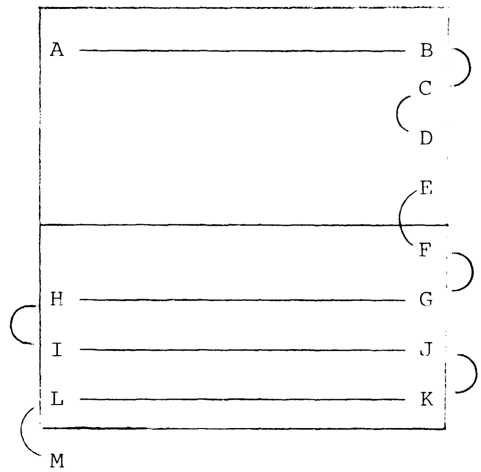
<u>Theorem 2</u>: If $s_k(n)$ is $0(n\log n/\log^* n)$ for some $k \geq 2$, then for all running times t, $n\log n \leq t(n) \leq 2^{cn}$, $NTIME_1(t) \subsetneq NTIME_k(t)$.

<u>Proof</u>: Let M be a nondeterministic single tape t(n) time bounded Turing machine. Let s be the number of tape symbols of M and q be the number of states of M. Fix a computation of M. By a crossing sequence argument one can show that for $t(n) \geq n\log n$ M uses at most t/logt tape cells [4]. Partition the tape into blocks of length $\frac{1}{k}\log t$ where $k > 12 \log s + 4 \log q$. In each block find the shortest crossing sequence. Redivide the tape into blocks where the block boundaries correspond to selected crossing sequences in such a manner that we have $kt/\log^2 t$ blocks, none of which is larger than $\frac{2}{k} \log t$.

The sum of the lengths of all crossing sequences is equal to t(n). For each crossing sequence at a block boundary there are at least $\frac{1}{k}\log t$ crossing sequences which are at least as long. Thus the sum of the lengths of crossing sequences at block boundaries is less than or equal to kt/logt.

We now represent the computation of M by a set of rectangles [11]. The top and bottom edges of the rectangles are the contents of the Turing machine tape between two selected crossing sequences and two instances of time say t_1 and t_2 respectively. The left and right edges are the segments of the crossing sequences between times t_1 and t_2. The horizontal partitioning is determined so that the longer of the two crossing sequence segments is $\frac{1}{k} \log t$ symbols long (Fig. 1) except for the last (possibly only) rectangle in a column. The entire computation of M is now simulated by first guessing the representation of the computation by rectangles, then checking that the rectangles fit together and finally verifying the rectangles. The sequence of rectangles is guessed in column order as a sequence of 4-tuples. No element of a 4-tuple (side of a rectangle) is longer that $\frac{2}{k}\log t$. Also there are at most $4k^2t/\log^2 t$ 4-tuples. Verifying that the rectangles fit together can be done in time 0(t/logt) by first verifying that all vertical borders fit and then verifying that all horizontal borders fit.

The final step in the simulation is to verify that the boundary of each square represents a portion of a computation. If we simply do this directly, the time required is some constant times the running time t of the original Turing machine. What we do instead is interpret each 4-tuple as an integer, sort the sequence of rectangle boundaries, delete duplicates and then verify. The point of this is that there are sufficient duplicates so that a substantial time savings occurs.



<u>Figure 1</u>: A portion of a computation divided into rectangles.

(Nondeterministically) sorting the sequence of at most $4k^2t/\log^2 t$ 4-tuples each of length clogt for some $c \leq 1$ can be done in time proporitional to $s_k(t/\log t)$. Duplicates are deleted by scanning the sorted sequence once.

It remains to show that the remaining rectangles can be verified fast. As the horizontal sides of each rectangle which consist of tape symbols of M have length at most $\frac{2}{k}\log t$ and as the vertical sides of each rectangle which consist of states of M have length at most $\frac{1}{k}\log t$, there are at most

$$s^{4\log t/k} \cdot q^{2\log t/k} = t^{(4\log s+2\log q)/k}$$

different rectangles in the sorted sequence. The maximum time that M can spend in one rectangle without cycling and without touching one of the crossing sequences at the right or left border is $(2\log t/k)q\, s^{2\log t/k}$. Hence each rectangle can be verified in time at most $(2\log t/k) \cdot q \cdot t^{2\log s/k}$, hence the whole sequence of rectangles without duplicates can be verified in time

$$(2\log t/k)^2 q \cdot t^{(6\log s+2\log q)/k} \leq 0(\sqrt{t})$$

since k was chosen large enough to make the exponent of t in the above equation smaller that 1/2.

So we have shown

$$NTIME_1(t) \subseteq NTIME(s_k(t/\log t)) \quad .$$

If $s_2(n) \leq n\log n/\log^* n$, then this implies via known nondeterministic hierarchy results [14]:

$$\mathrm{NTIME}_1(t) \subseteq \mathrm{NTIME}(t/\log^* t) \subsetneq \mathrm{NTIME}(t)$$

for $t(n) \le 2^{cn}$, t running time.

Concerning fast sorting algorithms, we remark that by a result from [16], any algorithm which sorts $n/\log n$ integers each of length $\log n$ by rearranging them, treating each integer as a unit, requires time $O(n\log n)$. This indicates in which direction not to look for a fast sorting algorithm.

By a crossing sequence argument one can show that $s_1(n)$ requires <u>almost</u> n^2 time. In fact if we consider a slight change in the definition of $s_1(n)$, then n^2 time is required. Let $s_1'(n)$ be the time required to sort $n/2\log n$ integers represented in binary, each of length $2\log n$ on a nondeterministic 1-tape machine.

<u>Theorem 3</u>: There exists a constant $c > 0$ such that $s_1'(n) \ge cn^2$.

<u>Proof</u>: We will show how to recognize $\{w\,w^R / w \in \{0,1\}^*\}$ in time $s_1'(n)$. The existence of $c > 0$ such that $s_1'(n) \ge cn^2$ follows immediately from the fact that recognition of $\{w\,w^R / \in \{0,1\}^*\}$ is of complexity n^2 [6] on a nondeterministic 1-tape machine. Assume a string w is written on the lower track of a single tape Turing machine. Break the string into $b = n/\log n$ blocks of length $\log n$ (we assume b is divisible by 4).

w_1	w_2	w_3	...	$w_{b/2}$	...	w_b

Shift all odd numbered blocks to the second track and one block to the right. On the lower track fill in every other block with the numbers b, $b-1$, $b-2$, ...

	w_1		w_3	...		w_{b-1}
b	w_2	$b-1$	w_4	...	1	w_b

This requires time $O(n\log n)$. Next sort the lower track, treating two blocks as a single integer of length $2\log n$. This requires time $s_1'(n)$. The result is illustrated below.

	w_1		w_3	...		w_{b-1}
1	w_b	2	w_{b-2}	...	b	w_2

The string w is a palindrome if and only if $w_1 w_b$, $w_3 w_{b-2}, \ldots, w_{b-1} w_2$ are palindromes.

Checking this takes time $O(n\log n)$.

Although sorting $n/2\log n$ binary integers each of length $2\log n$ on a single tape Turing machine requires time cn^2 for some constant c, curiously enough one can sort $n/\log n$ binary integers each of length $\log n$ in time strictly less than n^2.

<u>Theorem 4</u>: $S_1(n) \le O(\frac{n^2}{\log n} \log\log n)$.

<u>Proof</u>: Let $w_1, w_2, \ldots, w_b$ be a string of $b = n/\log n$ integers each of length $\log n$. Consider the first $\log n - 2\log\log n$ bits of each w_i. There are at most $n/(\log n)^2$ distinct combinations of $\log n - 2\log\log n$ bits.

For each combination c_i, beginning with $c_0 = 0 \ldots 0$ and in increasing order of the c_i, the Turing machine makes one pass of the tape, moving along two registers. The first has length $O(\log n)$ and contains the combination c_i. In the second register the machine collects all tails of words w_j which begin with c_i. If k_i words begin with c_i, then this register eventually requires length $2k_i\log\log n$. On completion of each pass the sequence of tails is sorted. If $a_{i1}, \ldots, a_{ik_i}$ is the sorted sequence of these tails, then the sequence $c_i\, a_{i1}, \ldots, c_i\, a_{ik_i}$ is concatenated with the output produced at earlier stages. After that the $(i+1)^{th}$ pass is started.

Clearly this is a (deterministic) sorting algorithm. The i^{th} pass takes time $O(n\log n + n2k_i\log\log n)$. The sequence of k_i tails each of length $2\log\log n$ can be sorted in time $(2k_i\log\log n)^2$.

To print out $c_i a_{i1}, \ldots, c_i a_{ik_i}$ one has to move the sequence $a_{i1}, \ldots, a_{ik_i}$ and the combination c_i at most over n cells. This takes time $O(n(2k_i\log\log n + \log n))$.

Thus the overall time spent is

$$\sum_{i=1}^{n/\log^2 n} (O(n\log n + k_i n\log\log n + (k_i\log\log n)^2)$$

$$\le O(n^2/\log n + n\log\log n \sum_i k_i + (\log\log n)^2 (\sum_i k_i)^2)$$

$$\le O(n^2/\log n + n^2\log\log n/\log n + (\log\log n)^2 n^2/\log^2 n)$$

because $\sum_i k_i = n/\log n$.

From this we conclude that $s_1(n) \le O(\frac{n^2}{\log n}\log\log n)$

4. Fast simulation of Turing machines by RAM's

Given an indirect addressing mechanism one can implement a fast sorting algorithm. Thus the proof of Theorem 2 suggests that the 4-Russians method can be used to simulate a deterministic k-tape Turing machine by a RAM speeding up the computation in the process. This is indeed the case. We first make precise our model of a RAM.

In the following list of RAM instructions

(1) A denotes the accumulator
(2) n stands for an integer
(3) <n> is the content of the memory cell with address n.

Instructions of the RAM

$A \leftarrow n$

$A \leftarrow \langle n \rangle$

$\langle n \rangle \leftarrow A$

$A \leftarrow \langle\langle n \rangle\rangle$

$\langle\langle n \rangle\rangle \leftarrow A$

$A \leftarrow A + \langle n \rangle$

$A \leftarrow A - \langle n \rangle$

<u>if</u> $A \geq (=) 0$ <u>then</u> <u>goto</u> LABEL 1 <u>else</u> <u>goto</u> LABEL 2

One unit of cost is charged for the execution of one instruction on the list

We now prove the following theorem:

Theorem 5: Let M be a deterministic k-tape Turing machine with time complexity $t(n) \geq n\log n$. If $\log t$ is computable on a deterministic RAM in time $t/\log t$, then M can be simulated by a deterministic RAM in time $O(t/\log t)$ (without increasing the total number of implied bit operations by more than a constant factor).

Proof: Without loss of generality assume M has a two symbol alphabet and M is block respecting with a block size of $\frac{1}{4k+2}\log t$.

For each 2k+1 tuple $r = (q, i_1, \ldots, i_k, c_1, \ldots, c_k)$ where q is a state of M, i_1, through i_k are integers denoting tape head positions in blocks, and c_1 through c_k are blocks of tape symbols of length $\frac{1}{4k+2}\log t$, simulate M until one of the k tape heads attempts to leave its block (some i_j takes value 0 or $\frac{1}{4k+2}\log t + 1$), but simulate for at most $\frac{1}{4k+2}\log t$ steps.

For each r store the result s in 2k+1 consecutive storage locations of a RAM in such a way that we can compute the address of the first location given r in $O(2k+1)$ operations. We present only a sketch of the method of storing the precomputed outcomes. The actual details of precomputing the outcomes and implementing the table look up are left to the reader.

Let $m = t^{1/(4k+2)}$ be the maximum value that q, i_j or c_j can take on. (The states q are assumed to be numbered starting at 1). The 2k+1 tuple $r = (q, i_1, \ldots, i_k, c_1, \ldots, c_k)$ will correspond to a leaf in a tree of height 2k+1. Each vertex in the tree has at most m descendants. The tree is stored in a 2-dimensional array A. The entry A(v,d) is a pointer to the d^{th} son of vertex v, if v is not a leaf. If v is a leaf corresponding to r, then A(v,d) is a pointer to the location of the first component of s, the precomputed outcome from r. Clearly, given r we can move along the path from the root to the vertex associated with r and hence to s in $O(k)$ steps.

The total number of distinct r's is at most m^{2k+1} or $\sqrt{t}$. For each r the Turing machine M is simulated for at most $\frac{1}{4k+2}\log t$ moves. Thus we can precompute the outcomes s for all 2k+1 tuples r and implement the look up mechanism in time $O(\sqrt{t}\log t)$.

Now note that the block respecting Turing machine makes at least $\frac{1}{4k+2}\log t$ moves between times when tape heads cross boundaries. Thus we can always simulate $\frac{1}{4k+2}\log t$ moves of the Turing machine by some constant number of moves on the RAM by table look up.

The theorem follows trivially. We require that $\log t$ be easily computed by the RAM so that we can construct the blocks of size $\frac{1}{4k+2}\log t$ and we require $t(n) \geq n\log n$ since the RAM requires n steps to look at the input.

As a consequence of the above theorem we get efficient algorithms for various problems.

Corollary 3: On a RAM with uniform cost measure one can:

(1) multiply two n-bit integers in time $O(n \log\log n)$ without ever computing numbers of more that c log n bits.
(2) multiply two nxn matrices over a finite field in time $O(n^{\log_2 7}/\log n)$

Proof: (1) follows from a result of Schönhage and Strassen [13] and (2) follows from a straightforward Turing machine implementation of Strassen's [17] matrix multiplication algorithm.

Acknowledgements:

The authors thank Professor A. Meyer and Dr. Z. Galil for fruitful and pleasant discussions.

References:

1. Aho, A.V., J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

2. Arlazarov, V.L., E.A. Dinic, M.A. Kronrod and I.A. Faradzev, "On Economical Construction of the Transitive Closure of a Directed Graph", Soviet Math Dokl 11:5, 1970, 1209-1210.

3. Cook, S.A., "An Observation of Time-Storage Trade-Off", ACM-STOC, 1973, 29-33.

4. Hartmanis, J., "Size Arguments in the Study of Computation Speeds", Proc. Symp. on Computers and Automata, Polytechnic Inst. of Brooklyn, 1971, 1-18.

5. Hartmanis, J. and R.E. Stearns, "On the Computational Complexity of Algorithms", Trans. AMS 117, 1965, 285-306.

6. Hennie, F.C., "One-tape Off-line Turing Machine Computations Inf. and Control, 8:6, 1965,553-578.

7. Hennie, F.C., "On-line Turing Machine Computations", IEEE-EC 15, 1966, 35-44.

8. Hennie, F.C. and R.E. Stearns, "Two-tape Simulation of Multitape Turing Machines", JACM 13:4, 1966, 533-546.

9. Hopcroft, J.E. and J.D. Ullman, "Some Results on Tape-Bounded Turing Machines", JACM 16, 1967, 168-177.

10. Hopcroft, J.E. and J.D. Ullman, Formal Languages and their Relation to Automata, Addison-Wesley, 1969.

11. Paterson, M.S., "Tape Bounds for Time-Bounded Turing Machines", JCSS 6, 1972, 116-124.

12. Savitch, W.J., "Relationships between Nondeterministic and Deterministic Tape Complexities", JCSS 4:2, 1970, 177-192.

13. Schönhage, A. and V. Strassen, "Schnelle Multiplikation grosser Zahlen", Computing 7, 1971, 281-222.

14 Seiferas, J.J., M.J. Fischer and A.R. Meyer, "Refinements of Nondeterministic Time and Space Hierarchies", 14th IEEE-SWAT 1973, 130-137.

15. Stearns, R.E., J. Hartmanis, and P.M. Lewis, "Hierarchies of Memory Limited Computations", IEEE-SWAT, 1965, 191-202.

16. Stoss, H.J., "Rangierkomplexitat von Permutationen", Acta Informatic 2, 1973, 80-96.

17. Strassen, V., "Gaussian Elimination is not Optimal", Numerische Mathematik 13, 1969, 354-356.

18. Valiant, L.G. and M.S. Paterson, "On the Depth of Circuits", unpublished manuscript, 1975.